Rodrigo Tomás Nogueira Cardoso

Algoritmos para Programação Dinâmica baseados em Famílias Invariantes

Dissertação de mestrado apresentada ao Departamento de Matemática do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial à obtenção do título de Mestre em Matemática.

Orientador: Ricardo H. C. Takahashi

Universidade Federal de Minas Gerais
Belo Horizonte, 18 de fevereiro de 2005

Agradecimentos

À Deus, pelo companheirismo e pela graça.

Aos meus pais, por terem me proporcionado chegar até aqui.

À minha família, pela presença.

À Cibele, pelo amor e compreensão.

Ao prof. Ricardo, pela indicação do tema, pela confiança, respeito, apoio, atenção.

Aos amigos e colegas de mestrado e de graduação, pela convivência e ajuda; por tornarem a vida mais leve!

Aos demais funcionários e professores do departamento.

Resumo

A maneira tradicional de se resolver um problema de programação dinâmica com variáveis discretas consiste em montar a "árvore de possibilidades"e procurar nela um caminho mínimo - este é o algoritmo ótimo, baseado no "princípio da otimalidade de Bellman". Tal algoritmo é de complexidade exponencial. Este trabalho propõe um método para tratar uma classe de problemas de programação dinâmica linear com variáveis discretas utilizando uma relaxação contínua nas variáveis, obtendo soluções aproximadas, com custo computacional equivalente ao da solução de um problema de programação linear estática com dimensão igual a $(n + pN)$, onde (n) é o número de "variáveis de estado"em um dos estágios, (p) é o número de "variáveis de decisão", e (N) é o número de estágios do processo de decisão. O método proposto se baseia na idéia de iterar, através do sistema dinâmico, um conjunto fechado com estrutura paramétrica invariante a essa iteração (um conjunto invariante). A otimização é feita com as variáveis de estado em apenas um dos estágios, restritas a tal conjunto, sendo utilizada a linearidade do sistema dinâmico para produzir uma composição da otimização nas variáveis de decisão com a otimização no vetor de estados.
Palavras-chave: otimização, sistemas dinâmicos, programação dinâmica, programação linear.

Abstract

The traditional way of solving a dynamic programming problem with discrete variables consists in building the ‘ ‘possibility tree”, in which a minimal path is determined - this is the optimal algorithm, based on “Bellman’s optimality principle”. Such algorithm has exponential complexity. This work proposes a method for dealing with a class of linear dynamic programming problems with discrete variables, employing a continuous relaxation that leads to approximate solutions with computational cost equivalent to the cost of solving a linear programming problem with dimension $(n+pN)$, in which (n) is the number of “state variables” in one stage, (p) is the number of ”decision variables”, and (N) is the number of stages of the decision process. The proposed method is based on the idea of iterating, through the dynamic system, a closed set with a parametric structure that is invariant under such iteration (an invariant set). The optimization is performed with the state variables just in one stage, constrained to such set, and the linearity of the dynamic system is employed in order to allow the composition of the optimization on the state variables with the optimization on the decision variables.
keywords: optimization, dynamical system, dynamical programming, linear programming.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Nesta dissertação de mestrado, proporemos algoritmos para programação dinâmica baseados em famílias invariantes. No contexto deste trabalho, resolver um problema de programação dinâmica consiste em otimizar uma função matemática, chamada de função objetivo ou função custo, restrita a um sistema dinâmico.

Um sistema dinâmico é uma função que varia no tempo[1] e que sua saída no tempo atual depende, além da entrada no tempo atual, da saída do tempo anterior. Trabalharemos aqui com sistemas dinâmicos discretos, lineares, de parâmetros concentrados, invariantes com o tempo[2].

A inspiração desta dissertação foi o artigo "*A linear dynamic system approach for cattle herd optimal shaping*" publicado no *International Journal of Systems Science* em 1997 por Takahashi et al. Este artigo apresenta a formulação da evolução de um rebanho de gado no tempo como um sistema dinâmico discreto linear e propõe uma relaxação de integralidade nas variáveis (considerá-las como números reais). Além desta, considera uma outra relaxação: propõe substituir a meta de programação[3] (que no caso era um autovetor associado ao autovalor unitário da matriz de transição do sistema) por uma bola ao seu redor e, assim, otimizar apenas no estágio inicial do problema(na aquisição do rebanho).

O artigo mostra que a pré-imagem no estágio inicial da bola em torno da meta (pela matriz de transição) é um elipsóide, portanto, um conjunto convexo. Como

[1] O parâmetro do sistema dinâmico pode ser qualquer outra grandeza (profundidade, por exemplo). Consideramos aqui o parâmetro como tempo, seguindo a definição de [Chen (1999)], que corresponde à maioria dos casos reais.

[2] Apresentaremos todas estas definições no próximo capítulo.

[3] É o ponto onde se quer chegar com a otimização dinâmica.

a otimização acontece em apenas um dos estágios, o problema de programação dinâmica é, assim, aproximado por um problema de programação não-linear convexa estática.

A partir deste resultado, generalizamos esta metodologia (também a chamaremos de formulação ou de algoritmo) para qualquer sistema dinâmico discreto linear sem a ação de controle[4]. Ou seja, qualquer sistema da forma:

$$x[k+1] = \mathrm{A}x[k],$$

$$k = 0, 1, ..., N-1,$$

no qual k é o indexador do tempo, N é o horizonte ou a quantidade total de tempo que se quer considerar, A é uma matriz e $x[k]$ é um vetor, chamado de estado do sistema no estágio k, e sintetiza a informação passada relevante para os estados futuros.

Será que esta metodologia continuaria valendo se considerássemos outro conjunto em torno da meta de programação, que não fosse uma bola? Quando valeria?

Neste trabalho, veremos que poderemos estender esta metodologia para qualquer conjunto pertencente a uma família invariante em torno da meta de programação. Uma família de conjuntos é invariante por uma transformação linear[5], ou apenas, é uma família invariante, quando a iteração por esta transformação linear de cada conjunto da família ainda é um conjunto da família.

Apresentaremos uma nova formulação para a otimização restrita à pré-imagem da meta de programação e para a pré-imagem de um hipercubo em torno da meta - duas famílias invariantes. Como estes conjuntos são descritos por equações lineares e continuaremos a otimizar apenas em um dos estágios, conseguiremos encontrar uma solução[6] para este problema dinâmico por meio de um problema estático de programação linear.

Depois, estudaremos esta metodologia para sistemas discretos lineares com a

[4]Vamos adotar esta nomenclatura, embora na teoria de controle, um sistema assim seja chamado de autônomo.

[5]No terceiro capítulo explicaremos o porque da definição de invariante por uma transformação linear.

[6]Vamos encontrar o ótimo do problema se as variáveis do problema forem números reais.

ação de controle[7]. Ou seja, sistemas da forma:

$$x[k+1] = \mathrm{A}x[k] + \mathrm{B}u[k],$$

$$k = 0, 1, ..., N-1,$$

sendo que k é o indexador do tempo, N é o horizonte ou a quantidade total de tempo que se quer considerar, A e B são matrizes, $x[k]$ é o estado do sistema no estágio de tempo k e $u[k]$ é a variável de controle ou decisão a ser tomada no estágio k.

A maneira tradicional de se resolver um problema de programação dinâmica consiste em usar o "Princípio da Otimalidade de Bellman"e, estágio por estágio, escolher qual a melhor seqüência de decisões a se tomar. Na prática, quando os estados são números discretos, o algoritmo consiste em listar todos os estados e todos os controles possíveis, montando a chamada "árvore de possibilidades", e procurar nela um caminho mínimo. Esta metodologia sempre encontra o ponto ótimo, mas a um custo computacional exponencial em função do número de estados e estágios do problema. Ou seja, o processo é intratável na prática para grande parte de problemas reais, nos quais o número de estados é muito alto.

Neste trabalho, como otimizamos em apenas um dos estágios, o número total de variáveis de estado é bem menor.

Se as variáveis do problema forem números reais, propomos considerá-lo como um problema de programação estática (linear ou não-linear) e poderemos encontrar, a um custo computacional polinomial em função das variáveis de estado, o valor do ótimo. Se quisermos aumentar a região factível do problema (para encontrar resultados melhores), sugerimos permitir uma relaxação na meta de programação (substituí-la por uma bola ou por um hipercubo, por exemplo).

Se as variáveis forem discretas (por exemplo, números inteiros), podemos considerar o problema dinâmico como um problema de programação inteira (estática). Só que, propomos fazer a relaxação linear (resolver o problema como se as variáveis fossem números reais). Assim, encontraremos uma cota inferior para o ponto ótimo, a um custo polinomial. Com a relaxação na meta, também poderemos aumentar a região factível do problema e encontrar resultados, algumas vezes, consideravelmente melhores do que os encontrados no problema sem a relaxação na meta.

[7]Geralmente, a programação dinâmica só trabalha com sistemas com ação de controle. Justificaremos a programação dinâmica com sistemas sem a ação de controle no próximo capítulo.

A metodologia proposta é, em geral, a seguinte:

1. Primeiro, permitimos fazer a relaxação na meta de programação, que é substituída por um conjunto pertencente a uma família invariante ao seu redor.

2. Depois, fazemos o retorno deste conjunto pertencente à família invariante para um dos estágios, por exemplo, o estágio inicial, usando a idéia de pré-imagem por funções contínuas.

3. Depois, se for o caso, fazemos o retorno da seqüência de ações de controle possíveis para este mesmo estágio.

4. Como o sistema é linear, a região factível será a soma da pré-imagem do conjunto pertencente à família invariante com as pré-imagens das ações de controle possíveis, interseção com as pré-imagens das demais restrições. Usando o "Princípio da Superposição", a soma daqueles conjuntos é a soma das equações que o representam.

5. Por fim, basta fazermos a otimização restrita a esta região factível para encontrarmos a solução do problema.

Esta metodologia é, portanto, muito mais simples (em termos de custo computacional) que os métodos tradicionais de programação dinâmica. Além disso, ela nos permite enxergar este problema com uma ótima geométrica, pensando na dinâmica como pré-imagens de conjuntos em vez de arcos de um grafo.

O trabalho está organizado em cinco capítulos. Este primeiro é a introdução e o último é a conclusão.

No segundo capítulo faremos a revisão bibliográfica. Como queremos apresentar os principais conceitos de sistemas dinâmicos e de programação dinâmica, esse capítulo está subdividido em duas partes. Na primeira, falaremos de sistemas dinâmicos, dando um destaque para os sistemas lineares, a matéria principal deste trabalho, e apresentando o Princípio da Superposição. Vamos também mostrar a descrição de espaço de estados, uma maneira de descrever (alguns) sistemas dinâmicos. Na segunda parte, apresentaremos a formalização do algoritmo clássico da programação dinâmica, baseado no Princípio da Otimalidade de Bellman. Vamos discutir o custo computacional deste algoritmo, bem como ilustrá-lo com um exemplo. Aí discutiremos as vantagens computacionais das metodologias por nós propostas. Falaremos também sobre programação dinâmica estocástica, tema de trabalhos futuros.

No terceiro capítulo vamos apresentar em detalhes os algoritmos para programação dinâmica baseados em famílias invariantes. Para isto, vamos primeiro definir o que são famílias invariantes. Depois, apresentaremos as abordagens: da pré-imagem da meta, de uma bola em torno da meta e de um hipercubo em torno da meta para o caso sem a ação de controle e as abordagens: da pré-imagem da meta e de um hipercubo em torno da meta para o caso com a ação de controle.

No quarto capítulo vêm os estudos de casos. Os dois primeiros são exemplos clássicos, um com variáveis reais (o pêndulo invertido) cujo resultado encontrado é o ponto ótimo, e outro com variáveis inteiras (o problema da mochila), para o qual discutiremos a relaxação linear. Não nos preocuparemos em apresentar testes em excesso, apenas os necessários para verificarmos os conceitos e as formulações apresentadas. O terceiro estudo de caso está melhor elaborado e remete ao início do trabalho: a evolução de um rebanho de gado no tempo. Além de reproduzir os resultados do trabalho de [Takahashi et al (1997)], apresentaremos testes com todas as demais formulações aqui propostas com e sem a ação de controle. Aí, verificaremos que, mesmo neste caso no qual as variáveis são números inteiros, podemos obter ótimos resultados com estas formulações a um custo computacional bem menor.

Capítulo 2

Revisão bibliográfica

Neste capítulo, vamos apresentar a fundamentação teórica do nosso trabalho: "Algoritmos para Programação Dinâmica baseados em Famílias Invariantes". Na primeira seção, apresentaremos os principais conceitos de sistemas dinâmicos e, na segunda, vamos apresentar a definição e o algoritmo clássico da programação dinâmica. Vamos, também, falar um pouco sobre o que se tem pesquisado na área e comentar sobre programação dinâmica estocástica, tema de trabalhos futuros.

2.1 Sistemas dinâmicos

Vamos, nesta seção, apresentar os conceitos básicos de sistemas dinâmicos, bem como os principais tipos de sistemas: contínuos ou discretos, de parâmetros concentrados ou distribuídos, etc. Vamos dar um destaque especial aos sistemas lineares, que constituem a matéria principal deste trabalho. Depois, vamos apresentar a descrição padrão de um sistema de parâmetros concentrados: as equações de espaço de estados para os sistemas contínuos e para os discretos.

De acordo com [Chen (1999)], podemos estudar sistemas físicos que variam no tempo (aqui chamados de sistemas dinâmicos ou sistemas) usando apenas métodos empíricos. Basta aplicarmos vários sinais no sistema e medirmos as respostas. Se o resultado obtido não estiver dentro do esperado, devemos ajustar alguns parâmetros até obtermos um satisfatório. Mas este processo pode ser complexo, caro e perigoso. Portanto, é melhor trabalharmos com métodos analíticos.

O estudo analítico de um sistema é composto por quatro partes[1]: modelagem,

[1]Colocamos assim, embora muitos autores considerem apenas três partes; a modelagem,

desenvolvimento da descrição matemática, análise e projeto. A distinção entre o sistema e seu modelo é a engenharia, ou seja, a maneira como o sistema pode ser descrito para ser melhor estudado. Depois da modelagem, aplicam-se leis da física ou outras propriedades do modelo para escrever as equações matemáticas do sistema. Por fim, vem a análise quantitativa (verificar se o resultado obtido responde ao problema original) ou qualitativa (mais interessada nas propriedades gerais).

2.1.1 Conceitos básicos

[Chen (1999)] define sistemas dinâmicos como funções (caixas-pretas) que variam no tempo e que possuem alguns parâmetros de entrada e alguns parâmetros de saída, estes unicamente determinados pelos parâmetros de entrada e pelos parâmetros de saída do tempo anterior.

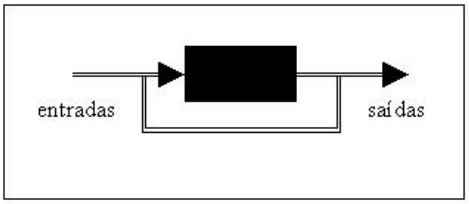

Figura 2.1: Sistema dinâmico como uma caixa-preta.

Um sistema que só possui um parâmetro (ou um terminal) de entrada e um de saída é chamado um sistema SISO (*single-input single-output*). Um sistema com mais de um terminal de entrada e de saída é chamado MIMO (*multi-input multi-output*).

Um sistema é chamado de contínuo no tempo (ou simplesmente contínuo) se aceita sinais contínuos no tempo como entrada e gera sinais contínuos no tempo como saída.

A entrada pode ser denotada por $u(t)$ para a entrada simples, e, para simplificar a notação, também para a entrada múltipla. O tempo t pode ser qualquer valor real. Se a entrada tem p terminais, então $u(t) = [u_1(t),\ u_2(t),\ ...,\ u_p(t)]^t$ é um vetor em $\mathbb{R}^p$. A variável $u(t)$ também é chamada de variável de controle ou, simplesmente, controle. Similarmente, a saída (simples ou múltipla) pode ser

neste caso, incluiria o desenvolvimento da descrição matemática.

denotada por $y(t)$. O tempo t pode ser qualquer valor real.

Um sistema é chamado sem memória se a saída $y(t_0)$ depende apenas da entrada aplicada em t_0, ou seja quando é independente da entrada aplicada antes ou depois de t_0. Entretanto, a maioria dos sistemas têm memória, ou seja, a saída em t_0 depende da entrada aplicada antes e depois de t_0 (passado e futuro), além da entrada aplicada em t_0 (tempo corrente).

Um sistema é dito causal se a saída corrente depende da entrada passada e corrente, mas não da futura. Um sistema não-causal depende da entrada futura e deve predizê-la. Nenhum sistema físico tem esta capacidade.

Definição 2.1.1 *O estado $x(t_0)$ de um sistema no tempo t_0 é a informação em t_0, tal que, junto com a entrada $u(t_0)$, determina a saída $y(t)$ para todo $t \geq t_0$. Se x é um vetor, suas coordenadas são chamadas de variáveis de estado.*

Um sistema é dito de parâmetros concentrados se possui um número finito de variáveis de estado, ou seja, o estado é um vetor (finito). O sistema é dito distribuído se tem um número infinito de variáveis de estado.

Usando o estado em um tempo t_0, podemos expressar as entradas e saídas de um sistema desta forma:

$$\left.\begin{array}{l} x(t_0) \\ u(t),\ t \geq t_0 \end{array}\right\} \rightarrow\ y(t),\ t \geq t_0.$$

Um sistema é dito invariante com o tempo (ou estacionário com horizonte ilimitado, ou apenas invariante), se para todo par de entrada-estado-saída

$$\left.\begin{array}{l} x(t_0) \\ u(t),\ t \geq t_0 \end{array}\right\} \rightarrow\ y(t),\ t \geq t_0$$

e para todo T, tivermos:

$$\left.\begin{array}{l} x(t_0+T) \\ u(t-T),\ t \geq t_0+T \end{array}\right\} \rightarrow\ y(t-T),\ t \geq t_0+T.$$

Isto significa que, se o estado inicial e a entrada são os mesmos, não importando em qual tempo eles foram aplicados, então a saída será a mesma. Por isso, quando o sistema for invariante com o tempo, podemos considerar, sem perda de generalidade, $t_0 = 0$ como tempo inicial.

Um sistema é discreto no tempo (ou simplesmente discreto) se aceita sinais discretos no tempo como entrada e gera sinais discretos no tempo como saída. Assumimos que o sistema discreto como um todo tenha um mesmo período T (ou seja, a entrada e a saída têm o mesmo período T). A entrada será denotada por $u[k] = u[kT]$, a saída será denotada por $y[k] = y[kT]$ e o estado será denotado por $x[k] = x[kT]$. A variável k é o indexador do tempo ($k = 0, 1, \ldots, N-1$) e corresponde aos estágios do problema e N é o horizonte, ou número total de estágios em que a entrada será aplicada. Para os sistemas discretos múltiplos também usaremos a mesma notação.

Pode ser que um sistema não possua variáveis de entrada. Assim, a partir de um estado inicial, as saídas são geradas como se a entrada u fosse zero para todo tempo. Chamamos este sistema de sistema sem a ação de controle e, neste caso, desconsideramos as variáveis u. No outro caso, não especificamos nada ou o chamamos de sistema com a ação de controle.

Um sistema discreto de parâmetros concentrados é dito estocástico quando consideramos a presença da aleatoriedade na sua entrada. [Bertsekas (1995)] considera, além das variáveis x (estado) e u (controle ou entrada), uma variável w que representa um parâmetro aleatório. Se o número de estados for finito, podemos considerar a probabilidade de transição $p_u(x_1, x_2)$ que é, para cada para cada par de estados (x_1,x_2) e entrada u, a probabilidade do próximo estado ser x_2 dado que o estado corrente é x_1 e a entrada é u.

O sistema é determinístico quando a transição de estados é unicamente determinada, ou seja, para cada para cada par de estados (x_1,x_2) e entrada u, a probabilidade do próximo estado ser x_2 dado que o estado corrente é x_1 e a entrada é u é 0 ou 1. Neste caso, então, podemos desconsiderar a variável w. Ao longo do texto, quando não especificarmos, estaremos considerando o sistema determinístico.

2.1.2 Sistemas dinâmicos lineares

Um sistema é linear quando para todo t_0, quaisquer dois pares de entradas-estados-saídas

$$\left.\begin{array}{l} x_i(t_0) \\ u_i(t),\ t \geq t_0 \end{array}\right\} \rightarrow\ y_i(t),\ t \geq t_0$$

com $i = 1$ ou 2 e quaisquer constantes reais α_1 e α_2, temos que:

$$\left.\begin{array}{l} \alpha_1 x_1(t_0) + \alpha_2 x_2(t_0) \\ \alpha_1 u_1(t) + \alpha_2 u_2(t),\ t \geq t_0 \end{array}\right\} \rightarrow\ \alpha_1 y_1(t) + \alpha_2 y_2(t),\ t \geq t_0.$$

Se a entrada $u(t)$ é identicamente nula para todo $t \geq t_0$, então a saída pode ser obtida exclusivamente em função do estado inicial $x(t_0)$. Do mesmo modo, se o estado inicial é zero, obtemos a saída exclusivamente em função da entrada. Ou seja, o sistema é separável. Assim, pela linearidade do sistema, temos que:

Princípio 2.1.1 (Princípio da Superposição)

$$\textit{saída de} \begin{cases} x(t_0) \\ u(t),\ t \geq t_0 \end{cases} = \textit{saída de} \begin{cases} x(t_0) \\ u(t) = 0,\ t \geq t_0 \end{cases} + \textit{saída de} \begin{cases} x(t_0) = 0 \\ u(t),\ t \geq t_0 \end{cases}$$

Portanto, as duas respostas podem ser estudadas separadamente. O resultado final será a soma destas duas respostas.

2.1.3 Descrição de espaço de estados

Todo sistema dinâmico contínuo, determinístico, de parâmetros concentrados pode ser descrito por um sistema de equações da forma:

$$\dot{x}(t) = f(x(t), u(t)) \tag{2.1}$$
$$y(t) = g(x(t), u(t)) \tag{2.2}$$

com $\dot{x}(t) = dx/dt$.

Estas duas equações (a primeira, diferencial e a segunda, algébrica) são chamadas no contexto da teoria de controle como equações de espaço de estados ou simplesmente equações de estado[2].

Todo sistema contínuo, linear, determinístico, de parâmetros concentrados pode ser descrito por um sistema de equações desta forma:

$$\dot{x}(t) = \mathrm{A}(t)x(t) + \mathrm{B}(t)u(t) \tag{2.3}$$
$$y(t) = \mathrm{C}(t)x(t) + \mathrm{D}(t)u(t) \tag{2.4}$$

Para um sistema com p entradas, q saídas e n variáveis de estado, $u \in \mathbb{R}^p$, $y \in \mathbb{R}^q$ e $x \in \mathbb{R}^n$. Portanto, as matrizes $\mathrm{A}(t)$, $\mathrm{B}(t)$, $\mathrm{C}(t)$ e $\mathrm{D}(t)$ são $n \times n$, $n \times p$, $q \times n$ e $q \times p$, respectivamente, para cada t.

[2]No contexto da programação dinâmica, a saída y é associada à função custo e a primeira equação é conhecida como equação de transição de estados.

Se o sistema é invariante com o tempo, então (2.3) e (2.4) podem ser escritas como:

$$\dot{x}(t) = \mathrm{A}x(t) + \mathrm{B}u(t) \tag{2.5}$$
$$y(t) = \mathrm{C}x(t) + \mathrm{D}u(t) \tag{2.6}$$

e neste caso A, B, C e D são matrizes constantes (com a respectiva dimensão já descrita).

Se o sistema dinâmico determinístico, de parâmetros concentrados, for discreto então (2.1) e (2.2) ficam desta forma:

$$x[k+1] = f(x[k], u[k]) \tag{2.7}$$
$$y[k] = g(x[k], u[k]) \tag{2.8}$$
$$k = 0, 1, \ldots, N-1$$

sendo que: k é o indexador do tempo, N é o horizonte, ou número de tempos ou estágios em que o controle será aplicado.

Todo sistema discreto, determinístico, de parâmetros concentrados, linear pode ser descrito por:

$$x[k+1] = \mathrm{A}[k]x[k] + \mathrm{B}[k]u[k] \tag{2.9}$$
$$y[k] = \mathrm{C}[k]x[k] + \mathrm{D}[k]u[k] \tag{2.10}$$
$$k = 0, 1, \ldots, N-1$$

Do mesmo modo, para um sistema com p entradas, q saídas e n variáveis de estado, $u \in \mathbb{R}^p$, $y \in \mathbb{R}^q$ e $x \in \mathbb{R}^n$. Portanto, as matrizes A[k], B[k], C[k] e D[k] são $n \times n$, $n \times p$, $q \times n$ e $q \times p$, respectivamente, para cada k.

Se o sistema é invariante com o tempo, então (2.9) e (2.10) podem ser escritas como:

$$x[k+1] = \mathrm{A}x[k] + \mathrm{B}u[k] \tag{2.11}$$
$$y[k] = \mathrm{C}x[k] + \mathrm{D}u[k] \tag{2.12}$$
$$k = 0, 1, \ldots, N-1$$

e neste caso A, B, C e D são matrizes constantes.

Se o sistema for sem a ação de controle, então (2.11) e (2.12) podem ser escritas como:

$$x[k+1] = \mathrm{A}x[k] \tag{2.13}$$
$$y[k] = \mathrm{C}x[k] \tag{2.14}$$
$$k = 0, 1, \ldots, N-1$$

com A e C matrizes constantes.

Se o sistema for discreto, estocástico, de parâmetros concentrados, linear e invariante com o tempo suas equações de estado são:

$$x[k+1] = \mathrm{A}x[k] + \mathrm{B}u[k] + \mathrm{R}w[k] \tag{2.15}$$
$$y[k] = \mathrm{C}x[k] + \mathrm{D}u[k] + \mathrm{S}w[k] \tag{2.16}$$
$$k = 0, 1, \ldots, N-1$$

e A, B, C, D, R e S são matrizes constantes.

0 enfoque principal deste trabalho é propor métodos para programação dinâmica cujos sistemas dinâmicos são determinísticos, discretos, lineares, de parâmetros concentrados, invariantes com o tempo. Apenas na seção 2.2.3 falaremos sobre sistemas estocásticos.

2.2 Programação dinâmica

A programação dinâmica é uma poderosa técnica de otimização de sistemas dinâmicos. Ela foi criada por Richard Bellman, que publicou seu livro "Dynamic Programming"em 1957. Trabalhos posteriores, dele próprio e de outros, mostram a importância do tema e suas numerosas aplicações.

A programação dinâmica trabalha com situações em que as decisões são tomadas em estágios, ou em etapas seqüenciais e a resposta, ou a decisão em cada estágio, influencia na decisão a ser tomada no estágio seguinte. Em cada estágio, as decisões são ordenadas baseado na soma do custo presente e do custo futuro esperado, assumindo que a decisão ótima está sendo tomada em cada estágio subseqüente.

Como a programação dinâmica trabalha com a tomada de decisões em estágios, buscando, em geral, a seqüencia de decisões que tenham um custo mínimo, vamos, neste trabalho, tratá-la como a minimização de uma função matemática, chamada de função objetivo ou função custo, restrita a um sistema dinâmico.

Estamos usando a definição de sistemas dinâmicos de [Chen (1999)], que considera o tempo como parâmetro do sistema. Esta definição não perde a sua generalidade caso o parâmetro seja outra grandeza.

2.2.1 Formulação do problema e algoritmo clássico

Na seção anterior, apresentamos as equações de estado para diversos tipos de sistemas dinâmicos. Para cada tipo, apresentamos duas equações: a primeira, referente à transição dos estados e a segunda, referente à saída do sistema. No contexto da programação dinâmica, vamos associar uma função custo à saída do sistema dinâmico. Assim, vamos caracterizar um sistema pela sua função custo e pela primeira equação da descrição de espaço de estados, correspondente à equação de transição dos estados.

Considere um sistema dinâmico determinístico, discreto, linear, de parâmetros concentrados, invariante com o tempo, como em (2.11).

Ao estágio k, associamos o custo $g[k](x[k], u[k])$, que representa o custo de transição do estado $x[k]$ para o estado $x[k+1]$ sob o controle $u[k]$.

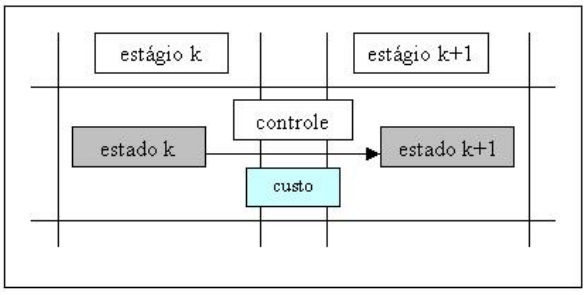

Figura 2.2: Diagrama mostrando a transição do estado no estágio k para o estado no estágio $k+1$, via controle (com seu custo associado.)

Geralmente, a otimização é sobre os controles $u[0], \ldots, u[N-1]$ e cada controle $u[k]$ é selecionado a partir da informação obtida no estado corrente $x[k]$.

Podemos otimizar também sobre um dos estados. Por exemplo, dado um estado no estágio final $x[N]$, podemos procurar qual o melhor estado no estágio inicial $x[0]$, que atinja pelo sistema dinâmico este estado final dado (ou vice-versa)[3].

Considere uma política (também chamada leis de controle ou estratégia), que consiste numa seqüência de funções:

$$\pi = \{\mu[0], \ldots, \mu[N-1]\},$$

na qual $\mu[k]$ mapeia $x[k]$ na variável de controle $u[k] = \mu[k](x[k])$.

Dado um estado inicial $x[0]$, uma política π é chamada admissível se $\mu[k](x[k]) = u[k]$ existe, para cada estágio k.

A função custo é aditiva, no sentido que o custo do estágio k, denotado por $g[k](x[k], u[k])$ é acumulativo com o tempo. Seja $g[N](x[N])$ o custo do estágio final.

Dado um estado inicial $x[0]$, o custo associado a uma política admissível π está bem definido:

$$J_\pi(x[0]) = g[N](x[N]) + \sum_{k=0}^{N-1} g[k](x[k], \mu[k](x[k])). \tag{2.17}$$

Para um dado estado inicial $x[0]$, uma política ótima π^* é a que minimiza este custo, ou seja:

$$J_{\pi^*}(x[0]) = \min_{\pi \in \Pi} \; J_\pi(x[0]) \tag{2.18}$$

sendo que Π é o conjunto de todas as políticas admissíveis.

Note que a política ótima, definida assim, depende do estado inicial. Entretanto, é possível definir a política que seja simultaneamente ótima para qualquer estado inicial:

$$J^* = \min_{\pi \in \Pi \text{ e } x[0] \in S} \; J_\pi(x[0]) \tag{2.19}$$

com S denotando o conjunto dos estados iniciais possíveis. Esta é a formulação clássica do problema de programação dinâmica e esta equação (2.19) é chamada

[3]Quando o sistema é sem a ação de controle como em (2.13), a otimização só pode ser sobre os estados.

equação funcional da programação dinâmica ou **equação de Bellman**.

[Bertsekas (1995)] apresenta o algoritmo clássico da programação dinâmica. Este algoritmo se baseia no princípio da otimalidade de Bellman.

Princípio 2.2.1 (Princípio da Otimalidade de Bellman) *Seja uma política ótima* $\pi^* = \{\mu[0]^*, \ldots, \mu[N-1]^*\}$ *para o problema de programação dinâmica. Considere o subproblema segundo o qual estamos em* $x[i]$ *no tempo* i *e queremos minimizar o custo de um tempo* i *até* N*:*

$$g[N](x[N]) + \sum_{k=i}^{N-1} g[k](x[k], \mu[k](x[k])).$$

Então, a política truncada $\{\mu[i]^*, \mu[i+1]^* \ldots, \mu[N-1]^*\}$ *é ótima para o subproblema.*

A justificativa para o princípio da otimalidade é muito simples. Se a política truncada não é ótima, podemos reduzir o custo total considerando esta política truncada ótima para cada subproblema (referente a cada $x[i]$). Para um exemplo com tráfego: se o caminho mais curto entre Belo Horizonte e a cidade do Rio de Janeiro passar por Juiz de Fora, então a porção da rota entre Juiz de Fora e o Rio tem que ser o caminho ótimo entre Juiz de Fora e o Rio.

O algoritmo da programação dinâmica é baseado nesta idéia. Considere $U[k](x[k])$ o conjunto de todas as entradas possíveis no estágio k sujeitas ao estado $x[k]$. Vamos definir a seguinte função recursiva:

$$\begin{cases} J[N](x[N]) = g[N](x[N]). \\ J[k](x[k]) = \min\limits_{u[k] \in U[k](x[k])} g[k](x[k], u[k]) + J[k+1](x[k+1]), \\ k = 0, 1, \ldots N-1. \end{cases} \tag{2.20}$$

Apresentamos o algoritmo num teorema:

Teorema 2.2.1 (Algoritmo clássico da programação dinâmica) *Para todo estado inicial* $x[0]$*, o custo ótimo* $J^*(x[0])$ *é igual a* $J[0](x[0])$*. A função* $J[0]$ *é dada pelo último passo de (2.20), que vai retroativamente do tempo N-1 até o tempo 0. Portanto, se* $u^*[k] = \mu^*[k](x[k])$ *minimiza (2.20) para cada* $x[k]$ *e* k*, então a política* $\pi^* = \{\mu[0]^*, \ldots, \mu[N-1]^*\}$ *é ótima.*

DEMONSTRAÇÃO: [Mascó (2001)] apresenta a dedução deste algoritmo para o caso determinístico (como formulado acima). [Bertsekas (1995)] apresenta a demonstração para o caso geral (estocástico). ■

Até aqui, não fizemos nenhuma hipótese sobre a função custo J. Se ela for linear, bem como o sistema dinâmico em questão, o problema é de programação dinâmica linear.

Só que em alguns casos, a função custo do problema é quadrática. [Phillips & Nagle (1995)] apresentam a formulação do problema de programação dinâmica para este caso. Eles consideram como função custo:

$$J[N] = \sum_{k=0}^{N} \left(x^t[k]\mathrm{Q}[k]x[k] + u^t[k]\mathrm{R}[k]u[k] \right)$$

com N finito e Q e R matrizes simétricas positivas semi-definidas[4]. O algoritmo apresentado é o mesmo já citado acima (Teorema 2.2.1), baseado no Princípio da Otimalidade de Bellman (princípio **??**).

2.2.2 Análise da complexidade do algoritmo

Na prática, quando as variáveis do problema são números inteiros, a solução de um problema de programação dinâmica consiste em duas etapas: montar a árvore de possibilidades e pesquisar nela o caminho mínimo. Quando as variáveis do problema são números reais, a resolução do problema pelo Princípio de Bellman pede que se aproxime as variáveis contínuas por variáveis discretas, a fim de listá-las e montar a árvore de possibilidades.

A complexidade ou custo computacional de um algoritmo é uma função matemática que permite compará-lo com os demais. Em geral, procura-se caracterizar a eficiência de um algoritmo estudando o tempo máximo de computação necessário para a resolução do problema em função de seu tamanho. A bibliografia sobre o assunto é extensa, mas citamos apenas [Goldbarg & Luna (2000)].

Vamos argumentar, a seguir, que a montagem da árvore de possibilidades tem custo computacional exponencial (ou seja, o tempo de computação está limitado por uma exponencial em função do tamanho do problema) e a pesquisa do caminho mínimo pode ser feita em tempo polinomial (o tempo máximo é um

[4]Falaremos sobre as matrizes simétricas positivas semi-definidas no próximo capítulo.

polinômio em função do tamanho do problema).

É claro que um tempo exponencial é algo ruim, pois a função exponencial cresce muito rapidamente, assim, um problema que necessite de um número exponencial de operações fica de uso drasticamente limitado em computadores atuais.

Se o número de estados possíveis for finito, podemos representar o problema de programação dinâmica num grafo, no qual os arcos correspondem às transições entre estados consecutivos com o respectivo custo associado. Este grafo é chamado de **árvore de possibilidades**. Pode-se adicionar ao final do grafo um nodo artificial t com cada estado $x[N]$ do estágio N conectado a ele com o custo $g[N](x[N])$. A seqüência de controle corresponde ao caminho ótimo começando neste nodo artificial t e terminando num nodo inicial i referente ao estágio 0.

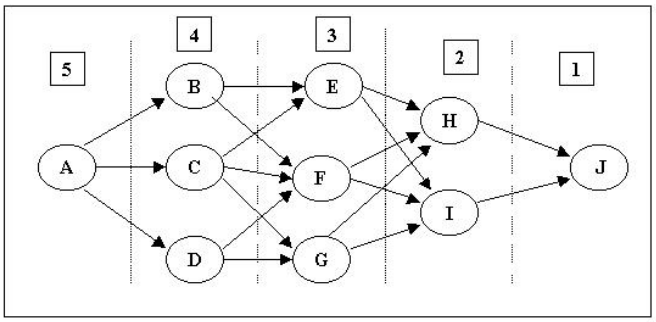

Figura 2.3: Exemplo de uma árvore de possibilidades (os quadrados são os estágios, as bolas são os estados e as flechas são os controles possíveis).

Só que montar esta árvore de possibilidades é, em geral, impraticável, exceto em problemas de baixa dimensão, dada a explosão combinatória de métodos enumerativos. Por exemplo, admitindo que exista um único estado inicial em um problema unidimensional (variáveis de estado com uma dimensão) com N estágios e que, para cada estado, existam M decisões possíveis. Disto resulta que o número de trajetórias possíveis é M^N, ou seja, a montagem da árvore **tem complexidade computacional exponencial**. Se forem $N = 30$ estágios e $M = 2$ decisões, teremos que o número de nodos do grafo será da ordem de 10^9, praticamente impossível de se montar a um custo razoável, exceto em supercomputadores.

Tendo montado a árvore de possibilidades e considerando o custo do arco como comprimento, podemos terminar de resolver o problema de programação dinâmica procurando nesta árvore o caminho mínimo. A seqüência do caminho seria a seqüência de controles. Como o grafo não é direcionado, o caminho mínimo é reverso, ou seja, poderíamos otimizar também do nodo i para o nodo t.

Um algoritmo tradicional de caminho mínimo, o *Dijkstra*, tem complexidade polinomial da ordem de n^2, ou $O(n^2)$, sendo que n é o número de nodos do grafo (ver [Goldbarg & Luna (2000)]). Assim, no caso da programação dinâmica, encontrar o caminho mínimo pelo algoritmo (2.2.1) proposto acima tem complexidade computacional $O(n^2)$, sendo que n é o número total de estados em todos os estágios.

A pesquisa nesta área consiste em encontrar formas alternativas para se montar o grafo (árvore de possibilidades), que é o que tem custo computacional exponencial. Basicamente, os aspectos envolvidos na pesquisa são técnicas de desenvolvimento da árvore de enumeração do problema, técnicas de formação desta árvore e técnicas complementares para obtenção de limites. [Goldbarg & Luna (2000)] citam algumas destas técnicas.

Uma técnica muito empregada na solução de problemas de programação dinâmica é a chamada *Branch-and-Bound*. Ela baseia-se na idéia de desenvolver uma enumeração inteligente dos pontos candidatos à solução ótima de um problema. O objetivo é reduzir o tamanho da árvore de possibilidades, "podando"os nodos cuja função custo não produzam acréscimo no critério de otimização. O termo *branch* refere-se ao fato de que o método efetua partições no espaço das soluções e o termo *bound* ressalta que a prova da otimalidade da solução utiliza de limites calculados ao longo da enumeração. Maiores detalhes sobre o método *Branch-and-Bound* encontramos, por exemplo, em [Garfinkel & Nemhauser (1972)].

Outra estratégia é a utilização de heurísticas ou aproximações para o problema. As heurísticas, em geral, pretendem a solução de problemas específicos e não são, via de regra, utilizáveis em outros problemas. Algumas heurísticas clássicas são (ver [Goldbarg & Luna (2000)]): redes neuronais, algoritmos genéticos, algoritmos gulosos míopes, relaxação lagrangeana e relaxação linear, que é um dos assuntos deste trabalho.

A discussão anteriormente apresentada mostra que, quando o número de estados possíveis aumenta, o esforço computacional necessário para se montar a

árvore de possibilidades do problema cresce exponencialmente. Neste trabalho, propomos uma formulação que **evita a montagem da árvore** (ou seja, que evita listar todos os estados possíveis).

Nossa formulação será exata quando o vetor de estados for uma variável contínua (com infinito número de estados possíveis) e quando as variáveis forem inteiras, sugerimos a relaxação linear, de modo que nossa formulação fornecerá uma cota inferior para o valor da função objetivo (podemos encontrar o ótimo se quisermos resolver pela formulação aqui proposta como um problema de programação inteira).

A idéia que propomos aqui consiste em otimizar apenas em um dos estágios (por exemplo, o estágio inicial), e assim, trabalhar com as variáveis de estado apenas em um dos estágios. Conseguimos isto, olhando para o problema com uma ótica geométrica. Em vez de colocar a dinâmica do sistema nos arcos de um grafo representando a transição de estados, vamos colocá-la na pré-imagem de conjuntos. Propomos considerar a pré-imagem de cada estado em um dos estágios pensado como pré-imagem de um conjunto por meio de uma função linear contínua (em vez de discreta). No próximo capítulo explicaremos melhor esta idéia.

Como mostraremos, esta formulação tem **complexidade computacional polinomial** da ordem de $p + n$, sendo que p é o número de controles possíveis e n o número de estados no estágio inicial.

Um exemplo numérico

A fim de fixar melhor os conceitos, apresentamos um exemplo da solução de um problema de programação dinâmica pela maneira clássica. Este exemplo está em [Goldbarg & Luna (2000)].

Como visto, a primeira etapa da resolução do problema consiste em listarmos todos os estados e controles possíveis para todos os estágios (montar a árvore de possibilidades), com os respectivos custos de transição.

Vamos considerar a árvore da figura (2.3), com os custos de transição de estados (associados aos controles) da tabela (2.1). Os nodos de origem estão na vertical e os nodos de destino estão na horizontal. Os "traços"correspondem à ausência de caminho entre os nodos.

	A	B	C	D	E	F	G	H	I	J
A	-	4	5	3	-	-	-	-	-	-
B	-	-	-	-	8	6	-	-	-	-
C	-	-	-	-	5	8	7	-	-	-
D	-	-	-	-	-	9	11	-	-	-
E	-	-	-	-	-	-	-	6	9	-
F	-	-	-	-	-	-	-	10	12	-
G	-	-	-	-	-	-	-	13	3	-
H	-	-	-	-	-	-	-	-	-	8
I	-	-	-	-	-	-	-	-	-	5
J	-	-	-	-	-	-	-	-	-	-

Tabela 2.1: Custos de transição de estados para a árvore da figura (2.3).

Agora, basta encontrarmos o caminho mínimo na árvore, procurando-o por estágios, usando o Princípio da Otimalidade de Bellman (como no algoritmo).

A primeira fase na solução do problema corresponde ao último estágio. A figura (2.4) ressalta as possibilidades para a política ótima de chegada aos nodos H e I, a partir de J, considerando que esta porção do caminho também será ótima.

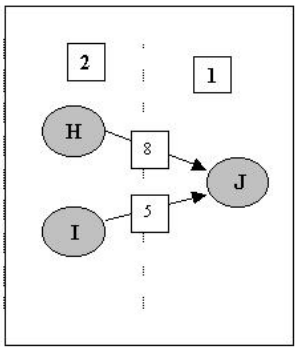

Figura 2.4: Primeira fase de decisão - política para os nodos H e I.

Com a tomada de decisão realizada na fase 1, podemos, a partir da política ótima já escolhida para H e I, calcular a política ótima para E, F e G conforme a figura (2.5).

A partir dos nodos E, F e G podemos desenvolver a terceira fase de decisão. A figura (2.6) mostra que as informações da primeira fase (nodos H e I) não são

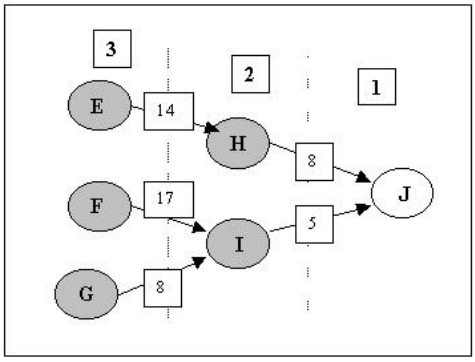

Figura 2.5: Segunda fase de decisão - política para os nodos E, F e G.

mais importantes para isso.

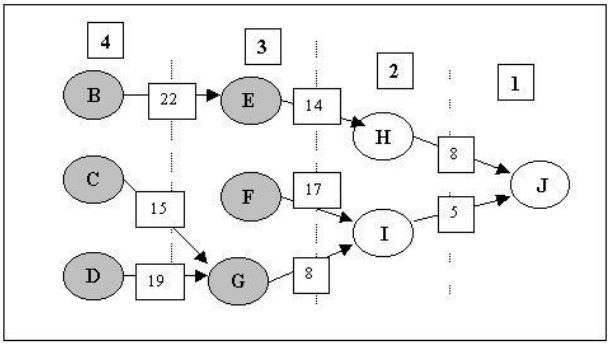

Figura 2.6: Terceira fase de decisão - política para os nodos B, C e D.

Finalmente, podemos concluir os cálculos determinando a política ótima para o nodo A, como na figura (2.7).

O caminho de custo mínimo, portanto, a seqüência dos controles ótimos, corresponde aos nodos A - C - G - I - J. Veja na figura (2.8). □

2.2.3 Programação dinâmica estocástica

Vamos fazer uma pausa para considerarmos problemas de programação dinâmica cujos sistemas são estocásticos, lineares, discretos, de parâmetros concentrados e invariantes com o tempo. Um sistema estocástico é aquele que considera o efeito

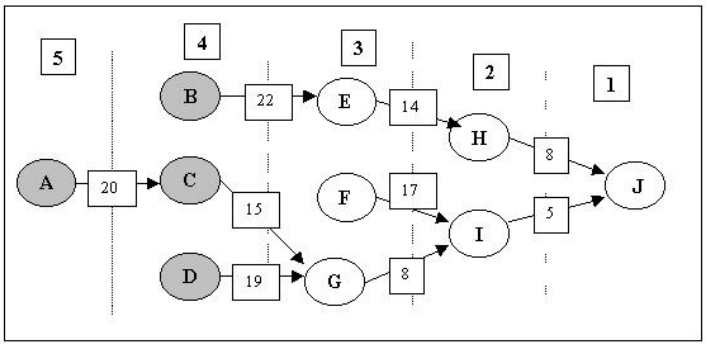

Figura 2.7: Quarta fase de decisão - política para o nodo A.

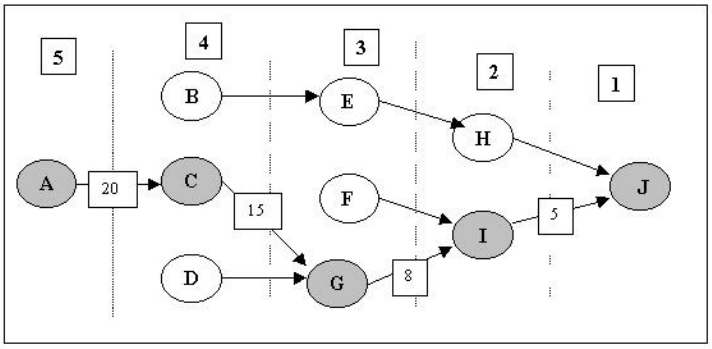

Figura 2.8: Caminho de custo mínimo na árvore de possibilidades.

da aleatoriedade na transição dos estados.

Neste caso, podemos expressar o sistema dinâmico como em (2.15):

$$x[k+1] = \mathrm{A}x[k] + \mathrm{B}u[k] + \mathrm{R}w[k],$$

sendo que A, B e R são matrizes constantes, $w[k]$ é uma variável que representa um parâmetro aleatório no estágio k.

Considere uma política $\pi = \{\mu[0], \mu[1], ..., \mu[N-1]\}$, para a qual $\mu[k]$ mapeia $x[k]$ na variável de controle $u[k]$, ou seja, $u[k] = \mu[k](x[k])$.

Considere o caso com horizonte finito (o número de estágios atinge um valor máximo N). Vamos chamar $g[k](x[k], \mu[k](x[k]), w[k])$ apenas de $g[k]$, o custo do estágio k, e vamos chamar $g[N](x[N])$ por $g[N]$, o custo do estágio final.

Como no caso determinístico, para um estado inicial $x[0]$ e uma política π a função de custo está bem definida e é a seguinte:

$$J_\pi(x[0]) = g[N] + \mathop{\mathrm{E}}_{\mathrm{w[k]}} \left[\sum_{k=0}^{N-1} g[k] \right] \tag{2.21}$$

na qual $\mathop{\mathrm{E}}_{\mathrm{w[k]}}[.]$ é o valor esperado em relação à distribuição da variável aleatória $w[k]$.

[Bertsekas (1995)] mostra que, neste caso, o algoritmo clássico da programação dinâmica (teorema 2.2.1) é ótimo quando consideramos a função custo acima.

O algoritmo é o mesmo, com a seguinte função de recursão:

$$\begin{cases} J[N](x[N]) = g[N]. \\ \\ J[k](x[k]) = \min\limits_{u[k] \in U[k](x[k])} \mathop{\mathrm{E}}_{\mathrm{w[k]}} \left[g[k] + J[k+1](x[k+1]) \right], \\ \\ k = 0, 1, \ldots N-1. \end{cases} \tag{2.22}$$

Quando o número de estados é finito, o sistema é mais convenientemente especificado em termos das probabilidades de transição entre os estados. Assim, precisamos saber a probabilidade de transição $p_u(i,j)$, que representa, para cada par de estados (i,j) e controle u, a probabilidade do próximo estado ser j dado que o estado corrente é i e o controle corrente é u.

Tal sistema pode ser descrito alternativamente como:

$$x[k+1] = w[k] \tag{2.23}$$

e a distribuição de probabilidade do parâmetro aleatório $w[k]$ é:

$$P(w[k] = j; x[k] = i, u[k] = u) = p_u(i,y).$$

Conseqüentemente, dado um sistema da forma $x[k+1] = Ax[k] + Bu[k] + Rw[k]$ com a distribuição de probabilidade P de $w[k]$, podemos encontrar uma descrição equivalente da probabilidade de transição:

$$p_u(i,j) = P(W[k](i,u,j);\ x[k] = i, u[k] = u),$$

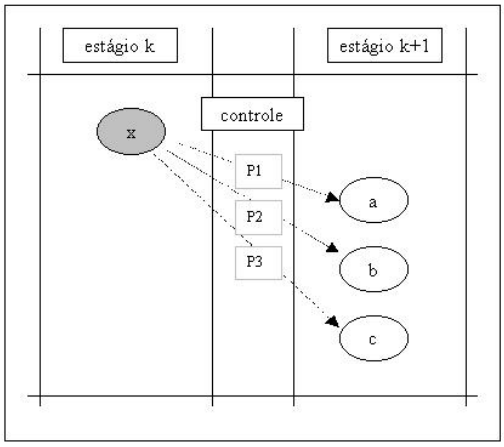

Figura 2.9: Diagrama mostrando a transição de estados com a presença da aleatoriedade, no qual $P1$,$P2$ e $P3$ são as probabilidades de transição do estado x para os estados a, b e c, respectivamente.

com $W[k](i,u,j) = \{w[k];\ j = Ai + Bu + Rw[k]\}$.

Assim, o sistema estocástico, linear, de parâmetros concentrados, invariante com o tempo, de estados discretos pode ser descrito como em (2.15) ou como em (2.23). Dependendo do problema, é mais conveniente usar uma ou outra descrição.

Seja $g[k](i,u)$ o custo esperado no estágio k do estado i ao ser aplicado o controle u. [Bertsekas (1995)] mostra que, também para a descrição em termos das probabilidades de transição, a função custo por estágio $J[k]$ (do passo do algoritmo clássico) está bem definida e é a seguinte:

$$J[k](i) = \min_{u \in U(i)} \left(g[k](i,u) + \sum_j p_u(i,j) J[k+1](j) \right). \tag{2.24}$$

Quando o horizonte é infinito, k pode assumir qualquer valor inteiro (não existe um N máximo). Apesar disso, assumimos que o número de estados seja finito (problema de parâmetros concentrados).

Considere a política $\pi = \{\mu[0], \mu[1], ...\}$. Seja $g[k]$ o custo do estágio k.

Como no caso com horizonte limitado, para um estado inicial $x[0]$ e uma política π a função de custo para o horizonte ilimitado também está bem definida

e é a seguinte:

$$J_\pi(x[0]) = \lim_{N\to\infty} \underset{\text{w[k]}}{\text{E}} \left[\sum_{k=0}^{N-1} \alpha^k g[k]\right] \tag{2.25}$$

sendo que α é um escalar que representa o fator de desconto: $\alpha \in (0, 1]$.

Quando $\alpha = 1$, como o número de estados é finito, podemos supor que exista um estado terminal t, ou seja que $p_u(t, t) = 1$ e $g[t](t, u) = 0$ para todo $u \in$ U, sendo que U é o conjunto de todos os controles admissíveis. Então, a essência do problema é chegar neste estágio final t com o mínimo custo. Chamamos, portanto, este problema de problema de caminho mínimo estocástico. O caminho mínimo determinístico se torna um caso especial deste, quando para cada par de estado-controle (x, u) existe um único estado j tal que $p_u(x, j) = 1$.

Então, neste caso, podemos gerar o custo ótimo no estágio N pelo algoritmo clássico da programação dinâmica com a função de recursividade:

$$J[k+1](x[0]) = \min_{u\in U} \underset{\text{w[k]}}{\text{E}} \left[g[k+1] + J[k](x[k])\right], \ \forall k \tag{2.26}$$

começando de uma condição inicial de custo nulo $J[0](x[0]) = 0$, para todo $x[0]$ (note que, neste caso, começamos do estágio 0 para frente, ao contrário do proposto no algoritmo).

O horizonte infinito é, por definição, o limite do custo referente ao estágio N, quando N tende a infinito:

$$J^*(x[0]) = \lim_{N\to\infty} J[N](x[0]), \ \forall x[0]. \tag{2.27}$$

De (2.26) e (2.27), podemos tirar que:

$$J^* = \min_{u\in U \text{ e } x[0]\in S} J^*(x[0]) \tag{2.28}$$

sendo que S é o conjunto de todos os estados iniciais possíveis.

Esta equação funcional em J^* é conhecida como **equação estocástica de Bellman**. Compare com a equação (2.19). Ela não é um algoritmo, mas um sistema de equações (uma por estado), que tem como solução a função custo para

todos os estados.

Uma política estacionária $\mu = \{\mu, \mu, \mu, ...\}$ é ótima se o custo associado a ela $J_\mu(x[0])$ for igual a $J^*(x[0])$ para todo estado inicial $x[0]$.

[Bertsekas (1995)] mostra que a equação estocástica de Bellman possui um único ponto fixo J^* que é o mínimo de $J_\pi(x[0])$ $\forall\pi$ e $\forall x[0]$, ou seja, que existe uma política ótima μ e que ela é estacionária, pois J^* é ponto fixo. [Bertsekas (1995)] apresenta também um algoritmo para se encontrar esta política, que nada mais é que o algoritmo clássico da programação dinâmica, considerando na função custo as probabilidades de transição.

Vamos analisar agora o problema de desconto, o caso $0 < \alpha < 1$. [Bertsekas (1995)] mostra que este problema pode ser convertido em um problema de caminho mínimo estocástico. A idéia é simples. Sejam os estados $i = 1, ..., n$ e considere um problema de caminho mínimo estocástico associado envolvendo estes estados e um estado terminal extra t, com transições de estado e custo como seguinte: o custo de se aplicar o controle u num estado $i \neq t$ é $g(i, u)$ e o próximo estado é j com probabilidade $\alpha p_u(i, j)$ e t com probabilidade $1 - \alpha$.

Suponha, agora, o uso da mesma política no problema de desconto e no problema de caminho mínimo estocástico associado. Então, enquanto o estado final não ocorre, a evolução dos estados nos dois problemas é governada pela mesma probabilidade de transição e o custo esperado no estágio k é o custo de problema de caminho mínimo estocástico associado multiplicado pela probabilidade do estado t não ter sido alcançado, que é α^k. Este é o custo esperado do estado k no problema de desconto.

Como no caso determinístico, mostramos que existe na programação dinâmica estocástica (com horizonte limitado e ilimitado) um algoritmo que garanta o ótimo, mas este algoritmo é impraticável, pois o número de variáveis cresce exponencialmente com o número de estágios do problema. Uma linha de pesquisa nesta área procura boas aproximações para a equação estocástica de Bellman, de modo que se torne praticável pelo menos a obtenção de uma estimativa da solução ótima.

Como exemplo, vamos relatar a aproximação estudada por [Farias & Van Roy (2003)]. Eles propõem resolver uma aproximação por meio da programação linear.

Para isto, considere P_u uma matriz cujas entradas são as probabilidades $p_u(i, j)$. Considere a equação (2.24), agora com o fator de desconto α. Para

simplificar a notação, vamos definir os operadores:

$$T_u J = g_u + \alpha P_u J$$

e

$$TJ = \min_{u \in U} (g_u + \alpha P_u J) = \min_{u \in U} (T_u J).$$

Como vimos, resolver um problema de programação dinâmica consiste em procurar um ponto fixo, solução da equação de Bellman: $J^* = TJ^*$.

Considere um vetor c com componentes positivas, chamado de vetor dos pesos da relevância dos estados. Vamos trabalhar com o dual da equação de Bellman:

$$\max \ c^t J \tag{2.29}$$

$$\text{sujeito a: } TJ \geq J$$

Pode-se mostrar que se J é factível, $J \leq J^*$; portanto, para todo vetor c, J^* é única solução de (2.29). O operador T não é linear, mas como $TJ \geq J$, então $TJ(x) \geq J(x)$ e daí segue o que [Farias & Van Roy (2003)] chamam de problema exato de programação linear:

$$\max \ c^t J \tag{2.30}$$

$$\text{sujeito a: } g_u(x) + \alpha \sum_{y \in S} P_u(x, y) J(y) \geq J(x) \ \ \forall x \in \text{S e } \forall u \in U.$$

sendo S o conjunto dos estados iniciais possíveis.

Infelizmente, como já comentamos, este problema sofre da "maldição da dimensionalidade", ou seja, o número de variáveis e de restrições aumentam exponencialmente com o número de estágios. Este problema é, portanto, intratável, ou é impraticável encontrar a solução deste problema na maioria dos casos.

[Farias & Van Roy (2003)] propuseram uma aproximação para (2.30) que reduz de modo drástico o número de variáveis e restrições e, portanto, o torna tratável. Eles sugerem aproximar $J : S \to \mathbb{R}$ por $\tilde{J} : S \times \mathbb{R}^k \to \mathbb{R}$, num espírito similar à regressão estatística.

Para isso, consideremos um vetor de parâmetros $r \in \mathbb{R}^k$. Encontrar $\tilde{J}$ é muito difícil e depende do tipo de problema. O foco do trabalho deles é computar os parâmetros para classes de funções linearmente parametrizadas $\Phi_k : S \to \mathbb{R}$: $\tilde{J} = \sum_{k=1}^{K} r_k \Phi_k$.

Definimos a matriz $\Phi = [\Phi_1 \dots \Phi_K]$. Queremos $\tilde{r} \in \mathbb{R}^k$ tal que $\Phi\tilde{r}$ seja uma boa aproximação para J^*. Assim, apresentamos o chamado problema aproximado de programação linear:

$$\max \; c^t \Phi r \tag{2.31}$$

$$\text{sujeito a: } g_u(x) + \alpha \sum_{y \in S} P_u(x,y)(\Phi r)(y) \geq (\Phi r)(x) \;\; \forall x \in S \text{ e } \forall u \in U.$$

Este problema aproximado tem k variáveis e, embora o número de restrições seja o mesmo do problema exato, muitas delas são inativas. [Farias & Van Roy (2003)] mostram que, de fato, o problema aproximado converge para o ótimo, e apresentam uma cota máxima para o erro.

Esta seção trouxe apenas os principais conceitos da programação dinâmica estocástica e um rápido comentário sobre o trabalho de [Farias & Van Roy (2003)]. Nas próximas páginas, voltaremos a nos concentrar na programação dinâmica determinística e retomaremos o caso estocástico apenas num dos estudos de casos (seção 4.1) e em trabalhos futuros.

Capítulo 3

Métodos numéricos propostos

Consideramos a programação dinâmica como a otimização de uma função custo restrita a um sistema dinâmico. A técnica para a solução deste problema complexo consiste em transformá-lo em uma seqüência de problemas mais simples. As soluções, obtidas de uma maneira incremental, determinam uma política (ou estratégia) ótima, de custo mínimo para o sistema.

Desta forma, as decisões são tomadas em estágios: para dar início ao procedimento de cálculo, é necessário primeiro resolver o sub-problema para um dos estágios, por exemplo, o estágio final, e usando o Princípio da Otimalidade de Bellman (princípio 2.2.1), continuar o processo com indução para trás, até atingir o estágio inicial. A solução em cada estágio é obtida através de um ordenamento baseado na soma do custo presente e do custo futuro esperado.

Como visto no capítulo anterior, os problemas de programação dinâmica com variáveis inteiras são resolvidos montando-se a árvore de possibilidades, listando, para cada estágio de tempo, os estados e os controles possíveis. Para se obter o mínimo da função custo do sistema dinâmico é só procurar o caminho mínimo nesta árvore.

3.1 Visão geral dos algoritmos propostos

A novidade deste trabalho é otimizar apenas em um dos estágios. Conseguimos isto, olhando para o sistema com uma ótica geométrica, ou seja, colocando a dinâmica do sistema na **pré-imagem de conjuntos**, em vez de colocá-la nos arcos de um grafo representando a transição de estados. A idéia é considerar apenas em um dos estágios (por exemplo, o estágio inicial) a pré-imagem de cada

estado pensado como pré-imagem de um conjunto por meio de uma função linear contínua.

Se as variáveis do problema forem números reais, ao resolvermos assim o problema de programação dinâmica, encontraremos o ponto ótimo (a menos da limitação do computador que só trabalha com um número finito de casas decimais).

Se as variáveis forem inteiras, propomos fazer uso da técnica conhecida como *relaxação linear*. Esta técnica consiste em aproximar o problema de variáveis discretas por um problema de variáveis contínuas (conhecido como problema relaxado) e, resolvendo o problema contínuo nas variáveis, conseguir uma cota inferior para o problema de variáveis discretas. Se quisermos o ótimo do problema com as variáveis inteiras, poderemos usar as formulações aqui propostas como um problema de programação inteira.

Assim, enxergamos o problema dinâmico como um problema linear ou não-linear estático com um número de variáveis bem menor que o original (pois trabalharemos apenas com as variáveis de estado em um dos estágios). Ou seja, conseguimos reduzir significativamente a complexidade computacional do problema.

Podemos, ainda, reduzir o valor da função custo do problema se aumentarmos a sua região factível. Definiremos, para isto, o que são famílias invariantes pelo sistema dinâmico. A chave para esta redução consiste em considerarmos um conjunto em torno da meta de programação pertencente a uma família invariante e investigar sua pré-imagem. Sendo o conjunto pertencente a uma família invariante, tal iteração será possível para um número arbitrário de instantes. Podemos, então, estudar a dinâmica do sistema através das equações destes conjuntos, iterando-os até um dos estágios (por exemplo, o estágio inicial) e otimizando apenas aí.

Uma meta de programação (que denotaremos por x^*) é o ponto onde se quer chegar com a otimização dinâmica, ou seja, queremos que ao final dos N estágios do problema, tenhamos $x[N] = x^*$. Portanto, esta meta é, em geral, um ponto de estabilidade do sistema dinâmico.

Vamos trabalhar com sistemas dinâmicos discretos, determinísticos, de parâmetros concentrados, invariantes com o tempo, sem a ação de controle e com a ação de controle.

No primeiro caso (sem o controle), apresentaremos três formulações: uma considerando a pré-imagem no estágio inicial da meta da programação dinâmica (por exemplo, um ponto fixo do sistema dinâmico), outra considerando a pré-imagem de uma bola em torno deste ponto e outra considerando a pré-imagem de um hipercubo em torno deste ponto.

Veremos que a complexidade computacional da formulação sem a ação de controle considerando a pré-imagem da meta é a complexidade de um problema de programação linear estático (supondo a função custo linear) com n variáveis e restrições, sendo que n é o número de estados em um dos estágios. Pela formulação com a aproximação por um hipercubo em torno da meta de programação, veremos que o custo computacional é o mesmo e que, com a folga dada, a função custo é, às vezes, consideravelmente menor. A formulação considerando uma bola em torno da meta pode ser resolvida como um problema de programação não-linear estático.

Como o sistema dinâmico é linear podemos estudar a ação de controle separadamente da ação do sistema (princípio 2.1.1). Assim, para considerarmos o caso com a ação de controle, basta fazermos a otimização da função custo restrita aos conjuntos encontrados na formulação sem a ação de controle transladadas pela união das pré-imagens dos controles possíveis.

No caso com a ação de controle, apresentaremos a formulação considerando a pré-imagem da meta de programação e a formulação considerando a pré-imagem de um hipercubo em torno desta meta.

Veremos que a complexidade computacional da formulação com a ação de controle (supondo a função custo linear) considerando a pré-imagem da meta é a complexidade de um problema de programação linear estático com $p+n$ variáveis e restrições, sendo que n é o número de estados em um dos estágios e p é o número de controles possíveis. Pela formulação com a aproximação por um hipercubo em torno da meta de programação, veremos que o custo computacional é o mesmo e que, com a folga dada, a função custo é, às vezes, consideravelmente menor.

Complexidade computacional

Como enxergaremos o problema de programação dinâmica como um problema de programação estática, vamos apresentar as referências para a análise da complexidade computacional e velocidade de convergência de algoritmos de programação estática linear e não linear.

Como já exposto, a complexidade computacional de um algoritmo é uma função matemática que permite compará-lo com os demais. Em geral, procura-se caracterizar a eficiência de um algoritmo estudando o tempo máximo de computação necessário para a resolução do problema em função de seu tamanho, dentre outras formas de compará-lo aos demais. A bibliografia sobre o assunto é extensa, citamos abaixo apenas algumas referências.

[Takahashi (2005] apresenta um estudo detalhado sobre algoritmos de programação não-linear estática. Ele define convergência de algoritmos e comenta a velocidade de convergência de cada um deles.

Para os estudos de casos com programação não-linear, implementamos o método de *Quasi-Newton* com o tratamento de restrições pelo *Método de Barreira* com a abordagem *BFGS restrito*. Este é um método de direção de busca, muito usado em otimização não linear quando a função objetivo e a região factível são convexas e conhece-se um ponto no interior da região factível. [Takahashi (2005)] mostra que esta abordagem tem convergência quadrática.

Para a resolução dos problemas de programação linear, duas classes de métodos são as mais usadas: *Simplex* e *Pontos Interiores*. Os métodos *Simplex* procuram pelo ótimo da função através dos vértices do politopo correspondente à região factível do problema. Os métodos de *Pontos Interiores* caminham buscando o ótimo pelo interior da região factível na direção contrária à de crescimento da função.

[Gonzaga (1990)] apresenta algoritmos de *Pontos Interiores* e mostra que eles têm custo computacional, no pior caso, polinomial em função do tamanho do problema. Ele define o tamanho de um problema de programação linear por $L = m + n + 1$, sendo que m é o número total de bits utilizados pela entrada de dados e n é a dimensão do espaço, ou seja, o número de restrições. [Goldbarg & Luna (2000)] e [Bazaraa (1999)] apresentam algoritmos do *Simplex* com diversos exemplos e aplicações. [Bazaraa (1999)] mostra que, no pior caso, estes algoritmos têm custo computacional exponencial, da ordem de 2^n, com n igual ao número de restrições do problema. Mas, na maioria dos casos práticos, o *Simplex* tem comportamento polinomial.

Neste trabalho, resolveremos os problemas de programação linear pela função *linprog* do Matlab[1], que usa métodos de Pontos Interiores.

[1]Matlab é marca registrada de The MathWorks, Inc.

3.2 Famílias invariantes

Neste trabalho, damos o enfoque em sistemas dinâmicos discretos, lineares, determinísticos, de parâmetros concentrados, invariantes com o tempo. O nosso objetivo é alterar o problema original, substituindo a meta da programação dinâmica por um conjunto pertencente a uma família invariante. Vamos, assim, aumentar a região factível do problema e, possivelmente, reduzir de modo considerável a função custo.

No caso do sistema dinâmico sem a ação de controle, cada iteração nada mais é que uma transformação linear: multiplicar o ponto (o vetor de estados) por uma matriz constante. Se esta matriz for não-singular, cada iteração será, simplesmente, uma mudança de coordenadas. Considerando a ação de controle, cada iteração é uma transformação afim, ou seja, uma transformação linear composta com uma translação.

Mas, como podemos estudar separadamente a transformação linear da translação, podemos trabalhar com o conceito de famílias invariantes (apenas) por uma transformação linear.

Definição 3.2.1 *Uma família $\mathcal{F}$ de conjuntos é invariante por uma transformação linear A, ou apenas, é uma família invariante, quando a iteração por A de cada conjunto $V \in \mathcal{F}$ é um conjunto $W \in \mathcal{F}$.*

Seja o sistema dinâmico discreto linear sem a ação de controle como em (2.13):

$$x[k+1] = Ax[k]$$

sendo que A é uma matriz constante de dimensão n.

Considere $x \in \mathbb{R}^n$ e $\overline{x} \in \mathbb{R}^n$ com $\overline{x} = Ax$. Considere também uma função real $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$.

Considere a família de conjuntos $\{\Phi_\alpha\}_{\alpha \in \mathbb{R}^m}$, na qual cada conjunto é parametrizado por $\alpha \in \mathbb{R}^m$ da forma:

$$\Phi_\alpha = \{x \in \mathbb{R}^n; f(x, \alpha) \leq 0\}. \tag{3.1}$$

Aplicando a transformação linear A num conjunto Φ_α da família, obtemos o conjunto:

$$A(\Phi_\alpha) = \Psi_{(\alpha,A)} = \{\overline{x} \in \mathbb{R}^n; \overline{x} = \ Ax, x \in \Phi_\alpha\}.$$

A pergunta é: quando o conjunto $\Psi_{(\alpha,A)}$ pertencerá à família $\{\Phi_\alpha\}_{\alpha\in\mathbb{R}^m}$?

Pela definição (3.2.1), o conjunto acima pertencerá à família, quando, para cada $\alpha \in \mathbb{R}^m$, existir um $\overline{\alpha} \in \mathbb{R}^m$, tal que, se $f(x,\alpha) \leq 0$, então $f(\overline{x},\overline{\alpha}) \leq 0$, para todo $\overline{x} \in \Psi_{(\alpha,A)}$.

Nota 3.2.1 *A família $\{\Phi_\alpha\}_{\alpha\in\mathbb{R}^m}$ é invariante (pela transformação A) se, e somente se, para cada α, existe um $\overline{\alpha}$, tal que $\Psi_{(\alpha,A)} = \Phi_{\overline{\alpha}}$.*

Considere x^* uma meta de programação, ou seja, queremos que para todo valor inicial $x[0]$, tenhamos $x[N] = A^N x[0] = x^*$, com N representando o estágio final.

Vamos dar um exemplo de uma família de conjuntos que não é invariante e dois exemplos de famílias invariantes. Antes disso, vamos definir e caracterizar o conjunto das matrizes quadradas simétricas definidas positivas e semi-definidas positivas.

Definição 3.2.2 *Chamamos $\mathcal{D}$ o conjunto de todas as matrizes quadradas simétricas definidas positivas e $\mathcal{E}$, o conjunto de todas as matrizes quadradas simétricas semi-definidas positivas.*

Uma matriz quadrada M é simétrica quando $M = M^t$. Sobre as matrizes definidas e semi-definidas positivas, apresentamos o teorema a seguir.

Teorema 3.2.1 *São equivalentes as seguintes afirmativas:*

1. *A matriz M, de dimensão n, pertence a $\mathcal{D}$ ($\in \mathcal{E}$).*
2. *Para todo vetor não nulo $x \in \mathbb{R}^n$, vale que $x^tMx > 0$ ($x^tMx \geq 0$).*
3. *Os autovalores de M são positivos (não-negativos).*
4. *Existe uma matriz não-singular (singular) N tal que $M = N^tN$.*

Demonstração: Ver [Chen (1999)]. ■

Considere a família de todas as bolas em $\mathbb{R}^n$ de raio ϵ: $\{\Phi_p\}_{p\in\mathbb{R}^n}$. Cada conjunto desta família pode ser parametrizado[2] por:

$$\Phi_p = \{x \in \mathbb{R}^n; ||x - p||_2 \leq \epsilon\},$$

[2]Veja a definição de bola e a justificativa desta parametrização no apêndice.

sendo que $||.||_2$ é norma euclidiana (a função f da equação 3.1 é a função "distância menor que ϵ" do vetor $x - p$).

Esta família de bolas em $\mathbb{R}^n$ de raio ϵ não é invariante. Como mostraremos a seguir, uma bola pertencente a esta família quando iterada pelo sistema dinâmico acima, não é necessariamente uma bola, mas um elipsóide (pertencente à família de elipsóides em $\mathbb{R}^n$ com raio ϵ)[3].

Exemplo 3.2.2 *A pré-imagem no estágio inicial k=0 por A de uma bola com centro em x^* no estágio k=N, resultado de $||x[N] - x^*||_2 \leq \epsilon$, é um elipsóide.*

Demonstração: Esta bola com centro em x^* e raio ϵ no estágio N pode ser escrita como o conjunto:

$$\Phi_{x^*} = \{x[N] \in \mathbb{R}^n; ||x[N] - x^*||_2 \leq \epsilon\}$$

Seja I a matriz identidade. A equação acima pode ser escrita como:

$$(x[N] - x^*)^t I(x[N] - x^*) \leq \epsilon^2.$$

Como $x[N] = A^N x[0]$ e chamando $x^* = A^N \bar{x}$, para algum $\bar{x}$ em $k = 0$, podemos reescrever a equação acima como:

$$(x[0] - \bar{x})^t (A^N)^t A^N (x[0] - \bar{x}) \leq \epsilon^2.$$

É claro que esta equação não pode ser escrita na forma Φ_p para nenhum $p \in \mathbb{R}^n$, como queríamos mostrar.
Pelo teorema (3.2.1), a matriz $E = (A^N)^t A^N$ é simétrica semi-definida positiva. Como veremos abaixo, esta é a equação de um elipsóide em $\mathbb{R}^n$ no estágio $k = 0$ com centro em $\bar{x}$ e matriz geradora E. □

Vamos apresentar, agora, dois exemplos simples de famílias invariantes: a de elipsóides e a de politopos.

Considere a família $\{\Phi_{p,E}\}_{p \in \mathbb{R}^n,\ E \in \mathcal{E}}$ de todos os elipsóides em $\mathbb{R}^n$, assim parametrizados[4]:

$$\Phi_{p,E} = \{x \in \mathbb{R}^n; (x - p)^t E(x - p) \leq \epsilon^2\}.$$

Vamos mostrar que esta família é invariante.

[3]O conjunto das bolas é um subconjunto do conjunto dos elipsóides. O conjunto dos elipsóides é invariante (como mostraremos), mas o subconjunto das bolas não é (só seria se a transformação linear tivesse todos os autovalores iguais a 1).

[4]Veja a definição de elipsóide e a justificativa desta parametrização no apêndice.

Teorema 3.2.3 *A pré-imagem no estágio inicial k=0 por A de um elipsóide com centro em x^* e raio ϵ no estágio k=N, é um elipsóide.*

DEMONSTRAÇÃO: Seja E a matriz geradora do elipsóide com centro em x^* e raio ϵ no estágio $k = N$. Este elipsóide pode ser escrito como o conjunto:

$$\Phi_{x^*,E} = \{x[N] \in \mathbb{R}^n; (x[N] - x^*)^t E(x[N] - x^*) \leq \epsilon^2\}.$$

Como $x[N] = A^N x[0]$ e chamando $x^* = A^N \bar{x}$, podemos reescrever a equação do conjunto acima como:

$$(x[0] - \bar{x})^t (A^N)^t E A^N (x[0] - \bar{x}) \leq \epsilon^2.$$

Pelo teorema (3.2.1), existe uma matriz não-singular N, tal que $E = N^t N$. Portanto, a matriz $M = (A^N)^t E A^N$ é simétrica e semi-definida positiva, pois $M = (A^N)^t N^t N A^N$, ou seja, $M = (NA^N)^t (NA^N)$.
Podemos reescrever a equação acima como:

$$\Phi_{\bar{x},M} = \{x[0] \in \mathbb{R}^n; (x[0] - \bar{x})^t M(x[0] - \bar{x}) \leq \epsilon^2\},$$

Logo, esta é a equação de um elipsóide no estágio $k = 0$ com centro em $\bar{x}$ e matriz geradora M, como queríamos mostrar. ■

Considere, agora, a família $\{\Phi_{p,D}\}_{p\in\mathbb{R}^n,\ D\in\mathcal{D}}$ de todos os elipsóides em $\mathbb{R}^n$ não-degenerados, assim parametrizados[5]:

$$\Phi_{p,D} = \{x \in \mathbb{R}^n; (x - p)^t D(x - p) \leq \epsilon^2\}.$$

Observamos que quando a matriz A do sistema dinâmico for não-singular, esta família também será invariante.

Corolário 3.2.4 *A pré-imagem no estágio inicial k=0 por uma matriz não-singular A de uma elipsóide não-degenerado com centro em x^* e raio ϵ no estágio k=N, é um elipsóide não-degenerado.*

Observamos também que a família das quádricas (em geral) é invariante. Mais ainda, quaisquer famílias de superfícies polinomiais de grau até n são invariantes. Do mesmo modo, podemos generalizar para a família das quádricas e das superfícies de grau n não-degeneradas quando as transformações lineares forem não-singulares, ou seja, cujas matrizes tiverem determinantes diferentes de zero.

Vamos mostrar agora que a família de todos os politopos também é invariante.

[5]Veja a definição de elipsóide não-degenerados e a justificativa desta parametrização no apêndice.

Definição 3.2.3 *Um politopo em $\mathbb{R}^n$ é a interseção de semi-espaços limitados por hiperplanos.*

A um vetor v do $\mathbb{R}^n$, pode-se associar a norma do máximo, ou norma infinita, assim definida:

$$||v||_\infty = \max_{i=1,\ldots,n} |v(i)|.$$

Considere o hipercubo P $= \{x[N]; ||x[N] - x^*||_\infty \leq \epsilon\}$, que chamaremos de hipercubo de raio ϵ com centro em x^* no estágio $k = N$. É claro que todo hipercubo é um politopo. Vamos mostrar que este hipercubo quando iterado pelo sistema dinâmico continua um politopo.

Teorema 3.2.5 *A pré-imagem no estágio inicial k=0 por A de um hipercubo P com centro em x^* no estágio k=N, resultado de $||x[N]-x^*||_\infty \leq \epsilon$, é um politopo.*

Demonstração: Como nos outros casos, consideremos $x^* = A^N\bar{x}$. O hipercubo P é a interseção dos seguintes semi-espaços limitados por hiperplanos[6]:

$$\begin{cases} e_i^t(x[N] - (x^* + e_i\epsilon)) \leq 0, & \text{para cada } i \\ -e_i^t(x[N] - (x^* - e_i\epsilon)) \geq 0, & \text{para cada } i. \end{cases}$$

Vamos, portanto, nos limitar em demonstrar que a pré-imagem do hiperplano

$$e_i^t(x[N] - (x^* + e_i\epsilon)) = 0$$

é um hiperplano e que $\bar{x}$ pertence à interseção de todos estes hiperplanos no estágio $k = 0$.
A matriz A pertence ao autoespaço $\mathbb{R}^{n\times n}$. Vamos fazer a seguinte decomposição em soma direta: $\mathbb{R}^n = (U = \mathbb{R}^m) \oplus (W = \mathbb{R}^{n-m})$ sendo que a dimensão de U é a dimensão da maior submatriz invertível de A^N. Chame esta submatriz de V. Então, podemos trabalhar num sistema de coordenadas no qual a matriz A^N se escreva como:

$$A^N = \begin{bmatrix} V & 0 \\ 0 & 0 \end{bmatrix}.$$

Consideremos primeiro apenas o autoespaço associado a U. Desta forma, x_U quer dizer a projeção de x nas coordenadas de U.
Como $x_U[N] = V^N x_U[0]$ e $x_U^* = V^N\bar{x}_U$, podemos reescrever a equação acima como:

$$e_i^t V^N x_U[0] - (V^N\bar{x}_U + e_i\epsilon)) = 0 \Rightarrow$$
$$V^N V^{-N} e_i^t V^N x_U[0] - (V^N\bar{x}_U + e_i\epsilon)) = 0 \Rightarrow$$

[6]Veja sobre esta parametrização do politopo e do hipercubo no apêndice.

$$V^N e_i^t (x_U[0] - (\bar{x}_U + V^{-N} e_i \epsilon)) = 0,$$

ou, de outra forma:

$$e_i^t (V^N x_U[0] - (V^N \bar{x}_U + e_i \epsilon)) = 0 \ \Rightarrow$$

$$e_i^t (V^N x_U[0] - (x_U^* + e_i \epsilon)) = 0.$$

Esta é a equação de um hiperplano em U no estágio $k = 0$ que contém $\bar{x}_U$ tal que $V^N \bar{x}_U = x_U^*$.
Como podemos fazer isto para cada hiperplano e, conseqüentemente, para cada semiespaço, a projeção da pré-imagem do hipercubo nas coordenadas de U é um politopo (interseção dos semiespaços limitados por hiperplanos) M que contém $\bar{x}_U$.
Vamos mostrar agora que a imagem de qualquer ponto em $k = 0$ cuja projeção em U esteja neste politopo estará no hipercubo em $k = N$. Ou seja, que as coordenadas associadas ao autoespaço W são irrelevantes. De fato, tome $x_U \in M \subset U$. Considere um $x_W \in W$ qualquer. Então $x = [x_U \ x_W]^t \in \mathbb{R}^n$. Assim,

$$A^N x \ = \ A^N [x_U \ 0]^t + A^N [0 \ x_W]^t \ = \ V^N x_U + 0 \ = \ V^N x_U \in P$$

por hipótese. Isto conclui a demonstração. ■

Para as próximas seções, será importante considerar que a pré-imagem em $k = 0$ do hipercubo P é o politopo formado pela interseção dos semiespaços limitados pelos hiperplanos:

$$\begin{cases} e_i^t (A^N x[0] - (x^* + e_i \epsilon)) \leq 0, & \text{para cada } i \\ -e_i^t (A^N x[0] - (x^* - e_i \epsilon)) \geq 0, & \text{para cada } i. \end{cases}$$

como mostramos acima.

Apesar desta demonstração específica para o hipercubo, é próprio da transformação linear levar hiperplano em hiperplano, portanto, levar semi-espaço em semi-espaço. Logo, a transformação linear leva interseção de semi-espaços em interseção de semi-espaços, ou seja, leva politopos em politopos. Esta família também é, portanto, invariante.

3.3 Otimização sem a ação de controle

Neste trabalho, queremos evitar a enumeração dos estados de todos os estágios, desenvolvendo uma metodologia que concentre todas as variáveis de estado em

apenas um dos estágios. Vamos apresentar esta nova metodologia considerando o caso da otimização restrita ao sistema dinâmico sem e com a ação de controle. Comecemos pelo mais simples: na ausência da ação de controle.

Assim sendo, considere o sistema dinâmico sem a ação de controle, como em (2.13):

$$x[k+1] = \mathrm{A}x[k],$$
$$k = 0, 1, \ldots, N-1.$$

Queremos trabalhar em apenas um dos estágios, por exemplo, o estágio inicial. Como na seção anterior, x^* é uma meta da programação dinâmica. Dado este estado no estágio final N, queremos encontrar o estado inicial que otimize uma função custo.

Fazer a iteração das restrições do problema e da função custo para o estágio inicial é substituir formalmente nas variáveis de cada estágio a relação do sistema dinâmico $x[k] = A^k x[0]$.

Seja g, a função que represente o custo total do sistema dinâmico como função do estado no estágio inicial e f a função que represente as demais restrições nas variáveis de estado, também em função do estágio inicial.

Propomos, então, a seguinte formulação geral para o problema de programação dinâmica discreto sem a ação de controle:

$$\min_{x[0]} g(x[0]) \tag{3.2}$$

$$\text{sujeito a: } \begin{cases} x[k+1] = Ax[k], \ k = 0, \ldots, N-1 \\ x[N] = x^* \\ f(x[0]) \leq 0. \end{cases}$$

O procedimento proposto para a solução deste problema é muito simples:

1. Permitimos uma relaxação na meta de programação dinâmica (no nosso caso, em x^*), que é substituída por um conjunto pertencente a uma família invariante ao seu redor.

2. Fazemos a iteração no tempo deste conjunto para apenas um dos estágios (no nosso caso, consideramos a relaxação no estágio final $k = N$ e trabalhamos no estágio inicial $k = 0$).

3. Fazemos a otimização (linear ou não-linear, isento da restrição dinâmica) apenas neste estágio.

Nas seções a seguir, vamos nos preocupar **apenas com a restrição da dinâmica** do sistema na formulação (3.2). Nos estudos de casos (próximo capítulo), consideraremos melhor as demais restrições do problema.

Estamos considerando as variáveis do problema como números reais. Respeitando, claro, a limitação da precisão do computador, a solução encontrada pela procedimento proposto neste caso é a solução ótima. Se as variáveis forem números inteiros, o procedimento proposto corresponde a fazer a relaxação linear nas variáveis e encontrar um limite inferior para a função custo.

Esta formulação proposta perde o caráter exponencial dos problemas de programação dinâmica, como era nossa intenção, pois evita a formação da árvore de possibilidades. A complexidade computacional vai depender do problema a ser estudado, mas, será em função do vetor de estados apenas no estágio inicial.

Se quisermos trabalhar o problema com variáveis inteiras como um problema de programação inteira, basta acrescentarmos na formulação geral (3.2) a restrição de integralidade para as variáveis. Lembramos apenas, que resolver um problema de programação inteira requer uma solução de complexidade exponencial em função de suas variáveis ([Goldbarg & Luna (2000)]).

Antes de apresentar a abordagem com a relaxação na meta, vamos mostrar a otimização restrita apenas à pré-imagem desta meta. Em seguida, consideraremos um conjunto pertencente a uma família invariante elipsoidal em torno da meta. Por fim, consideraremos um conjunto pertencente à família politópica em torno deste ponto.

Apresentaremos estudos de casos com testes comparando as três formulações no próximo capítulo.

3.3.1 Otimização restrita à pré-imagem da meta

Antes de relaxar a meta de programação por um conjunto pertencente a uma família invariante, vamos otimizar a função objetivo restrita à pré-imagem desta meta x^*. Como o sistema é linear, a pré-imagem de um ponto é um ponto (se a matriz for invertível) ou uma variedade linear (caso contrário). No primeiro caso, teríamos uma única solução como resultado da otimização, mas no segundo

teríamos infinitas soluções, fazendo sentido perguntar qual a solução de custo mínimo.

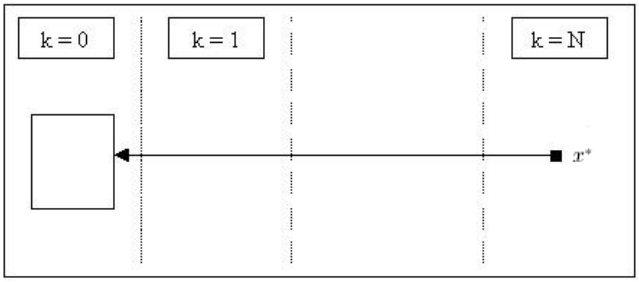

Figura 3.1: Otimização restrita à pré-imagem da meta.

A fim de encontrar um problema de programação linear estático, vamos supor que a função custo seja linear $g(x[0]) = c^t x[0]$, sendo que c é um vetor com a mesma dimensão de $x[0]$. Propomos, então, a seguinte formulação para este caso:

$$\min_{x[0]} c^t x[0] \tag{3.3}$$

$$\text{sujeito a: } A^N x[0] = x^*$$

Portanto, usando métodos de Pontos Interiores para a resolução deste problema de programação linear, a complexidade computacional é polinomial na ordem da dimensão do vetor de estados $x[0]$ no estágio inicial. Esta formulação é boa, como veremos nos estudos de casos, mas traz consigo as possíveis dificuldades de um problema com restrição de igualdade. Nestes casos, a região factível fica muito pequena, podendo dificultar muito o encontro do ponto ótimo, atrasando a convergência do método.

3.3.2 Otimização restrita à pré-imagem de um elipsóide

[Takahashi et. al (1997)] propuseram esta técnica. Neste trabalho, x^* era um ponto de estabilidade do sistema dinâmico, ou seja, um autovetor associado ao autovalor unitário da matriz do sistema dinâmico. A idéia era dar uma perturbação em x^* e otimizar apenas a variável inicial $x[0]$ de modo a alcançar a

solução final a menos de um erro.

Para isto, bastaríamos tomar uma bola de raio ϵ em torno de x^* no estágio $k = N$. De acordo com o exemplo 3.2.2, fazendo o retorno desta bola para o estágio inicial $k = 0$, teremos um elipsóide com raio ϵ com centro em algum $\bar{x}$ tal que $x^* = A^N \bar{x}$.

Quando x^* for um ponto de estabilidade, então $x^* = A^k x^*$ para todo k. Neste caso, o elipsóide estaria centrado no próprio x^*.

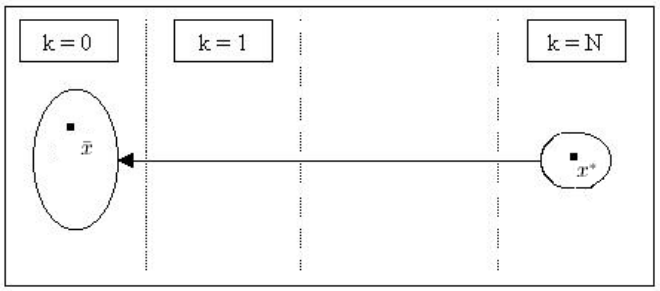

Figura 3.2: Otimização restrita à pré-imagem de um elipsóide.

O exemplo 3.2.2 apresenta a equação do elipsóide no estágio inicial. Assim, basta otimizar a função objetivo restrita a este elipsóide.

Este problema pode ser formulado como:

$$\min_{x[0]} g(x[0]) \tag{3.4}$$

$$\text{sujeito a: } (x[0] - \bar{x})^t (A^N)^t A^N (x[0] - \bar{x}) \leq \epsilon^2$$

Teremos, no estágio inicial, uma região factível maior que na formulação anterior (pré-imagem de uma bola em vez de pré-imagem de um ponto apenas). Possivelmente, ao otimizarmos seguindo esta formulação, conseguiremos uma função custo ainda menor que no caso sem a relaxação.

Este é um problema de programação não-linear estática, e pode ser resolvido por algoritmos simples como os *Quasi-Newton* com o tratamento de restrições pelo *Método de Barreira* com a abordagem *BFGS restrito*, que têm convergência quadrática.

3.3.3 Otimização restrita à pré-imagem de um politopo

Através desta abordagem, se considerarmos como função custo uma função linear, enxergamos o problema de programação dinâmica como um problema de programação linear estático - como no primeiro caso (formulação 3.3) - e ainda conseguiremos uma redução na função objetivo - como no segundo caso (formulação 3.4).

A idéia é considerar o hipercubo P em torno de x^* no estágio final $k = N$: $\mathrm{P} = \{x[N]; ||x[N] - x^*||_\infty \leq \epsilon\}$.

De acordo com o teorema 3.2.5, fazendo o retorno de P para $k = 0$, continuaríamos com um politopo. A idéia continua sendo a mesma da formulação (3.4), só que agora otimizamos a função restrita a um politopo.

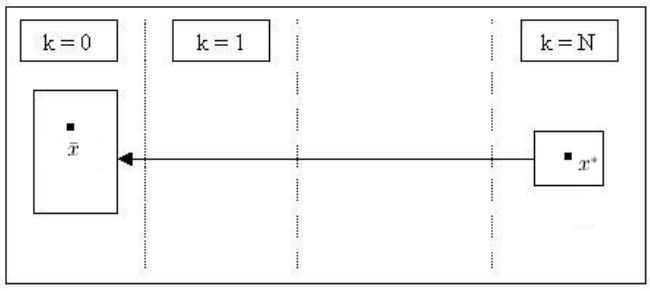

Figura 3.3: Otimização restrita à pré-imagem de um politopo.

Como mostramos no teorema, este politopo em $k = 0$ é formado pela interseção dos seguintes semi-espaços cujas fronteiras são hiperplanos:

$$\begin{cases} e_i^t(A^N x[0] - (x^* + e_i\epsilon)) \leq 0, & \text{para cada } i \\ -e_i^t(A^N x[0] - (x^* - e_i\epsilon)) \geq 0, & \text{para cada } i. \end{cases}$$

Vamos considerar a função custo linear: $g(x[0]) = c^t x[0]$. Podemos, então, formular este problema como:

$$\min_{x[0]} c^t x[0] \tag{3.5}$$

$$\text{sujeito a: } \begin{cases} e_i^t(A^N x[0] - (x^* + e_i\epsilon)) \leq 0, & \text{para cada } i \\ -e_i^t(A^N x[0] - (x^* - e_i\epsilon)) \geq 0, & \text{para cada } i. \end{cases}$$

Como as todas restrições, bem como a função objetivo são lineares, podemos utilizar métodos de programação linear para resolver o problema. Implementado por métodos de Pontos Interiores, esta formulação, como em (3.3), tem complexidade computacional polinomial da ordem de n, a dimensão de vetor de estados no estágio inicial, sem um possível inconveniente de uma restrição de igualdade. E assim como em (3.4), como a região factível é maior, poderemos conseguir uma função custo consideravelmente menor.

3.4 Otimização com a ação de controle

Vamos, agora, acrescentar o caso geral: otimizar a função custo restrita a um sistema dinâmico com ação de controle. Isto quer dizer que, em cada estágio, podemos interferir no sistema dinâmico.

Suponha o sistema com p entradas em cada um dos N estágios. Assim, além das variáveis de estado $x[k]$, teremos as variáveis de entrada ou controle $u_i[k]$ para cada entrada i e estágio de tempo k. Vamos usar a notação $u[k]$ para designar o vetor p-dimensional correspondente às variáveis de entrada ou controle no estágio k: $[u_1[k], ..., u_p[k]]^t$.

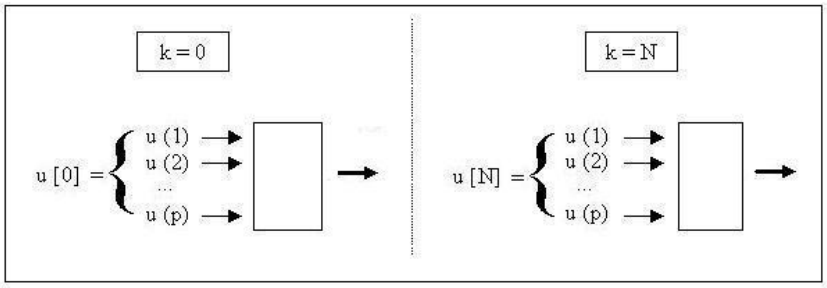

Figura 3.4: Esquema de um sistema dinâmico com p entradas em cada um dos N estágios.

Considere o sistema dinâmico com a ação de controle, como em (2.11):

$$x[k+1] = \mathrm{A}x[k] + \mathrm{B}u[k],$$

$$k = 0, 1, \ldots, N-1.$$

Também aqui, queremos trabalhar com as variáveis de estado em apenas um dos estágios. Dado o vetor x^* (uma meta da programação dinâmica) no estágio N, queremos encontrar o estado no estágio inicial e a seqüência de controles que minimizem a função custo.

A função g representa o custo como função do estágio inicial e cada função h_k representa o custo da variável de controle $u[k]$ no estágio k. Portanto, o custo total a ser minimizado é:

$$J = g(x[0]) + \sum_{k=0}^{N-1} h_k(u[k]).$$

As funções f e $\bar{f}$ são as restrições nas variáveis de estado no estágio inicial e de controle, respectivamente. Como na ausência de controle, fazer a iteração das variáveis de estado destas restrições e da função custo para um mesmo estágio é reproduzir formalmente nas variáveis em cada estágio a restrição do sistema dinâmico.

A proposição abaixo mostra como escrever o estado em qualquer estágio K em função do estado inicial $x[0]$ e da seqüência de controles anteriores, entre o estágio inicial e o estágio $K-1$.

Proposição 3.4.1 *Considere o sistema dinâmico com a ação de controle como em (2.11). Podemos escrever x[K], em cada estágio K, em função apenas do estado inicial x[0] e da seqüência dos controles anteriores, desta forma:*

$$x[K] = A^K x[0] + Bu[K-1] + \sum_{k=1}^{K-1} A^k Bu[N-(k+1)].$$

DEMONSTRAÇÃO: Vamos mostrar por indução em K.
Quando $K = 1$, por (2.11), temos que:

$$x[1] = Ax[0] + Bu[0].$$

Suponha, agora que a tese seja válida para até o estágio $n = K$.

$$x[n] = A^K x[0] + Bu[n-1] + \sum_{k=1}^{n-1} A^k Bu[N-(k+1)].$$

Esta é a hipótese de indução. Vamos mostrar que, então, a tese também vale para o estágio $n+1$. Também por (2.11), temos que:

$$x[n+1] = Ax[n] + Bu[n].$$

Substituindo a hipótese de indução na equação acima, temos que:

$$x[n+1] = A(A^n x[0] + Bu[n-1] + \sum_{k=1}^{n-1} A^k Bu[N-(k+1)]) + Bu[n].$$

Daí, segue que:

$$x[n+1] = A^{n+1}x[0] + ABu[n-1] + \sum_{k=2}^{n} A^k Bu[N-(k+1)]) + Bu[n].$$

Ou seja:

$$x[n+1] = A^{n+1}x[0] + Bu[n] + \sum_{k=1}^{n} A^k Bu[N-(k+1)]),$$

como queríamos mostrar. ■

Propomos a seguinte formulação para o problema de programação dinâmica, discreto, com a ação de controle:

$$\min_{x[0],u[0],\ldots,u[N-1]} g(x[0]) + \sum_{k=0}^{N-1} h_k(u[k]) \tag{3.6}$$

$$\text{sujeito a: } \begin{cases} x[k+1] = Ax[k] + Bu[k], \quad k = 0, \ldots, N-1 \\ x[N] = x^* \\ f(x[0]) \leq 0 \\ \bar{f}_{i,k}(u(i)[k]) \leq 0, \ i = 1, \ldots p, \ k = 0, \ldots, N. \end{cases}$$

A metodologia aqui proposta é semelhante à da otimização sem a ação de controle:

1. Primeiro, permitimos fazer a relaxação na meta de programação, que é substituída por um conjunto pertencente a uma família invariante ao seu redor.

2. Depois, fazemos o retorno deste conjunto pertencente à família invariante e de cada ação de controle possível para o estágio inicial, usando a mesma idéia de pré-imagem por funções contínuas.

3. Como o sistema é linear, a região factível será a soma da pré-imagem do conjunto pertencente à família invariante com as pré-imagens das ações de controle, interseção com as pré-imagens das demais restrições. Usando o Princípio da Superposição (2.1.1), a soma daqueles conjuntos é a soma das equações que o representam.

4. Por fim, basta fazermos a otimização restrita a esta região factível para encontrarmos a solução do problema.

Como no caso sem a ação de controle, vamos nos preocupar **apenas com a restrição dinâmica** do sistema na formulação (3.6). Nos estudos de casos, consideraremos melhor as demais restrições do problema.

Mas, desde já observamos que, com esta formulação, poderemos facilmente otimizar as variáveis de controle limitadas por um valor máximo e/ou um valor mínimo:

$$\min \leq u(i)[k] \leq \max.$$

Também aqui, estamos considerando as variáveis do problema como números reais. Respeitando, claro, a limitação da precisão do computador, a solução encontrada pela procedimento proposto também será a ótima. Se as variáveis forem números inteiros, o procedimento proposto corresponde a fazer a relaxação linear e encontrar um limite inferior para a função custo ótima do poblema com variáveis inteiras.

Como no caso da ausência da ação de controle, esta formulação perde o caráter exponencial dos problemas de programação dinâmica, como era nossa intenção, pois evita a montagem da árvore de possibilidades. A complexidade computacional vai depender do problema a ser estudado, mas, será em função do vetor de estados apenas no estágio inicial e das variáveis de controle.

Se o problema tem variáveis inteiras, também podemos resolvê-lo usando as formulações aqui propostas, tratando as variáveis enquanto tais (sem a relaxação linear). Para isto, basta acrescentarmos na formulação (3.6) a restrição de variáveis inteiras e, como no caso sem a ação de controle, resolver este problema, só que, a um custo exponencial.

Antes de apresentar a abordagem com a relaxação na meta, vamos mostrar a otimização restrita apenas à pré-imagem desta meta. Em seguida, consideraremos um conjunto pertencente a uma família invariante politópica em torno da meta.

No próximo capítulo, apresentaremos estudos de casos com testes comparando as duas formulações.

3.4.1 Otimização restrita à pré-imagem da meta

Considere, como nos casos anteriores, x^* uma meta da programação dinâmica no estágio final. Podemos pensar em otimizar a função objetivo restrita à pré-imagem deste ponto x^*, sujeita agora às ações de controle ao longo de cada estágio.

Como vimos, a pré-imagem da meta no estágio inicial sem a ação de controle é um ponto (matriz não-singular) ou uma variedade linear (caso contrário). Como o sistema é linear, para aplicar a ação de controle precisamos apenas de transladar este conjunto de acordo com as pré-imagens no estágio inicial do controle de cada estágio.

Pela proposição (3.4.1), é possível escrever $x[N]$, o estado no estágio N, em função apenas do estado inicial $x[0]$ e da seqüência dos controles anteriores:

$$x[N] = A^N x[0] + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)]. \tag{3.7}$$

Como queremos no estágio final $x[N] = x^*$, a restrição do sistema dinâmico pode ser escrita como:

$$A^N x[0] + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] = x^*. \tag{3.8}$$

Verificamos aqui o Princípio da Superposição (princípio 2.1.1). De fato, a restrição do sistema dinâmico (3.8) é a soma da restrição do sistema sem a ação de controle com a seqüência de controles:

- restrição do sistema sem a ação de controle:

$$A^N x[0] = x^*.$$

- seqüência de ações de controle:

$$Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)].$$

Vamos supor que a função custo seja linear, ou seja, que este seja um problema de programação linear estático. Assim, $g(x[0]) = c^t x[0]$ e, para cada k, $h_k(u[k]) = d_k^t u[k]$, sendo que c é um vetor com a dimensão de $x[0]$ e cada d_k é um vetor com a dimensão de $u[k]$.

Propomos, então, a seguinte formulação para este problema:

$$\min_{x[0],u[0],\ldots,u[N-1]} c^t x[0] + \sum_{k=0}^{N-1} d_k^t u[k] \tag{3.9}$$

$$\text{sujeito a: } A^N x[0] + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] = x^*$$

Poderemos resolver este problema por algoritmos de Pontos Interiores, a um custo computacional polinomial da ordem do número de restrições, ou seja, do número de estados no estágio inicial. Só que, como no caso sem a ação de controle, esta formulação traz consigo as possíveis dificuldades de um problema com restrição de igualdade. Nestes casos, a região factível fica muito pequena, podendo dificultar muito o encontro do ponto ótimo, atrasando a convergência do método.

3.4.2 Otimização restrita à pré-imagem de um politopo

A idéia aqui é a mesma da formulação sem a ação de controle: considerar o mesmo hipercubo P em torno de x^* no estágio final (hipercubo com centro em x^* no estágio $k = N$). De acordo com o teorema (3.2.5), fazendo o retorno de P para $k = 0$, continuaríamos com um politopo.

Podemos usar o Princípio da Superposição (2.1.1) para encontrar a região factível deste problema. Este princípio afirma que podemos estudar separadamente os estados e o controle. A região factível, então, será a soma da pré-imagem do politopo P com as pré-imagens das ações de controle. Ou seja, será a soma das restrições dos estados com a seqüência de controles:

- restrições da pré-imagem do lado da coordenada i do hipercubo sem a ação de controle (interseção de dois semi-espaços):

$$\begin{cases} e_i^t(A^N x[0] - (x^* + e_i\epsilon)) \leq 0 \\ -e_i^t(A^N x[0] - (x^* - e_i\epsilon)) \geq 0 \end{cases}$$

- seqüência de ações de controle:

$$Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)]$$

- restrições da pré-imagem do lado da coordenada i do politopo sujeita ao controle:

$$\begin{cases} e_i^t(A^N x[0] - (x^* + e_i\epsilon)) + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] \leq 0 \\ -e_i^t(A^N x[0] - (x^* - e_i\epsilon)) + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] \geq 0. \end{cases}$$

Também a fim de trabalhar num problema de programação linear, vamos considerar a função custo linear. Então, $g(x[0]) = c^t x[0]$ e, para cada k, $h_k(u[k]) = d_k^t u[k]$.

Podemos, então, formular este problema como:

$$\min_{x[0],u[0],\ldots,u[N-1]} c^t x[0] + \sum_{k=0}^{N-1} d_k^t u[k] \tag{3.10}$$

$$\text{sujeito a: } \begin{cases} e_i^t(A^N x[0] - (x^* + e_i\epsilon)) + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] \leq 0, \\ \text{para cada } i \\ -e_i^t(A^N x[0] - (x^* - e_i\epsilon)) + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] \geq 0, \\ \text{para cada } i. \end{cases}$$

Poderemos, também aqui, resolver este problema através de métodos de Pontos Interiores com a complexidade computacional da ordem da formulação 3.9, sem o possível inconveniente da restrição de igualdade.

Além disso, como a região factível é maior, esta formulação pode conseguir uma função custo consideravelmente menor, como veremos nos estudos de casos a seguir.

Capítulo 4

Estudos de casos

Vários problemas em campos como engenharia, economia e logística podem ser resolvidos por meio da programação dinâmica. Em geral, problemas de otimização cujas decisões devem ser tomadas em estágios e cuja decisão tomada no estágio anterior influencia diretamente na decisão a ser tomada no estágio a seguir.

Com o objetivo de exemplificar a eficácia computacional das formulações detalhadas no capítulo anterior, vamos apresentar aqui três estudos de casos.

No primeiro, apresentaremos um exemplo cujas variáveis do problema são números reais. Como já discutido, a nossa formulação encontra o resultado ótimo para este caso. Escolhemos um problema clássico: simples, mas muito útil: um pêndulo invertido numa carroça. Vamos minimizar a força que a carroça faz no pêndulo, sujeito a uma ação do vento.

Depois, apresentaremos nossa metodologia para um problema clássico de otimização combinatória: o problema da mochila. Vamos mostrar, como em [Mascó (2001)], a formulação deste problema como um problema de programação inteira e como um problema de programação dinâmica. Vamos apresentar o resultado obtido ao resolvermos pelo algoritmo da programação dinâmica (teorema 2.2.1) e compará-lo com o resultado obtido pela relaxação linear do problema de programação inteira e com os obtidos pelas nossas formulações (com a relaxação nas variáveis e com a relaxação na meta).

Por fim, discutiremos a evolução de um rebanho de gado no tempo. [Takahashi et al. (1997)] estudaram o problema e apresentaram a formulação (3.4), descrita na seção 3.3.2 do capítulo anterior. Além desta, vamos apresentar as demais formulações aqui propostas (com e sem a ação de controle) para este pro-

blema e, ao final, vamos comparar e comentar os resultados obtidos. Aqui, a relaxação linear do problema e a relaxação na meta apresentam ótimos resultados. Outro fato notável é que o resultado da função objetivo do sistema com a ação de controle é consideravelmente menor do que a do sistema sem a ação de controle, como veremos.

Não nos preocuparemos em apresentar testes em excesso, apenas os necessários para verificarmos os conceitos e compararmos as formulações apresentadas.

Para resolvermos os problemas de programação linear, usamos a função *linprog* do Matlab, que usa métodos de Pontos Interiores. Para resolvermos os problemas de programação não-linear usamos a função *Quasi-Newton com barreira via BFGS restrito* implementada em Matlab. O tempo (em *segundos*) foi medido pelas funções *tic/toc* do Matlab. Os programas foram rodados numa máquina Athon XP1600+ com 1,4 GHz.

4.1 Pêndulo invertido

[Chen (1999)] apresenta diversos exemplos práticos de sistemas dinâmicos cujas variáveis são números reais. Em geral, otimizam-se funções-custo quadráticas restritas a estes sistemas, como em [Phillips & Nagle (1995)]. Só que, neste trabalho, vamos considerar uma função custo linear, e um estudo com funções custo quadráticas fica para trabalhos futuros[1].

Escolhemos o problema do pêndulo invertido em uma carroça. Este mecanismo simples pode ser usado para modelar a decolagem de um veículo espacial, por exemplo. No nosso caso, propomos minimizar a soma da força a ser exercida na carroça em cada estágio, quando o pêndulo está sujeito à ação do vento.

Fazendo a otimização pelas formulações propostas no capítulo anterior deste trabalho, obteremos o resultado ótimo a um custo computacional polinomial em função das variáveis de estado no estágio inicial e do número total de entradas (controles) do problema.

[1]Embora o ferramental para este estudo já esteja pronto, bastaria apenas implementar.

4.1.1 Apresentação do problema

O problema:

Considere uma carroça com um pêndulo invertido no seu topo, erguido por um fio rígido, como na figura 4.1. Por simplicidade, o carro e o pêndulo se movem apenas no plano e o atrito, a massa do fio rígido e a resistência do ar são desprezados. Vamos supor que o pêndulo tenha massa de 10 gramas, a carroça tenha massa de 1000 gramas e o comprimento do fio rígido seja de 0,15 metros. Considere a aceleração da gravidade como 9,8 metros por segundo ao quadrado.

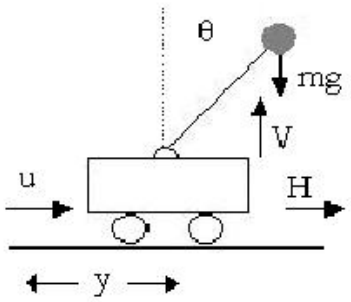

Figura 4.1: Carroça com um pêndulo invertido.

O problema consiste em manter o pêndulo na posição vertical. Por exemplo, se o pêndulo vai numa direção, a carroça deve deslocar-se na direção contrária, a fim de mantê-lo na posição vertical. Queremos encontrar qual o valor da força a ser aplicada na carroça a fim de que consigamos este resultado.

Consideraremos uma interferência no pêndulo (do vento, por exemplo), de modo que poderemos aplicar a força na carroça em apenas em um dos sentidos. Faremos isto para que a função custo seja linear, ou seja, para que este seja um problema de programação linear.

O modelo:

Seja M a massa da carroça, m a massa do pêndulo, l o comprimento do fio rígido, θ o ângulo que o pêndulo faz com a normal e g a aceleração da gravidade. Considere H e V as forças horizontal e vertical, respectivamente, que a carroça exerce no pêndulo, u a força a ser aplicada na carroça a fim que o pêndulo per-

maneça na posição vertical e y o deslocamento da carroça provocado pela força u.

Da aplicação das Leis de Newton para o movimento linear, segue que:

$$M\frac{d^2y}{dt} = u - H$$

$$H = m\frac{d^2}{dt}(y + l\sin\theta) = m\frac{dy}{dt} + ml\frac{d\theta}{dt}\cos\theta - ml(\frac{d\theta}{dt})^2\sin\theta$$

$$mg - V = m\frac{d^2}{dt^2}(l\cos\theta) = ml(-\frac{d^2\theta}{dt^2}\sin\theta - (\frac{d\theta}{dt})^2\cos\theta).$$

Da aplicação das Leis de Newton para o movimento rotacional do pêndulo pelo fio rígido, segue que:

$$mgl\sin\theta = ml\frac{d^2\theta}{dt^2}l + m\frac{d^2y}{dt^2}l\cos\theta.$$

Estas são equações não-lineares. Como o objetivo é manter o pêndulo na posição vertical, podemos assumir que θ e $d\theta/dt$ são pequenos (próximos de zero). Assim, podemos usar a aproximação $\sin\theta = \theta$ e $\cos\theta = 1$.

Considerando apenas os termos lineares em θ e $d\theta/dt$, ou seja, cancelando os termos em θ^2, $(d\theta/dt)^2$, $\theta(d\theta/dt)$ e $\theta(d^2\theta/dt^2)$, obtemos das equações acima que:

$$M\frac{d^2y}{dt} = u - m\frac{dy}{dt} + ml\frac{d\theta}{dt}$$

$$mg = V$$

$$g\theta = l\frac{d^2\theta}{dt^2} + \frac{d^2y}{dt^2}$$

e, portanto:

$$M\frac{d^2y}{dt} = u - mg\theta \tag{4.1}$$

e

$$Ml\frac{d^2\theta}{dt^2} = (M+n)g\theta - u. \tag{4.2}$$

O sistema dinâmico:

Vamos obter a descrição do sistema dinâmico do pêndulo invertido, a partir das equações 4.1 e 4.2. Para isto, considere as seguintes variáveis de estado:

$$x_1 = y$$

$$x_2 = \frac{dy}{dt}$$

$$x_3 = \theta$$

$$x_4 = \frac{d\theta}{dt}.$$

A variável de controle ou entrada deste sistema é a força u a ser aplicada na carroça a fim de que o pêndulo permaneça na posição vertical.

Substituindo estas variáveis nas equações 4.1 e 4.2 e colocando-as na forma matricial, obtemos:

$$\begin{bmatrix} \dot{x_1}[k+1] \\ \dot{x_2}[k+1] \\ \dot{x_3}[k+1] \\ \dot{x_4}[k+1] \end{bmatrix} =$$

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -mg/M & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & (M+m)g/Ml & 0 \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \\ x_3[k] \\ x_4[k] \end{bmatrix} +$$

$$\begin{bmatrix} 0 \\ 1/M \\ 0 \\ -1/Ml \end{bmatrix} \begin{bmatrix} u[k] \end{bmatrix}.$$

ou na forma sintética:

$$\boxed{\dot{x} \;=\; Mx + Nu.}$$

Esta é a descrição de um sistema contínuo no tempo, como em (2.3). Como queremos trabalhar com sistemas discretos no tempo, vamos fazer a discretização deste sistema.

Para isto, consideremos um instante de tempo h (pequeno) tal que a derivada $\dot{x}$ possa ser aproximada, em cada instante de tempo t, por:

$$\dot{x}(t) = \frac{dx}{dt}(t) = \frac{x(t+h) - x(t)}{h}.$$

Vamos assumir um mesmo período h para o sistema. A variável k será o indexador do tempo ($k = 0, 1, \ldots, N-1$) e corresponderá ao instante no tempo discreto e N será o horizonte, ou número de tempos ou estágios em que a entrada será aplicada.

Assim, a entrada discreta será denotada por $u[k] = u[kh]$ e o estado discreto será denotado por $x[k] = x[kh]$.

Podemos, então, escrever o sistema discretizado como:

$$\frac{x[k+1] - x[k]}{h} \;=\; Mx[k] + Nu[k],$$

e, portanto:

$$\boxed{x[k+1] \;=\; Ax[k] + Bu[k].}$$

sendo que a matriz $A = I - hM$ (I é a matriz identidade) e a matriz $B = hN$.

Ou seja, o sistema dinâmico discretizado é:

$$\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \\ x_3[k+1] \\ x_4[k+1] \end{bmatrix} =$$

$$\begin{bmatrix} 1 & h & 0 & 0 \\ 0 & 1 & -hmg/M & 0 \\ 0 & 0 & 1 & h \\ 0 & 0 & h(M+m)g/Ml & 1 \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \\ x_3[k] \\ x_4[k] \end{bmatrix} +$$

$$\begin{bmatrix} 0 \\ h/M \\ 0 \\ -h/Ml \end{bmatrix} \begin{bmatrix} u[k] \end{bmatrix}.$$

A meta de programação a ser alcançada no estágio N será:

$$x^* = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Em princípio, vamos considerar $h = 0,010$ e $N = 10$ estágios.

Como queremos um problema linear, vamos considerar, em cada estágio de tempo, a interferência do vento no pêndulo. Assim, poderemos considerar o controle u apenas em um sentido.

Em cada estágio, vamos supor que o vento provoque uma interferência fixa, por exemplo de $-0,125$, no ângulo que o pêndulo faz com a normal. Esta interferência pode ser modelada como o vetor:

$$w = \begin{bmatrix} 0 \\ 0 \\ -0,125 \\ 0 \end{bmatrix}$$

O sistema dinâmico, então, será:

$$\boxed{x[k+1] \;=\; Ax[k] + Bu[k] + w.}$$

Funções objetivo e restrições:

Propomos as seguintes funções objetivo para este problema:

Função objetivo 1: A política ótima 1 consiste em minimizar a soma quadrática da força a ser aplicada na carroça em cada estágio de tempo a fim de que o pêndulo permaneça na posição vertical. Esta política pode ser escrita como:

$$\min_{x[0],u[0],\ldots,u[N-1]} \sum_{i=0}^{N-1} u[i]^2.$$

Função objetivo 2: Se a variável de controle for em apenas um dos sentidos (ou não-negativa ou não-positiva), podemos considerar a função custo linear. A política ótima 2 consiste em minimizar a soma da força a ser aplicada na carroça em cada estágio de tempo a fim de que o pêndulo permaneça na posição vertical. Esta política pode ser escrita como:

$$\min_{x[0],u[0],\ldots,u[N-1]} \sum_{i=0}^{N-1} u[i].$$

Função objetivo 3: A política ótima 3 consiste em minimizar a soma quadrática do ângulo de deslocamento do pêndulo no estágio inicial. Considere o vetor $c = [0010]$. Esta política ótima pode ser escrita como:

$$\min_{x[0]} cx[0]^2.$$

As restrições deste problema são:

1. restrição do sistema dinâmico: $x[k+1] = Ax[k] + Bu[k] + w,\ k = 1, \ldots, n.$
2. meta de programação: $x[N] = x^*$.
3. restrição na variável de controle: $u[k] \geq 0$, para cada k.
4. restrição de folga: depois de N estágios de tempo, o estado deve ter atingido a meta x^* proposta, com um erro de $p\%$, para um p dado. Considerando $||.||$ uma norma qualquer, e ϵ um número positivo pequeno (relacionado com p), esta restrição pode ser modelada como: $||x[N] - x^*|| \leq \epsilon$.

4.1.2 Formulações propostas

Podemos seguir a mesma idéia da proposição 3.4.1, para afirmar que a restrição do sistema dinâmico neste caso será:

$$A^N x[0] + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] + w + \sum_{k=1}^{N-1} A^k w = x^*.$$

Este é um problema de programação dinâmica linear com a ação de controle. Vamos aplicar as duas formulações propostas neste trabalho (seções 3.4.1 e 3.4.2) para este problema.

Vamos apresentar as formulações e os resultados apenas para a função objetivo 2.

FORMULAÇÃO 1:
Otimização restrita à pré-imagem da meta de programação (considerando a igualdade $x[N] = x^*$ na restrição 4):

$$\min_{x[0],u[0],\ldots,u[N-1]} \sum_{k=0}^{N-1} u[k]$$

$$\text{sujeito a: } \begin{cases} A^N x[0] + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] + w + \sum_{k=1}^{N-1} A^k w = x^* \\ u[k] \geq 0, \quad \text{para cada } k. \end{cases}$$

FORMULAÇÃO 2:
Otimização restrita à pré-imagem de um hipercubo em torno da meta (considerando a norma do máximo na restrição 4):

$$\min_{x[0],u[0],\ldots,u[N-1]} \sum_{k=0}^{N-1} u[k]$$

$$\text{sujeito a: } \begin{cases} e_i^t(A^N x[0] - (x^* + e_i\epsilon)) + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] \\ \quad +w + \sum_{k=1}^{N-1} A^k w \leq 0, \quad \text{para cada } i \\ -e_i^t(A^N x[0] - (x^* - e_i\epsilon)) + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] \\ \quad +w + \sum_{k=1}^{N-1} A^k w \geq 0, \quad \text{para cada } i \\ u[k] \geq 0, \quad \text{para cada } k. \end{cases}$$

4.1.3 Comparações entre os métodos e comentários

Vamos apresentar os resultados obtidos variando o ângulo da interferência do vento (terceira coordenada do vetor w: w_3) e, na formulação 2, o raio (ϵ) do hipercubo em torno da meta x^*.

A meta x^* será a mesma em todos os casos: o vetor nulo. Quando não especificarmos, estaremos considerando o espaçamento da discretização $h = 0,01$ e o número de estágios $N = 10$.

Os dados para a formulação 1 com $w_3 = -0,125$ estão nas tabelas 4.1 e 4.2. A evolução do vetor de estados está na figura 4.2. O tempo gasto neste processo foi de 0,6820 segundos e a função objetivo foi $J^* = 10^{-15}$.

	x_1	x_2	x_3	x_4
$x[0]$	-0,0003	0,0073	1,4428	-4,9443
$x[10]$	10^{-15}	10^{-14}	10^{-14}	10^{-13}

Tabela 4.1: Vetor de estados da formulação 1 do problema do pêndulo com $w_3 = -0,125$.

$u[0]$	$u[1]$	$u[2]$	$u[3]$	$u[4]$	$u[5]$	$u[6]$	$u[7]$	$u[8]$	$u[9]$
10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}

Tabela 4.2: Seqüência de controles da formulação 1 do problema do pêndulo com $w_3 = -0,125$.

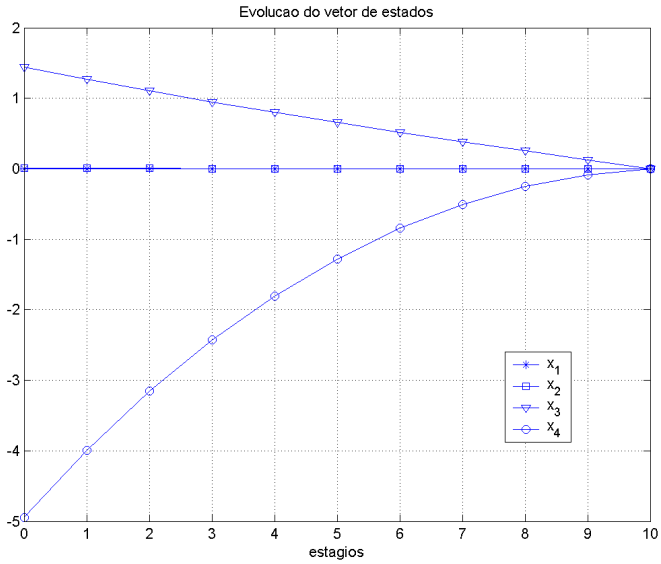

Figura 4.2: Evolução de vetor de estados da formulação 1 do problema do pêndulo com $w_3 = -0,125$.

Os dados para a formulação 2 com o mesmo parâmetros $w_3 = -0,125$ e raio do politopo $\epsilon = 0,1$ estão nas tabelas 4.3 e 4.4. A evolução do vetor de estados está na fugura 4.3. O tempo gasto neste processo foi de 0,34 segundos e a função objetivo foi $J^* = 10^{-15}$.

	x_1	x_2	x_3	x_4
$x[0]$	0,0039	-0,0330	1,3232	-4,2168
$x[10]$	0,0002	-0,0394	-0,0776	0,0983

Tabela 4.3: Vetor de estados da formulação 2 do problema do pêndulo com $w_3 = -0,125$ e $\epsilon = 0,1$.

$u[0]$	$u[1]$	$u[2]$	$u[3]$	$u[4]$	$u[5]$	$u[6]$	$u[7]$	$u[8]$	$u[9]$
10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}

Tabela 4.4: Seqüência de controles da formulação 2 do problema do pêndulo com $w_3 = -0,125$ e $\epsilon = 0,1$.

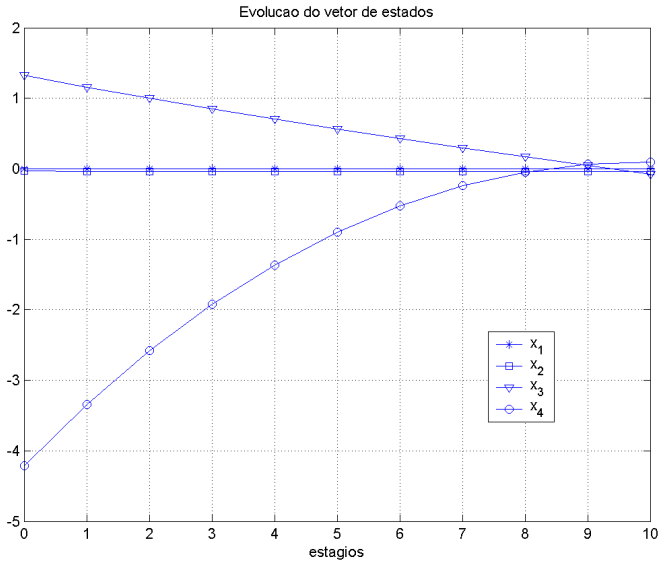

Figura 4.3: Evolução de vetor de estados da formulação 2 do problema do pêndulo com $w_3 = -0,125$ e $\epsilon = 0,1$.

Embora tenham chegado (aproximadamente) no mesmo resultado, a formulação 1 gastou o dobro do tempo da formulação 2 para o mesmo parâmetro $w_3 = -0,125$. A presença da restrição de igualdade, como comentado no capítulo anterior, pode atrasar muito a convergência do método.

Como as variáveis do problema são contínuas (números reais), estes resultados obtidos são s ótimos.

Se aumentarmos o raio do politopo para $\epsilon = 10$ obteremos os resultados das tabelas 4.5 e 4.6. O tempo gasto neste processo foi de 0,31 segundos e a função objetivo foi $J^* = 10^{-16}$. O tempo e o resultado foram basicamente os mesmos. Isto se deve ao fato da matriz A do sistema dinâmico ter raio espectral[2] maior que 1. Assim, as pré-imagens no estágio inicial são menores que as respectivas imagens no tempo final.

	x_1	x_2	x_3	x_4
$x[0]$	0,0113	-0,1082	1,0322	-2,4835
$x[10]$	0,0002	-0,1123	-0,2704	0,2876

Tabela 4.5: Vetor de estados da formulação 2 do problema do pêndulo com $\epsilon = 10$.

$u[0]$	$u[1]$	$u[2]$	$u[3]$	$u[4]$	$u[5]$	$u[6]$	$u[7]$	$u[8]$	$u[9]$
10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}

Tabela 4.6: Seqüência de controles da formulação 2 do problema do pêndulo com $\epsilon = 10$.

Vamos agora variar a interferência do vento. Os dados para a formulação 1 com $w_3 = +0,125$ estão nas tabelas 4.7 e 4.8. O tempo gasto neste processo foi de 0,581 segundos e a função objetivo foi $J^* = 10^{-15}$.

	x_1	x_2	x_3	x_4
$x[0]$	0,0003	-0,0073	-1,4428	4,9443
$x[10]$	10^{-15}	-10^{-14}	-10^{-15}	10^{-14}

Tabela 4.7: Vetor de estados da formulação 1 do problema do pêndulo com $w_3 = +0,125$.

[2]Maior autovalor real em módulo.

$u[0]$	$u[1]$	$u[2]$	$u[3]$	$u[4]$	$u[5]$	$u[6]$	$u[7]$	$u[8]$	$u[9]$
10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}

Tabela 4.8: Seqüência de controles da formulação 1 do problema do pêndulo com $w_3 = +0,125$.

Os dados para a formulação 2 com o mesmo parâmetros: $w_3 = +0,125$ e $\epsilon = 0,1$ estão nas tabelas 4.9 e 4.10. O tempo gasto neste processo foi de 0,311 segundos e a função objetivo foi $J^* = 10^{-15}$.

	x_1	x_2	x_3	x_4
$x[0]$	0,0017	-0,0208	-1,3043	4,1824
$x[10]$	0,0001	-0,0146	0,0986	-0,0078

Tabela 4.9: Vetor de estados da formulação 2 do problema do pêndulo com $w_3 = +0,125$ e $\epsilon = 0,1$.

$u[0]$	$u[1]$	$u[2]$	$u[3]$	$u[4]$	$u[5]$	$u[6]$	$u[7]$	$u[8]$	$u[9]$
10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}	10^{-16}

Tabela 4.10: Seqüência de controles da formulação 2 do problema do pêndulo com $w_3 = +0,125$ e $\epsilon = 0,1$.

Se aumentarmos o raio do politopo para $\epsilon = 10$ obteremos os resultados das tabelas 4.11 e 4.12.
O tempo gasto neste processo foi de 0,31 segundos e a função objetivo foi $J^* = 10^{-16}$.

Também aqui percebemos que, como o raio espectral da matriz A é maior que 1, as pré-imagens no estágio inicial são menores que as respectivas imagens no tempo final. Assim, claro, mesmo com outros valores de w_3, a relaxação na meta não faria tanta diferença.

Agora, observe que o problema colocado desta maneira "controlou o sistema"' pelo estado inicial $x[0]$ e "zerou"' as variáveis de controle (o objetivo era mini-

	x_1	x_2	x_3	x_4
$x[0]$	0,0111	-0,1091	-0,1509	-2,1694
$x[10]$	0,0001	-0,1122	-0,9190	0,0820

Tabela 4.11: Vetor de estados da formulação 2 do problema do pêndulo com $\epsilon = 10$.

$u[0]$	$u[1]$	$u[2]$	$u[3]$	$u[4]$	$u[5]$	$u[6]$	$u[7]$	$u[8]$	$u[9]$
10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}	10^{-17}

Tabela 4.12: Seqüência de controles da formulação 2 do problema do pêndulo com $\epsilon = 10$.

mizá-las). Ou seja, aparentemente não houve ação de controle, mas o controle foi feito através da condição inicial do sistema.

Vamos então retirar o estado inicial da otimização e colocá-lo como um parâmetro de entrada. Para isto, basta substituir formalmente nas restrições das formulações 1 e 2 acima o valor dado (como entrada) para $x[0]$.

Na tabela 4.13 apresentamos os resultados da formulação 1 para o vetor $x[0]$ nulo como um parâmetro de entrada e $w_3 = -0,125$. O problema foi infactível e não encontrou o ótimo. Por isso, nem vamos apresentar a seqüência de controles.

	x_1	x_2	x_3	x_4
$x[0]$	0	0	0	0
$x[10]$	3,1939	71,0106	-23,0824	-511,1914

Tabela 4.13: Vetor de estados da formulação 1 do problema do pêndulo com $w_3 = -0,125$.

Apresentamos os dados da formulação 2 para estes mesmos parâmetros acima com raio do hipercubo de $\epsilon = 0,5$ nas tabelas 4.14 e 4.15. O tempo gasto neste processo foi de 0,31 segundos e a função objetivo foi $J^* = 10^{-5}$.

Vamos considerar agora como parâmetros, além do $x[0] = [0\ 0\ 0,5\ 0]^t$ como

	x_1	x_2	x_3	x_4
$x[0]$	0	0	0	0
$x[10]$	0	0	-1,3	-1,8

Tabela 4.14: Vetor de estados da formulação 2 do problema do pêndulo com $\epsilon = 0,5$.

$u[0]$	$u[1]$	$u[2]$	$u[3]$	$u[4]$	$u[5]$	$u[6]$	$u[7]$	$u[8]$	$u[9]$
10^{-5}	10^{-6}	10^{-6}	10^{-6}	10^{-7}	0	10^{-9}	10^{-9}	10^{-8}	10^{-8}

Tabela 4.15: Vetor de estados da formulação 2 do problema do pêndulo com $\epsilon = 0,5$.

está nas tabelas 4.16 e 4.17, $h = 0,001$, $N = 20$ e $w_3 = -0,0125$.

Os dados da formulação 1 estão na tabela 4.16. O problema ainda foi infactível (apesar de $x[20]$ estar bem próximo da meta x^*).

	x_1	x_2	x_3	x_4
$x[0]$	0	0	0,5	0
$x[20]$	0	0,08	0,26	0

Tabela 4.16: Vetor de estados da formulação 1 do problema do pêndulo com $h = 0,001$, $N = 20$ e $w_3 = -0,0125$.

Os dados da formulação 2 com raio $\epsilon = 0,5$ para estes mesmos parâmetros estão na tabela 4.17. O problema encontrou o ótimo em 0,32 segundos. A evolução do vetor de estados está na figura 4.4. A seqüência dos 20 estágios de controle está na figura 4.5, e a função objetivo (soma de todos os $u[k]$) foi de apenas 796,2967.

Repare que, nos dois casos, a relaxação na meta (formulação 2) não foi tão significativa (apenas de 0,5), mas fez com que o problema ficasse factível (encontrou a solução ótima).

Nota 4.1.1 *De fato, isto pode acontecer outras vezes: um problema ser infactível*

	x_1	x_2	x_3	x_4
$x[0]$	0	0	0,5	0
$x[20]$	0	0	0,25	0,5

Tabela 4.17: Vetor de estados da formulação 2 do problema do pêndulo com $\epsilon = 0,5$, $h = 0,001$, $N = 20$ e $w_3 = -0,0125$.

na formulação 1, por causa da sua restrição de igualdade, mas o problema com uma pequena relaxação na meta (formulação 2) já ser factível.

4.2 O problema da mochila

O problema da mochila (*knapsack problem*) é um problema clássico de otimização combinatória e de muita importância prática por sua aplicação em diversos segmentos, como: investimento de capitais, carregamento de veículos, orçamento, etc. Só que este problema é de complexidade não-polinomial, pertence à classe

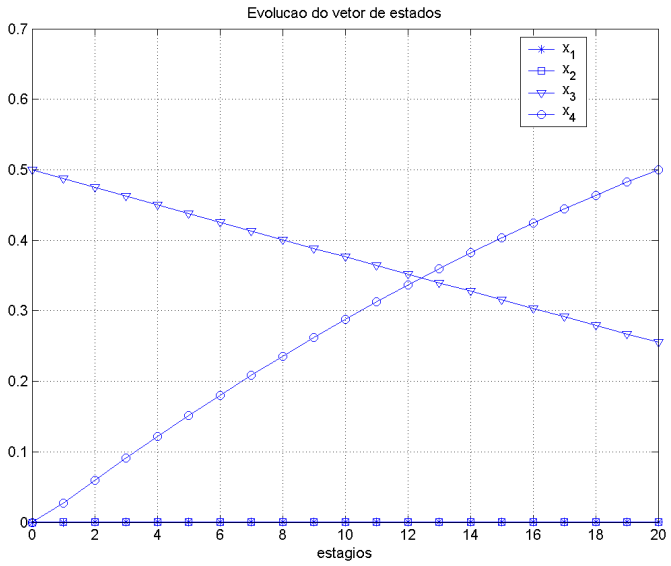

Figura 4.4: Evolução de vetor de estados da formulação 2 do problema do pêndulo com $x[0] = [0\ 0\ 0,5\ 0]^t$, $w_3 = -0,125$ e $\epsilon = 0,1$.

NP-árduo[3].

Metaforicamente, podemos entender este problema como o desafio de encher uma mochila sem ultrapassar um determinado limite de peso, otimizando o valor do produto carregado. Este produto, em geral, está disponível em embalagens indivisíveis.

[Goldbarg & Luna (2000)], dentre outros, apresentam a formulação tradicional deste problema como programação inteira. Se usarmos a mesma formulação proposta na programação inteira, mas permitindo que as variáveis sejam números reais (ou seja, se permitimos a relaxação linear), temos o problema da mochila linear.

[Mascó (2001)] apresenta uma formulação para este problema usando programação dinâmica. Entretanto, ele coloca que é viável resolvê-lo assim apenas se o número de produtos for menor que 15. Feita a relaxação linear e usando a formulação por nós proposta, este número pode ser bem maior. Mostraremos que o resultado obtido será o mesmo do problema da mochila linear.

[3][Goldbarg & Luna (2000)] definem o que quer dizer um problema pertencer à classe NP-árduo.

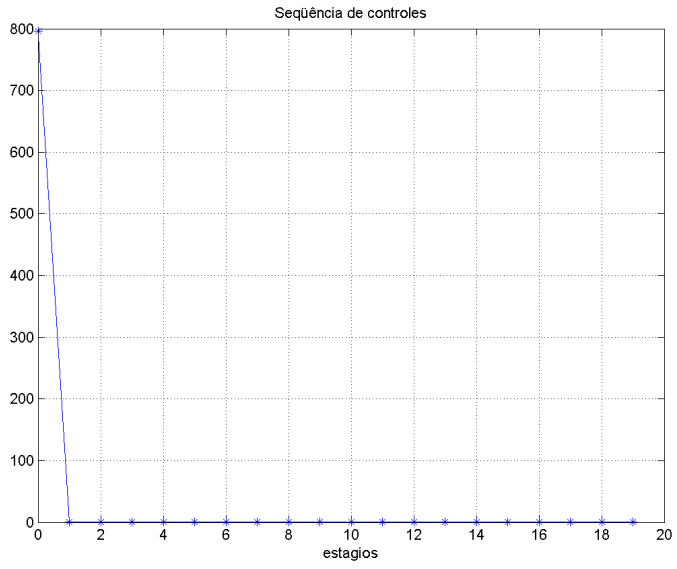

Figura 4.5: Seqüência de controles da formulação 2 do problema do pêndulo com $x[0] = [0\ 0\ 0,5\ 0]^t$, $w_3 = -0,125$ e $\epsilon = 0,1$.

4.2.1 Apresentação do problema

O problema:

Vamos, aqui, como em [Mascó (2001)], pensar no problema da mochila como o problema de transporte de carga (carregamento de veículos)[4]. Dispomos de um furgão cuja carga máxima é de 900 quilos. O objetivo é transportar quatro produtos distintos que vêm em embalagens de 300, 500, 400 e 200 quilos, respectivamente. Estas embalagens são indivisíveis. O transporte destes produtos produzem um benefício de 80, 170, 120 e 50 unidades monetárias, respectivamente. Devemos escolher a carga que maximize o benefício de uma viagem.

O modelo:

Seja x_i, $i = 1, \ldots, n$ a quantidade de um elemento i a ser alocada e v_i o valor do elemento i por unidade. Considere, também, a_i o consumo de recurso por unidade do elemento i e b o recurso total.

O problema pode ser formulado como:

$$\max_{x_1,\ldots,x_n} \sum_{i=1}^{n} v_i x_i \tag{4.3}$$

$$\text{sujeito a: } \begin{cases} \sum_{i=1}^{n} a_i x_i \leq b \\ x_i \geq 0 \\ x_i \text{ inteiros.} \end{cases}$$

Neste caso, $b = 900$. Os demais dados da formulação estão na tabela 4.18.

Se existir apenas um objeto de cada tipo para ser escolhido, define-se o problema da mochila 0-1 (também conhecido por problema da mochila unidimensional ou problema da mochila binário), em que a restrição das variáveis inteiras é substituída por $x_i \in \{0, 1\}$. Se as variáveis são limitadas, o problema se chama problema da mochila limitado.

[4]Basta considerarmos o furgão como fazendo o papel da mochila.

v_1	v_2	v_3	v_4
80	170	120	50
a_1	a_2	a_3	a_4
300	500	400	200

Tabela 4.18: Dados do problema da mochila para a formulação como programação inteira.

Esta formulação para o problema tem complexidade computacional exponencial. Agora, o problema da mochila linear, em que as variáveis são números reais, pode ser resolvido de modo eficiente como um problema de programação linear qualquer.

O sistema dinâmico:

Considere as variáveis para o sistema dinâmico:

- cada estágio k corresponde a um dos elementos para ser alocado: $k = 1, \ldots, n$.
- cada variável de estado $x[k]$ corresponde à quantidade de recurso disponível para ser alocado no estágio k.
- cada variável de controle $u[k]$ corresponde à quantidade de recurso alocado no estágio k.

A equação do sistema dinâmico é:

$$\boxed{x[k+1] = x[k] - u[k]}$$

e $k = 1, \ldots, n$. Aqui, A $= 1$ e B $= -1$ (sistema unidimensional).

Função objetivo e restrições:

A função objetivo é maximizar a soma do custo de cada estágio. Seja v_k como definido acima. Então, o custo de cada estágio é:

$$L_k(x[k], u[k]) = v_k u[k].$$

As restrições são:

1. restrição do sistema dinâmico: $x[k+1] = x[k] - u[k],\ k = 1, \ldots, n$.
2. condição inicial: $x[1] = b$.
3. restrição da capacidade: $\sum_{i=1}^{n} u[k] \leq b$.
4. restrição nas variáveis de controle: $0 \leq u[k] \leq x[k]$, para cada k.
5. números inteiros: $u[k]$ inteiros, para cada k.
6. restrição de folga: depois de N estágios de tempo, o estado deve ter atingido a meta x^* proposta, com um erro de $p\%$, para um p dado. Considerando $||.||$ uma norma qualquer, e ϵ um número positivo pequeno (relacionado com o p), esta restrição pode ser modelada como: $||x[N] - x^*|| \leq \epsilon$.

4.2.2 Formulações propostas

Antes de propormos nossas formulações, vamos mostrar a formulação do problema proposta por [Mascó (2001)]:

FORMULAÇÃO TRADICIONAL:

$$\max_{u[1],\ldots,u[n]} \sum_{i=1}^{n} v_i u[i]$$

$$\text{sujeito a: } \begin{cases} x[k+1] = x[k] - u[k],\ k = 1, \ldots, n \\ x[1] = b \\ \sum_{i=1}^{n} u[k] \leq b \\ 0 \leq u[k] \leq x[k], \text{ para cada } k \\ u[k] \text{ inteiros, para cada } k. \end{cases}$$

Vamos, agora, apresentar as formulações propostas neste trabalho (seções 3.4.1 e 3.4.2) para este problema. Para isto, vamos permitir a relaxação linear e vamos considerar b como a meta de programação x^*, em vez de uma condição inicial. Então, a matriz A do sistema dinâmico será -1 e a matriz B será 1.

Para escrever a variável de estado da restrição 4 em função do estado inicial, precisaremos usar a proposição 3.4.1. Substituindo cada $x[k]$, esta restrição ficaria assim:

$$0 \leq u[K] \leq A^K x[0] + Bu[K-1] + \sum_{k=1}^{K-1} A^k B u[N-(k+1)].$$

Mas, vamos considerá-la, para cada estágio K, assim:
$\begin{cases} A^K x[0] - u[K] + Bu[K-1] + \sum_{k=1}^{K-1} A^k Bu[N-(k+1)] \geq 0, \\ u[K] \geq 0. \end{cases}$

FORMULAÇÃO 1:
Otimização restrita à pré-imagem da meta de programação (considerando a igualdade $x[N] = b$ na restrição 6):

$$\max_{x[0],u[1],\ldots,u[n]} \sum_{i=1}^{n} v_i u[i]$$

$$\text{sujeito a:} \begin{cases} A^N x[0] + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] = b \\ \sum_{k=1}^{K-1} u[k] \leq b \\ A^K x[0] - u[K] + Bu[K-1] + \sum_{k=1}^{K-1} A^k Bu[N-(k+1)] \geq 0 \\ \quad \text{para cada } K \\ u[k] \geq 0, \quad \text{para cada } k. \end{cases}$$

FORMULAÇÃO 2:
Otimização restrita à pré-imagem de um hipercubo em torno da meta (considerando a norma do máximo na restrição 6):

$$\max_{x[0],u[1],\ldots,u[n]} \sum_{i=1}^{n} v_i u[i]$$

$$\text{sujeito a:} \begin{cases} e_i^t(A^N x[0] - (b + e_i\epsilon)) + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] \leq 0, \\ \quad \text{para cada i} \\ -e_i^t(A^N x[0] - (b - e_i\epsilon)) + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] \geq 0, \\ \quad \text{para cada i} \\ A^K x[0] - u[K] + Bu[K-1] + \sum_{k=1}^{K-1} A^k Bu[N-(k+1)] \geq 0 \\ \quad \text{para cada } K \\ u[k] \geq 0, \quad \text{para cada } k. \end{cases}$$

Os dados para estas três formulações estão na tabela 4.19.

v_1	v_2	v_3	v_4	b
80/300	170/500	120/400	50/200	900

Tabela 4.19: Dados do problema da mochila para as formulações como programação dinâmica.

4.2.3 Comparações entre os métodos e comentários

Primeiro, vamos apresentar o resultado da formulação tradicional. Ou seja, vamos resolver este problema da mochila como um problema de programação dinâmica segundo o algoritmo (2.2.1). O resultado seria o mesmo se resolvêssemos como um problema de programação inteira (formulação 4.3).

Precisamos primeiro montar a árvore de possibilidades do problema. Vamos listar todos os estados com os respectivos controles ou entradas possíveis. Os estados possíveis são os pesos admissíveis no furgão, ou seja, de 0 até 900. O estado inicial é $x[0] = 900$. Nas figuras a seguir, os estados estão representados por bolas.

Os controles possíveis variam com os estágios. No primeiro estágio podemos tirar (segundo a lógica da formulação) embalagens de 300 quilos, no segundo, de 500 quilos, no terceiro, de 400, e no quarto estágio podemos tirar da mochila embalagens de 200 quilos. Nas figuras abaixo, os controles estão representados por quadrados. O benefício parcial associado à seqüência de controles até cada estágio está depois dos estados de chegada dentro de quadrados duplos.

Vamos tomar a primeira decisão. A partir do estado inicial de 900, podemos tirar da mochila 0, 1, 2 ou 3 embalagens de 300 quilos, como na figura 4.6. O benefício associado é de 80 unidades por embalagem.

Os estados possíveis no segundo estágio são aqueles atingíveis pela ação de controle do primeiro estágio, ou seja, 900, 600, 300 e 0. A ação de controle aqui diz para tirar embalagens de 500 quilos com um benefício associado de 170 unidades por embalagem. Estes estados, os controles possíveis e os benefícios parciais associados as ações de controle dos dois estágios estão na figura 4.7.

A ação de controle do terceiro estágio é tirar embalagens de 400 quilos com um benefício associado de 120 unidades por embalagem. Se é possível fazer a transição de estágios, chegando num estado por mais de um caminho, vamos, claro, escolher o de maior benefício, de modo que, em cada estágio, só "chega

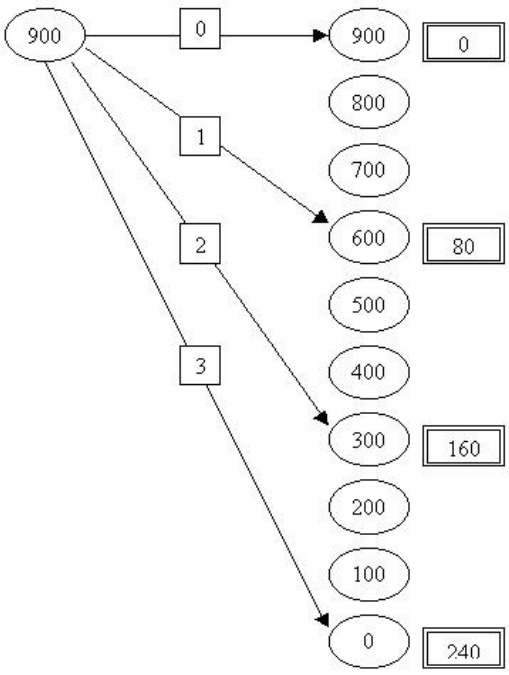

Figura 4.6: Primeiro estágio de decisão do problema da mochila.

uma flecha"em cada estado. Por exemplo, a partir do estado 900, poderíamos tirar 2 embalagens e chegar no estado 100 com um benefício de 240 unidades. Mas, "o caminho" anterior já conseguiu chegar neste estado com um benefício de 250 unidades. Portanto, é melhor continuar neste "caminho". A figura 4.8 lista esta, e as demais ações deste estágio.

Por fim, a ação de controle do quarto e último estágio é tirar embalagens de 200 quilos com um benefício associado de 50 unidades por embalagem. Também aqui não vamos considerar as ações de controle que chegariam num estado com um benefício menor que o que já estava em seu quadrado duplo. Além disso, se duas ações chegariam num mesmo estado, escolheremos a que chegar com o maior benefício. Veja as ações na figura 4.9.

A seqüência de controles com a seqüência ótima em destaque está na figura 4.10. Ou seja, precisamos levar 1 embalagem do produto 2 (referente ao estágio 2) e uma embalagem do produto 3 (referente ao estágio 3). O benefício será de 290 unidades monetárias.

Vamos, agora, apresentar a resolução da relaxação linear do problema de programação inteira (4.3), ou seja, vamos resolvê-lo como um problema de pro-

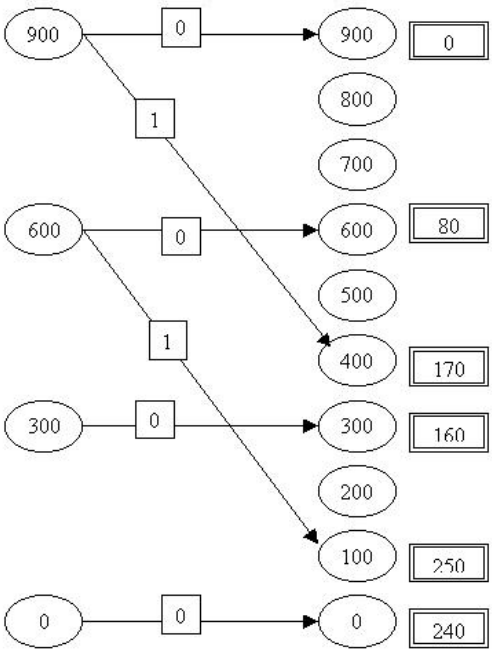

Figura 4.7: Segundo estágio de decisão do problema da mochila.

gramação linear, permitindo que as variáveis sejam números reais (respeitando as limitações do computador). A função objetivo foi $J^* = 306$.

Os dados estão na tabela 4.20.

x_1	x_2	x_3	x_4	tempo (s)
0	1,8	0	0	0,26

Tabela 4.20: Solução deste problema da mochila relaxado.

Ou seja, se permitirmos dividir as embalagens, deveríamos levar 1,8 embalagens do produto 2 na mochila a um benefício de 306 unidades monetárias. A diferença seria de 16 unidades monetárias em relação ao resultado inteiro (resolução do problema com as variáveis inteiras).

Se fizermos a aproximação $x_2 = 1$, a função objetivo será bem menor: $J = 170$ (diferença de 120 unidades monetárias para menos), mas se pudermos dar uma "apertada na mochila" e permitirmos $x_2 = 2$, a função objetivo será $J = 340$

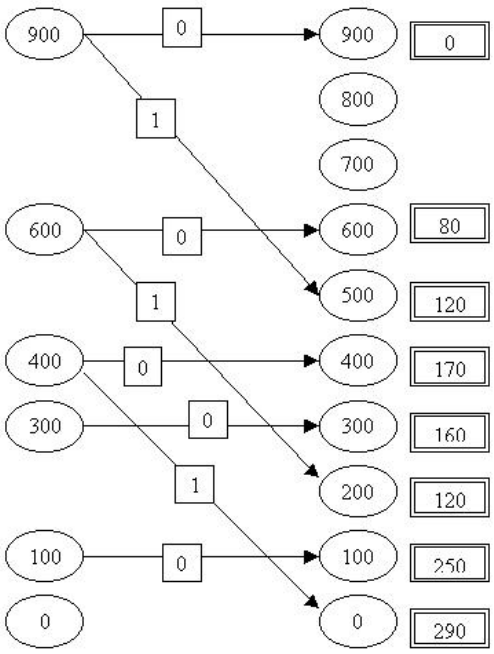

Figura 4.8: Terceiro estágio de decisão do problema da mochila.

(diferença de 50 unidades para mais).

Ficando com $x_2 = 1$, vamos procurar quais outras embalagens poderiam entrar na mochila sem ultrapassar sua capacidade. Descobriremos que poderia entrar qualquer outro produto (poderíamos ter qualquer outra coordenada x_i igual a 1) e, claro, vamos escolher a que der maior retorno, ou seja, vamos escolher o produto 3 ($x_3 = 1$). Este é o resultado ótimo do problema inteiro.

Finalmente, vamos apresentar o resultados das formulações propostas. Os dados da formulação 1 estão na tabela 4.21. A meta de programação x^* é 900.

	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$
$x[k]$	0	0	900	900	900
$u[k]$	0	900	0	0	-

Tabela 4.21: Solução deste problema da mochila pela formulação 1.

A função objetivo foi de 306 unidades monetárias, como na resolução pela

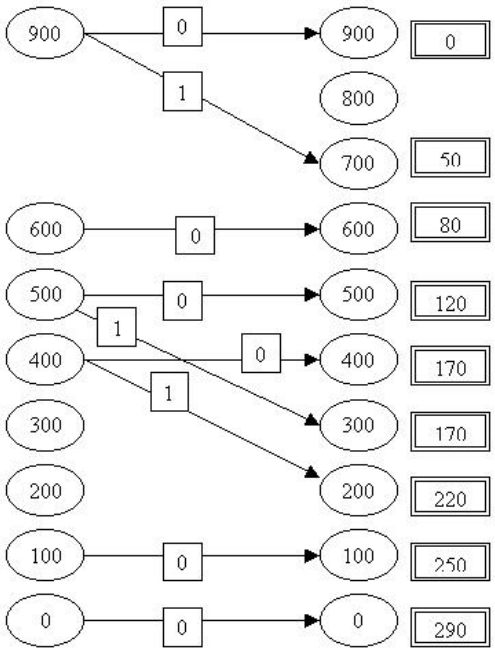

Figura 4.9: Quarto estágio de decisão do problema da mochila.

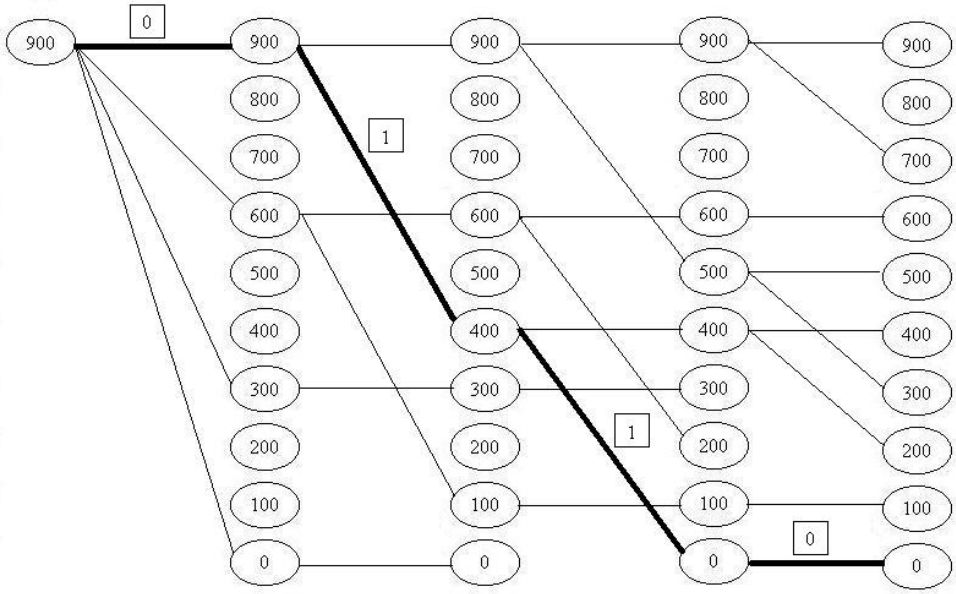

Figura 4.10: Seqüência de controles do problema da mochila com destaque para a seqüência ótima.

relaxação linear do problema de programação inteira. O tempo gasto foi de $0,29$ segundos.

Nota 4.2.1 *A formulação 1 apresentou o mesmo resultado do problema da mochila linear.*

Testamos a formulação 2 para diferentes raios do hipercubo em torno da meta $x^* = 900$. Os dados estão na tabela 4.22.

raio		$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	J
0,1	$x[k]$	0	0	900,1	900,1	900,1	306,034
	$u[k]$	0	900,1	0	0	-	
1,0	$x[k]$	0	0	901	901	901	306,34
	$u[k]$	0	901	0	0	-	
10,0	$x[k]$	0	0	910	910	910	309,4
	$u[k]$	0	910	0	0	-	

Tabela 4.22: Solução deste problema da mochila pela formulação 2.

O tempo gasto em cada processo foi de $0,30$ segundos.

A relaxação na meta não apresentou um resultado muito expressivo pois a matriz A do sistema dinâmico (corresponde ao número 1, coeficiente de $x[k]$) tem raio espectral igual a 1. Assim, pré-imagem do conjunto em torno da meta tem a mesma dimensão deste conjunto em torno da meta, ou seja, a região não aumenta, como gostaríamos. Portanto, a variação nos estados e nos controles é da mesma ordem de grandeza do que for permitido na relaxação.

4.3 Evolução de um rebanho de gado no tempo

A evolução de um rebanho de gado no tempo pode ser modelada como um sistema dinâmico discreto linear. Através deste modelo, pode-se procurar a política ótima que tenha o mínimo custo inicial para aquisição do rebanho e o máximo aproveitamento dos recursos da fazenda. Esta mesma abordagem também pode ser empregada em diversos rebanhos com diferentes funções objetivo e restrições.

Só que, como as variáveis do problema são números inteiros, propomos fazer a relaxação linear. Assim, vamos resolver através das formulações propostas no capítulo anterior considerando que os números são reais. Encontraremos como resultado uma cota inferior para a função objetivo do problema com variáveis inteiras.

[Takahashi et al. (1997)] estudaram este problema. Eles apresentaram a estrutura do sistema dinâmico sem a ação de controle, propuseram as três primeiras funções objetivo e as quatro primeiras restrições que se seguem e estudaram a formulação (3.4) para o problema sem a ação de controle.

4.3.1 Apresentação do problema

O problema:

Sr. Caldeira, um importante professor da UFMG, pretende se aposentar e dedicar os anos que lhe restam como um fazendeiro no interior mineiro. Ele possui um grande interesse pela criação de gado, uma atividade economicamente rentável que lhe permitirá gozar seus dias com maior tranqüilidade. Não satisfeito com a maneira tradicional de se começar um rebanho, ele procura um método de otimizar o risco de investimento. Quer começar com um quantidade de gado mínima para que, ao fim de 7 anos, ele tenha o seu território devidamente ocupado, a saber, com 1000 cabeças. Como bom entendedor do processo, ele sabe que 7 anos é um tempo médio de estabilidade do rebanho. Vamos considerar o caso em que o Sr. Caldeira quer adquirir o rebanho apenas na fase inicial e não pretende negociá-lo durante o processo (caso sem ação de controle) e o caso em que ele pretende negociar o rebanho durante o processo (caso com ação de controle). Devido à limitações orçamentárias, no caso em que ele vai negociar, ele pode comprar até 10 cabeças ou vender até 10 cabeças por ano.

O modelo:

Consideramos neste trabalho o modelo de evolução no tempo de um rebanho de gado por um sistema dinâmico linear proposto por [Carvalho (1972)]. O caso brasileiro foi estudado por [Mascolo (1980)]. Este modelo tem como base as seguintes premissas:

Premissa 1: O sistema de produção é intensivo ou semi-intensivo.
Premissa 2: Existe um método de inseminação artificial.

Premissa 3: Todos os machos são vendidos com idade S, que varia de acordo com o tipo do rebanho. Num rebanho típico, esta idade é de 3 anos.
Premissa 4: Nenhuma fêmea é vendida antes de 3 anos. Uma fração fixa das fêmeas acima desta idade são descartadas (vendidas, por exemplo) a cada ano. Isto deve acontecer a fim de alcançar a estabilidade do rebanho, depois de um certo número de anos.

Os parâmetros do modelo são:

m_1 : um menos a taxa de mortalidade para a faixa etária de 0 a 1 ano
m_2 : um menos a taxa de mortalidade para a faixa etária de 1 a 2 anos
m_3 : um menos a taxa de mortalidade para a faixa etária de 2 a 3 anos
m_4 : um menos a taxa de mortalidade para a faixa etária acima de 3 anos
f : taxa de natalidade para vacas acima de 3 anos
f_3 : taxa de natalidade para vacas com 3 anos
R : taxa de descarte anual das fêmeas acima de 3 anos

O sistema dinâmico:

Considere k como a variável de tempo, sendo que $k = 0$ denota o estágio inicial, $k = 1$ denota um ano depois, e assim sucessivamente.

A variável de estado do problema é um vetor $x[k]$, para cada estágio k, com oito posições. Cada uma das posições representa uma faixa etária de cada um dos sexos, como descrito abaixo. O valor de cada posição representa a quantidade de indivíduos deste estado.

$x_1[k]$: número de fêmeas entre 0 e 1 ano no estágio k
$x_2[k]$: número de fêmeas entre 1 e 2 anos no estágio k
$x_3[k]$: número de fêmeas entre 2 e 3 anos no estágio k
$x_4[k]$: número de fêmeas acima de 3 anos com bezerros no estágio k
$x_5[k]$: número de fêmeas acima de 3 anos sem bezerros no estágio k
$x_6[k]$: número de machos entre 0 e 1 ano no estágio k
$x_7[k]$: número de machos entre 1 e 2 anos no estágio k
$x_8[k]$: número de machos entre 2 e 3 anos no estágio k

Se formos considerar a negociação do rebanho durante o processo, teremos também a variável de controle. Esta será um vetor $u[k]$, para cada estágio k, com até oito posições, correspondendo à faixa etária e ao sexo que se quer negociar, semelhante ao descrito acima.

$u_1[k]$: número de fêmeas entre 0 e 1 ano a serem negociadas no estágio k
$u_2[k]$: número de fêmeas entre 1 e 2 anos a serem negociadas no estágio k
$u_3[k]$: número de fêmeas entre 2 e 3 anos a serem negociadas no estágio k
$u_4[k]$: número de fêmeas acima de 3 anos com bezerros a serem negociadas no estágio k
$u_5[k]$: número de fêmeas acima de 3 anos sem bezerros a serem negociadas no estágio k
$u_6[k]$: número de machos entre 0 e 1 ano a serem negociados no estágio k
$u_7[k]$: número de machos entre 1 e 2 anos a serem negociados no estágio k
$u_8[k]$: número de machos entre 2 e 3 anos a serem negociados no estágio k

A variável $u_i[k]$ positiva quer dizer compra e negativa quer dizer venda.

É claro que as variáveis do problema deveriam ser números inteiros. Poderíamos resolver este problema como um problema de programação inteira. Mas, em vista da metodologia apresentada, vamos considerar todas as variáveis como números reais (vamos fazer a relaxação linear).

O sistema dinâmico modelado sem a ação de controle é dado pelas equações:

$$\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \\ x_3[k+1] \\ x_4[k+1] \\ x_5[k+1] \\ x_6[k+1] \\ x_7[k+1] \\ x_8[k+1] \end{bmatrix} =$$

$$\begin{bmatrix} 0 & 0 & m_3 f_3/2 & 0 & (m_4-R)f/2 & 0 & 0 & 0 \\ m_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & m_2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & m_3 f_3 & 0 & (m_4-R)f & 0 & 0 & 0 \\ 0 & 0 & m_3(1-f_3) & (m_4-R) & (m_4-R)(1-f) & 0 & 0 & 0 \\ 0 & 0 & m_3 f_3/2 & 0 & (m_4-R)f/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & m_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & m_2 & 0 \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \\ x_3[k] \\ x_4[k] \\ x_5[k] \\ x_6[k] \\ x_7[k] \\ x_8[k] \end{bmatrix}.$$

Ou na forma sintética:

$$\boxed{x[k+1] \;=\; \mathrm{A}x[k].}$$

Podemos colocá-lo também em função do valor inicial:

$$x[k+1] = \mathrm{A}x[k] = \mathrm{A}^{k+1}x[0].$$

Esta equação representa o rebanho definido implicitamente por um parâmetro R que torna o sistema estável. Este parâmetro representa a fração de fêmeas acima de 3 anos que serão descartadas. Esta eliminação de indivíduos permite que, após um intervalo de tempo, a população seja estável.

Alcança-se a estabilidade no sistema quando o maior autovalor da matriz dinâmica A é menor ou igual a um. Encontra-se este estado, variando o parâmetro R num algoritmo de bisseção. Para simplificar a notação, também vamos denotar por A a matriz do sistema dinâmico quando seu maior autovalor for 1 e x^* o autovetor associado ao autovalor unitário. Encontra-se este autovetor com um cálculo simples de álgebra linear (substitua o autovalor $\lambda = 1$ no sistema $(A - \lambda I)x = 0$ e encontre uma base para este autoespaço). Assim, para esta matriz teremos que $x^* = Ax^*$.

O sistema dinâmico com a ação de controle modelado é dado pelas equações:

$$\begin{bmatrix} x_1[k+1] \\ x_2[k+1] \\ x_3[k+1] \\ x_4[k+1] \\ x_5[k+1] \\ x_6[k+1] \\ x_7[k+1] \\ x_8[k+1] \end{bmatrix} =$$

$$\begin{bmatrix} 0 & 0 & m_3 f_3/2 & 0 & (m_4 - R)f/2 & 0 & 0 & 0 \\ m_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & m_2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & m_3 f_3 & 0 & (m_4 - R)f & 0 & 0 & 0 \\ 0 & 0 & m_3(1 - f_3) & (m_4 - R) & (m_4 - R)(1 - f) & 0 & 0 & 0 \\ 0 & 0 & m_3 f_3/2 & 0 & (m_4 - R)f/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & m_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & m_2 & 0 \end{bmatrix} \begin{bmatrix} x_1[k] \\ x_2[k] \\ x_3[k] \\ x_4[k] \\ x_5[k] \\ x_6[k] \\ x_7[k] \\ x_8[k] \end{bmatrix} +$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1[k] \\ u_2[k] \\ u_3[k] \\ u_4[k] \\ u_5[k] \\ u_6[k] \\ u_7[k] \\ u_8[k] \end{bmatrix}.$$

Ou na forma sintética:

$$\boxed{x[k+1] \;=\; \mathrm{A}x[k] + \mathrm{B}u[k].}$$

A matriz B acima é a matriz identidade, indicando que controle será feito em todas as coordenadas de $u[k]$ (podemos negociar qualquer faixa-etária de qualquer sexo do rebanho). Se quisermos fazer o controle em coordenadas específicas, por exemplo, na coordenada m de $u[k]$, basta fazermos apenas a posição $B(m,m) = 1$ e as demais posições de B iguais a zero ($B(i,j) = 0$ quando $(i,j) \neq (m,m)$). Podemos, tendo feito isto, considerar apenas a coluna m de B (e somente a coordenada m de cada $u[k]$).
Por exemplo, se quisermos o controle nas coordenadas 3 (fêmeas entre 2 e 3 anos) e 7 (machos entre 2 e 3 anos) de $u[k]$ basta colocar 1 apenas nas posições $B(3,3)$ e $B(7,7)$ da matriz B e zerar as demais. Podemos, assim, considerar apenas as colunas 3 e 7 de B e as coordenadas 3 e 7 de $u[k]$:

$$\mathrm{B}u[k] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u_3[k] \\ u_7[k] \end{bmatrix}.$$

Funções objetivo:

Função objetivo 1: Considere c um vetor linha de 8 posições que associa aos indivíduos associados à posição x_i o tamanho médio c_i ocupado por cada um. A política ótima 1 consiste em minimizar o total de indivíduos no momento inicial considerando este fator de ocupação c. Esta função objetivo pode ser escrita como:

$$\min_{x[0]} cx[0]$$

Função objetivo 2: Considere c um vetor linha de 8 posições que associa aos indivíduos associados à posição x_i o custo médio de aquisição por cabeça q_i. A política ótima 2 consiste em minimizar custo de aquisição inicial. Esta função objetivo pode ser escrita como:

$$\min_{x[0]} qx[0]$$

Função objetivo 3: Considere o vetor q da função objetivo 2. Acrescente a ele a posição q_9 que corresponde ao preço médio de um indivíduo macho de 3 anos. O rendimento anual (para cada ano k) pode ser modelado pela seguinte função:

$$g[k] = q_9 m_3 x_8[k] + q_5 R m_4 (x_4[k] + x_5[k]).$$

Considere J, um mais a taxa de juros do período. A política ótima 3 consiste em maximizar a diferença entre o que se investiu e o que se obteve de retorno a cada ano, tendo como parâmetro uma mesma taxa de juros. Esta função objetivo pode ser escrita como:

$$\max_{x[0]} \sum_{k=1}^{N} g[k] J^{N-k} - qx[0] J^N.$$

Função objetivo 4: É uma função objetivo para o caso com a ação de controle. A função g representa o custo como função do estágio inicial e cada função h_k representa o custo da variável de controle $u[k]$ no estágio k. A política ótima 4 consiste em minimizar o total de indivíduos a serem comprados (no momento inicial e nos demais). Esta função objetivo pode ser escrita como:

$$\min_{x[0],u[0],\ldots,u[N-1]} g(x[0]) + \sum_{k=0}^{N-1} h_k(u[k]).$$

Função objetivo 5: Vamos considerar a função objetivo 4 como linear. Sejam c e d_k vetores linha. Assim, $g(x[0]) = cx[0]$ e, para cada k, $h_k(u[k]) = d_k u[k]$. A política ótima 5 também consiste em minimizar o total de indivíduos a serem comprados (no momento inicial e nos demais), vendendo o máximo possível. Esta função objetivo pode ser escrita como:

$$\min_{x[0],u[0],\ldots,u[N-1]} cx[0] + \sum_{k=0}^{N-1} d_k u[k].$$

Restrições:

Restrição 1: Os valores das coordenadas do vetor $x[k]$ deverão ser não-negativos em cada instante k. Esta restrição pode ser modelada como:

$$x[k] \geq 0, \quad \text{para cada tempo } k.$$

Restrição 2: Existe uma limitação no espaço da fazenda, que tem uma capacidade L. Cada estado ocupa uma fração c_i do espaço da fazenda. Portanto, esta restrição pode ser modelada como:

$$cx[k] \leq L, \quad \text{para cada tempo } k.$$

Restrição 3: Depois de N anos o rebanho deve ter atingido um estado de estabilidade com uma população que oscile menos que $p\%$, para um p dado. Considerando $||.||$ uma norma qualquer, e ϵ um número positivo pequeno (relacionado com o p), esta restrição pode ser modelada como:

$$||x[N] - x^*|| \leq \epsilon.$$

Restrição 4: Cada variável de controle está limitada entre um valor mínimo e um valor máximo:

$$\min \ \leq u(i)[k] \leq \ \max, \quad \forall i, \ \forall k.$$

Restrição 5: Para minimizar o risco de uma epidemia, a aquisição do rebanho deve acontecer apenas no estágio inicial $k = 0$. Esta restrição não precisa ser considerada, por ser intrínseca ao modelo proposto.

Dados do problema:

Os dados do exemplo desta seção é de um típico rebanho de gado de corte zebu da região de Minas Gerais. Segundo (BDMG, 1970) os parâmetros do modelo são: $m_1 = 0,95$, $m_2 = 0,97$, $m_3 = 0,98$, $m_4 = 0,98$, $f = 0,7$, $f_3 = 0,625$.

O tempo necessário para que o sistema adquira a estabilidade é $N = 7$ anos e o erro admissível após N anos é $\epsilon = 0,01$. Os vetores c e d_k são vetores unitários. Cada variável de controle $u_i[k]$ pode variar entre -10 e 10.

A taxa de descarte anual, calculada pelo método da bisseção é $R = 0,1781$. Com ele, montamos a matriz A do problema, cujo maior autovalor é 1. Precisamos, agora, encontrar o vetor de estabilidade x^*, autovetor de A associado ao autovalor unitário, que será a meta da programação dinâmica.

O tamanho da fazenda é de $L = 1000$ cabeças de gado. Para obedecer à restrição 2, queremos $cx[k] \leq L$, para cada estágio k. Em particular, queremos que no estágio final $cx[N]$ seja bem próximo de L. Se calcularmos x^* de tal modo que $cx^* = L$, esta restrição não precisará ser considerada.

Assim, o vetor x^* (já com a relaxação linear) é:

$$x^* = \begin{bmatrix} 98,4631 \\ 93,5358 \\ 90,7258 \\ 195,0674 \\ 242,1790 \\ 97,5337 \\ 92,6530 \\ 89,8695 \end{bmatrix}.$$

4.3.2 Formulações sem a ação de controle

Apresentamos, a seguir, as três formulações vistas na seção 3.3 com o enfoque neste problema de evolução de um rebanho de gado no tempo.

Aqui, x^* é um autovetor associado ao autovalor unitário da matriz A, logo é um vetor de estabilidade: $x^* = A^k x^*$, $\forall k$. Então, podemos considerar o elipsóide e o politopo no estágio inicial em função de x^*.

Além disso, aqui consideraremos, além da restrição do sistema dinâmico, as demais restrições do problema. Para isto, escreveremos todas as restrições do estado em função do estágio inicial. Isto é simples, pois $x[k] = A^k x[0]$ para qualquer estágio k.

A restrição 1 fica como: $A^k x[0] \geq 0$, para k variando de 0 até N. Porém, basta considerarmos $x[0] \geq 0$ para que o estado nos demais estágios também seja não-negativo.

Para simplificar as formulações, vamos apresentá-las (bem como os resultados) apenas para a função objetivo 1.

FORMULAÇÃO 1:
Para a otimização restrita à pré-imagem da meta de programação, precisamos considerar a restrição 3 como igualdade $x[N] = x^*$:

$$\min_{x[0]} cx[0]$$
$$\text{sujeito a:} \begin{cases} x[0] \geq 0 \\ A^N x[0] = x^*. \end{cases}$$

FORMULAÇÃO 2:
Para a otimização restrita à pré-imagem de uma bola em torno da meta, precisamos considerar a norma euclidiana na restrição 3:

$$\min_{x[0]} cx[0]$$
$$\text{sujeito a:} \begin{cases} x[0] \geq 0 \\ (x[0] - x^*)^t (A^N)^t A^N (x[0] - x^*) \leq \epsilon^2. \end{cases}$$

FORMULAÇÃO 3:
Para a otimização restrita à pré-imagem de um hipercubo em torno da meta, precisamos considerar a norma do máximo na restrição 3:

$$\min_{x[0]} cx[0]$$
$$\text{sujeito a:} \begin{cases} x[0] \geq 0 \\ e_i(A^N x[0] - (x^* + e_i\epsilon)) \leq 0, \quad \text{para cada i} \\ -e_i(A^N x[0] - (x^* - e_i\epsilon)) \geq 0, \quad \text{para cada i.} \end{cases}$$

4.3.3 Comparações entre os métodos e comentários

Vamos apresentar os dados já fazendo o arredondamento do número real encontrado para número inteiro. Para melhor compararmos, colocamos a meta x^* em todas as tabelas.

Os dados para a formulação 1 estão na tabela 4.23 e a evolução do rebanho a cada ano está na figura 4.11.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	Total
$x[0]$	98	94	91	195	242	0	0	0	720
$x[7]$	98	94	91	195	242	97	93	90	1000
x^*	98	94	91	195	242	97	93	90	1000

Tabela 4.23: Vetor de estados para a formulação 1 do problema do gado sem ação de controle.

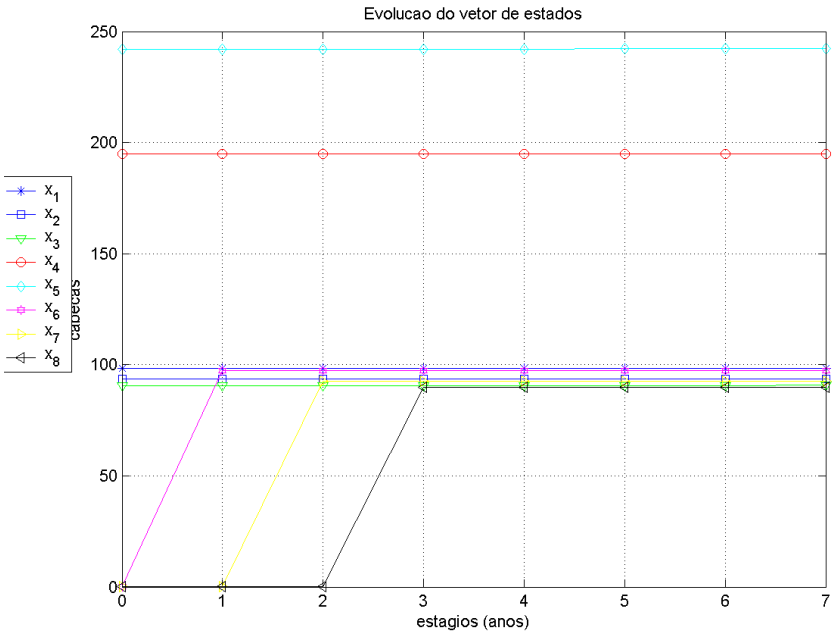

Figura 4.11: Evolução do rebanho para a formulação 1 do problema do gado sem ação de controle.

Os dados para a formulação 2 estão na tabela 4.24 e a evolução do rebanho a cada ano está na figura 4.12.

Os dados para a formulação 3 estão na tabela 4.25 e a evolução do rebanho a cada ano está na figura 4.13.

A tabela 4.26 apresenta uma comparação entre o valor inicial e o tempo gasto (em segundos) nestes três processos:
- (A): resolução por *Quasi-Newton* considerando a formulação 2;
- (B): resolução por *Quasi-Newton* considerando a formulação 3;
- (C): resolução pelo *linprog* considerando a formulação 1;
- (D): resolução pelo *linprog* considerando a formulação 3.

Percebemos, pelo tempo gasto, que o custo computacional da formulação via programação linear é muito menor. A formulação 1 obteve um bom resultado, mas traz a inconveniência de uma restrição de igualdade. A formulação 2 encontrou o menor resultado, mas não é tão eficiente. Como dito no capítulo anterior,

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	Total
$x[0]$	97	92	89	192	245	0	0	0	715
$x[7]$	98	93	90	194	241	97	92	89	994
x^*	98	94	91	195	242	97	93	90	1000

Tabela 4.24: Vetor de estados para a formulação 2 do problema do gado sem ação de controle.

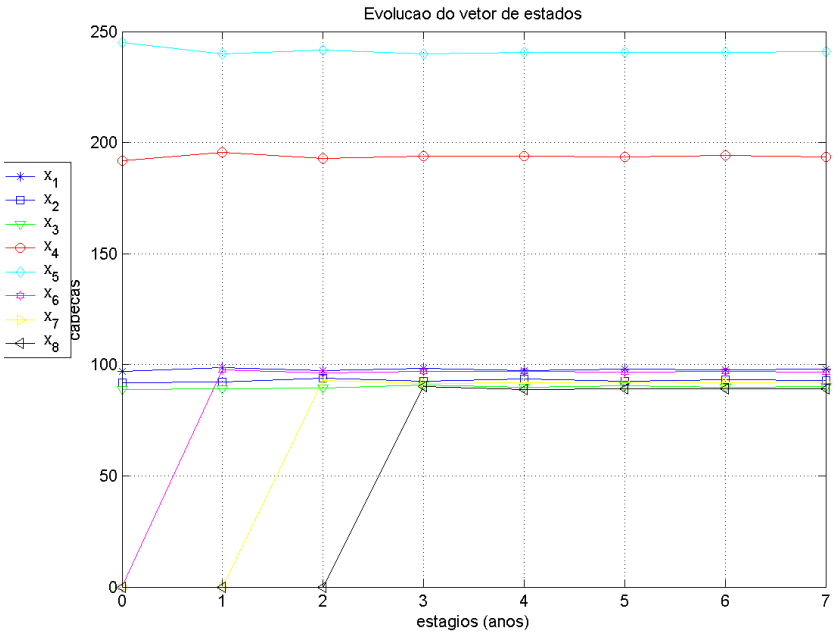

Figura 4.12: Evolução do rebanho para a formulação 2 do problema do gado sem ação de controle.

a formulação 3 têm a vantagem de ser um problema de programação linear e tem uma região factível maior, encontrando resultados melhores que a formulação 1, pois a matriz do sistema dinâmico tem raio espectral menor ou igual a 1.

Um problema similar pode ser proposto considerando que o rebanho masculino deverá ser vendido após dois anos (alteração no S da premissa 3). Na formulação, basta considerar o custo $c_8 = 0$. Os dados para este problema com a formulação 3 estão na tabela 4.27.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	Total
$x[0]$	99	94	90	195	242	0	0	0	719
$x[7]$	98	93	91	195	242	97	92	90	997
x^*	98	94	91	195	242	97	93	90	1000

Tabela 4.25: Vetor de estados para a formulação 3 do problema do gado sem ação de controle.

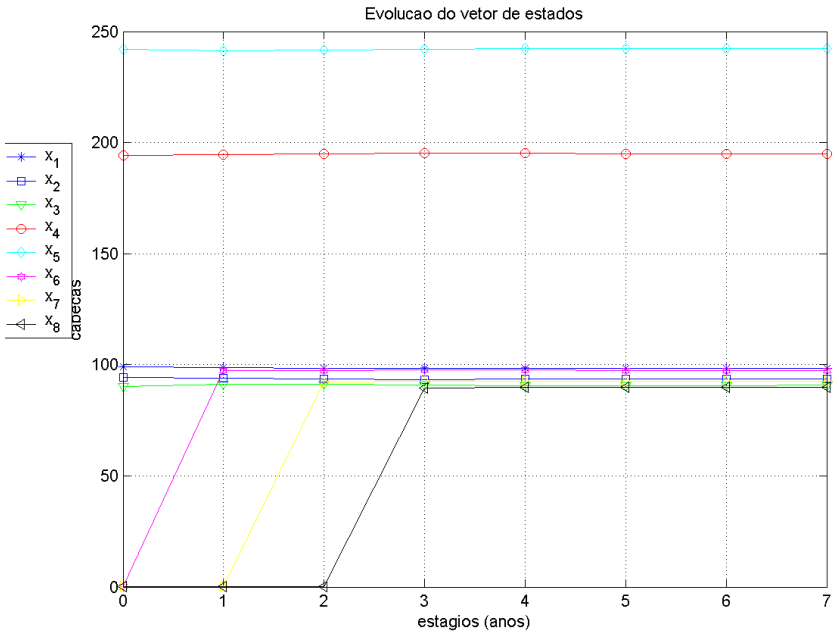

Figura 4.13: Evolução do rebanho para a formulação 3 do problema do gado sem ação de controle.

Vamos agora fazer a análise de sensibilidade do problema com a formulação 3. Vamos considerar pequenas variações nos parâmetros e verificar a interferência nos valores de $x[0]$.

É difícil fazer uma associação entre os parâmetros do modelo e as variáveis duais, como acontece normalmente, pois os parâmetros neste caso não dizem respeito ao termo independente, mas às entradas da matriz. Mesmo assim, é simples

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	total	tempo (s)
A	97	92	89	192	245	0	0	0	715	552,65
B	97	91	89	192	245	9	4	1	728	179,66
C	98	94	91	195	242	0	0	0	720	0,09
D	99	94	90	195	242	0	0	0	719	0,04

Tabela 4.26: Comparação entre o valor inicial e o tempo gasto nestes três processos do problema do gado sem ação de controle.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	Total
$x[0]$	99	94	90	194	241	0	0	0	718
$x[7]$	98	93	91	195	242	97	92	90	997
x^*	98	93	91	195	242	97	93	90	998

Tabela 4.27: Vetor de estados para a formulação 3 com $c_8 = 0$ do problema do gado sem ação de controle.

perceber que ao aumentar em uma unidade a taxa de mortalidade da faixa de 0 a 2 anos, precisaremos de um indivíduo a mais no início. A taxa de mortalidade na faixa acima de 2 anos não altera a solução inicial. A taxa de natalidade para vacas acima de 3 anos segue o mesmo padrão: cada unidade alterada corresponde a variação de um indivíduo na solução inicial. A taxa de natalidade para vacas com 3 anos segue o padrão: cada variação de 5% corresponde a variação de 3 ou 4 indivíduos na solução inicial.

	referência	$m_1 = 0,94$	$m_1 = 0,96$	$m_2 = 0,96$	$m_2 = 0,98$
Variação		-0,01	+0,01	-0,01	+0,01
Função	719	720	718	720	718

Tabela 4.28: Dados da análise de sensibilidade para a formulação 3 do problema do gado sem ação de controle (1).

	referência	$m_3 = 0,97$	$m_3 = 0,99$	$m_4 = 0,97$	$m_4 = 0,99$
Variação		-0,01	+0,01	-0,01	+0,01
Função	719	719	719	719	719

Tabela 4.29: Dados da análise de sensibilidade para a formulação 3 do problema do gado sem ação de controle (2).

	referência	$f = 0,69$	$f = 0,71$	$f_3 = 0,575$	$f_3 = 0,675$
Variação		-0,01	+0,01	-0,05	+0,05
Função	719	720	718	723	716

Tabela 4.30: Dados da análise de sensibilidade para a formulação 3 do problema do gado sem ação de controle (3).

4.3.4 Formulações com a ação de controle

Apresentamos, a seguir, as duas formulações vistas na seção 3.4 com o enfoque deste problema de evolução de um rebanho de gado no tempo.

Também aqui, x^* é um autovetor associado ao autovalor unitário da matriz A, logo é um vetor de estabilidade: $x^* = A^k x^*,\ \forall k$.

Além disso, também aqui consideraremos, além da restrição do sistema dinâmico, as demais restrições do problema. Para isto, escreveremos todas as restrições do estado em função do estágio inicial.

Como estaremos comprando e vendendo em cada estágio, precisaremos, neste caso, considerar toda a restrição 1. Para isto, basta substituir o valor do estado em cada estágio, seguindo a proposição 3.4.1, nas restrições $x[k] \geq 0$ para cada tempo k.

Para simplificar as formulações, vamos apresentá-las (bem como os resultados) apenas para a função objetivo 5.

FORMULAÇÃO 1:
Para a otimização restrita à pré-imagem da meta de programação, precisamos considerar a restrição 3 como igualdade $x[N] = x^*$:

$$\min_{x[0],u[0],\ldots,u[N-1]} cx[0] + \sum_{k=0}^{N-1} d_k u[k]$$

$$\text{sujeito a: } \begin{cases} x[0] \geq 0 \\ A^K x[0] + Bu[K-1] + \sum_{k=1}^{K-1} A^k Bu[N-(k+1)] \geq 0, \ 1 \leq K \leq N \\ A^N x[0] + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] = x^* \\ \min \leq u[k] \leq \max. \end{cases}$$

FORMULAÇÃO 2:
Para a otimização restrita à pré-imagem de um hipercubo em torno da meta, precisamos considerar a norma do máximo na restrição 3:

$$\min_{x[0],u[0],\ldots,u[N-1]} cx[0] + \sum_{k=0}^{N-1} d_k u[k]$$

$$\text{sujeito a: } \begin{cases} x[0] \geq 0 \\ A^K x[0] + Bu[K-1] + \sum_{k=1}^{K-1} A^k Bu[N-(k+1)] \geq 0, \quad 1 \leq K \leq N \\ e_i^t(A^N x[0] - (x^* + e_i\epsilon)) + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] \leq 0, \\ \quad \text{para cada i} \\ -e_i^t(A^N x[0] - (x^* - e_i\epsilon)) + Bu[N-1] + \sum_{k=1}^{N-1} A^k Bu[N-(k+1)] \geq 0, \\ \quad \text{para cada i} \\ \min \leq u[k] \leq \max. \end{cases}$$

4.3.5 Comparações entre os métodos e comentários

Vamos apresentar os dados já fazendo o arredondamento do número real encontrado para número inteiro.

Vamos considerar primeiro, o controle em todas as coordenadas de $u[k]$. Ou seja, a matriz B é a matriz identidade.

Os dados para a formulação 1 estão nas tabelas 4.31 e 4.32 e a evolução do rebanho a cada ano está na figura 4.14.

O resultado da função custo J é 456 cabeças, ou seja, além das 426 cabeças na aquisição inicial devemos comprar mais 30 cabeças ao longo do processo.

Os dados para a formulação 2 estão nas tabelas 4.33 e 4.34 e a evolução do rebanho a cada ano está na figura 4.15.

O resultado da função custo J também é 455 cabeças, ou seja, além das 424 cabeças na aquisição inicial devemos comprar mais 31 cabeças ao longo do pro-

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	Total
$x[0]$	147	167	0	0	112	0	0	0	426
$x[7]$	98	93	91	195	242	98	93	90	1000
x^*	98	94	91	195	242	97	93	90	1000

Tabela 4.31: Vetor de estados para a formulação 1 do problema do gado com ação de controle.

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
$u[0]$	10	10	-10	-10	-10	-10	0	0
$u[1]$	10	10	-10	-10	10	-10	-10	0
$u[2]$	10	10	10	-10	10	-10	-10	-10
$u[3]$	10	10	10	10	10	-10	-10	-10
$u[4]$	-2	10	10	10	10	-3	-10	-10
$u[5]$	0	10	10	10	10	-2	10	-10
$u[6]$	8	10	10	10	10	5	10	10

Tabela 4.32: Seqüência de controles para a formulação 1 do problema do gado com ação de controle.

cesso.

A tabela 4.35 apresenta uma comparação entre o valor inicial e o tempo gasto (em segundos) nos dois processos:
- (A): resolução pelo *linprog* considerando a formulação 1;
- (B): resolução pelo *linprog* considerando a formulação 2.

Os dois processos obtiveram praticamente o mesmo resultado, mostrando que resolvem o problema. Só que a formulação 2 resolveu na metade do tempo da formulação 1. Como as dimensões do problema são pequenas, este dado parece não ter relevância, mas se a dimensão do problema aumentar muito, como já comentado, a restrição de igualdade na formulação 1 pode atrasar bastante sua convergência.

É notável nos dois processos o fato das coordenadas x_3 e x_4 (respectivamente, número de fêmeas entre 2 e 3 anos e número de fêmeas acima de 3 anos) serem nulas no estágio inicial, o que não acontecia no caso sem a ação de controle. Isto proporciona uma redução significativa no custo de aquisição do rebanho (custo

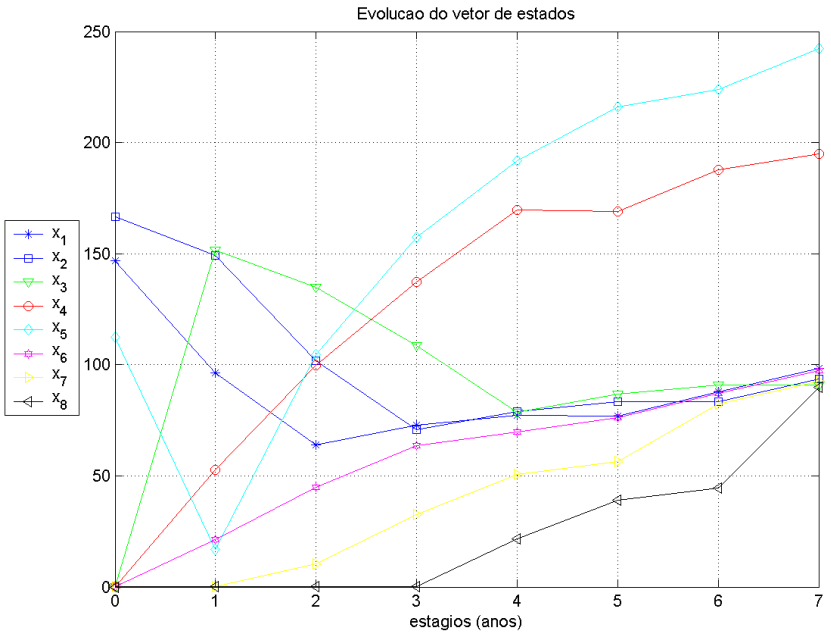

Figura 4.14: Evolução do rebanho para a formulação 1 do problema do gado com ação de controle.

inicial) em relação a ausência da ação de controle.

Nota 4.3.1 *Os valores das funções objetivo das formulações com a ação de controle foram consideravelmente menores que os das formulações sem a ação de controle.*

Vamos apresentar agora o resultado da formulação 2 para o controle feito apenas nas coordenadas 3 (fêmeas entre 2 e 3 anos) e 7 (machos entre 2 e 3 anos) de $u[k]$. Como vimos, para isto, basta colocar 1 apenas nas posições $B(3,3)$ e $B(7,7)$ da matriz B, zerar as demais. Podemos, assim, considerar apenas as colunas 3 e 7 de B e as coordenadas 3 e 7 de $u[k]$.

O resultado da função custo J também é 582 cabeças, só que agora precisamos ter um investimento inicial de 604 cabeças, mas até o final do processo temos o retorno de 19 cabeças.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	Total
$x[0]$	146	166	0	0	112	0	0	0	424
$x[7]$	98	93	91	195	242	97	92	90	998
x^*	98	94	91	195	242	97	93	90	1000

Tabela 4.33: Vetor de estados para a formulação 2 do problema do gado com ação de controle.

	u_1	u_2	u_3	u_4	u_5	u_6	u_7	u_8
$u[0]$	10	10	-10	-10	-10	-10	0	0
$u[1]$	10	10	-10	-10	10	-10	-10	0
$u[2]$	10	10	10	-10	10	-10	-10	-10
$u[3]$	10	10	10	10	10	-10	-10	-10
$u[4]$	-2	10	10	10	10	-3	-10	-10
$u[5]$	0	10	10	10	10	-2	10	-10
$u[6]$	7	10	10	10	10	5	10	10

Tabela 4.34: Seqüência de controles para a formulação 2 do problema do gado com ação de controle.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	total	tempo (s)
A	147	167	0	0	112	0	0	0	426	0,85
B	146	166	0	0	112	0	0	0	424	0,45

Tabela 4.35: Comparação entre o valor inicial e o tempo gasto nestes dois processos do problema do gado com ação de controle.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	Total
$x[0]$	220	243	0	0	141	0	0	0	604
$x[7]$	98	93	91	195	242	97	92	90	998
x^*	98	94	91	195	242	97	93	90	1000

Tabela 4.36: Vetor de estados para a formulação 2 do problema do gado com ação de controle apenas nas coordenadas 3 e 7.

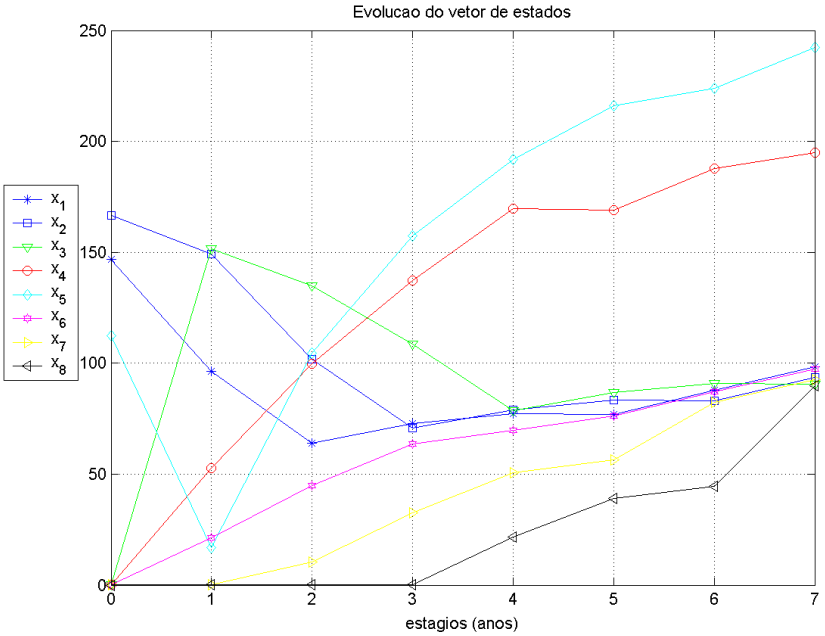

Figura 4.15: Evolução do rebanho para a formulação 2 do problema do gado com ação de controle.

	u_3	u_7
$u[0]$	-10	0
$u[1]$	-10	-10
$u[2]$	10	-10
$u[3]$	10	-10
$u[4]$	10	-10
$u[5]$	7	-2
$u[6]$	2	0

Tabela 4.37: Seqüência de controles para a formulação 2 do problema do gado com ação de controle apenas nas coordenadas 3 e 7.

Capítulo 5

Conclusões e trabalhos futuros

Apresentamos, a seguir, as conclusões deste trabalho de mestrado.

- Propomos neste texto uma uma nova estratégia para a programação dinâmica cujos sistemas forem discretos, lineares, de parâmetros concentrados, invariantes com o tempo: otimizar em apenas um dos estágios. Este fato reduz drasticamente o número de estados e, portanto, o custo computacional da execução.
- Fazemos isto, considerando a pré-imagem da meta de programação, das ações de controle possíveis, bem como das demais restrições do problema em apenas um dos estágios, por exemplo, no estágio inicial.
- Se as variáveis do problema forem números reais, enxergamos o problema dinâmico como um problema estático de programação convexa linear ou não-linear e encontramos o ponto ótimo.
- Se as variáveis do problema forem números inteiros, enxergamos o problema dinâmico como um problema estático de programação inteira. Só que propomos a relaxação linear, para encontrarmos uma cota inferior para o ótimo inteiro.
- Os algoritmos propostos têm complexidade computacional polinomial.
- Se a matriz de transição do sistema dinâmico tiver raio espectral menor ou igual a 1, a pré-imagem no estágio inicial de um conjunto por esta matriz será maior ou igual que a respectiva imagem no estágio final.
- Assim, neste caso será vantagem fazer a relaxação na meta de programação: substituí-la por um conjunto pertencente a uma família invariante ao seu redor.

- A vantagem é que, fazendo a relaxação na meta, aumentaremos a região factível do problema e em muitos casos encontraremos um valor para a função objetivo consideravelmente melhor.
- Além disso, trabalhar com um conjunto em torno da meta, quando aumenta a região factível do problema, tem a vantagem de sair de um possível inconveniente: a restrição de igualdade.
- A região factível no estágio inicial (em cada coordenada) será tão maior quanto menor for o módulo do autovalor correspondente à coordenada.
- Se a função custo for linear, sugerimos usar a família politópica (considerar um hipercubo em torno da meta de programação), a fim de trabalhar com um problema de programação linear.
- As formulações aqui propostas também valem se a aleatoriedade do problema de programação dinâmica estocástica for linear (ou seja $x[k+1] = Ax[k] + Bu[k] + Rw[k]$, com A, B e R matrizes constantes). Neste caso, basta aplicarmos o mesmo raciocínio usado na matriz e nas variáveis de controle para a matriz e para as variáveis da aleatoriedade.

Propomos como trabalhos futuros:

- estudo da relação do ϵ dado e do raio espectral da matriz com a redução na função objetivo.
- estudo mais aprofundado da teoria do controle.
- aplicação destes algoritmos em funções custo não-lineares.
- aplicações destes algoritmos em problemas multi-objetivo.
- estudo de séries temporais.
- estudo mais aprofundado da programação dinâmica estocástica.
- estudo mais aprofundado da programação dinâmica não-linear.
- estudo de outras técnicas de aproximação da solução de um problema de programação dinâmica.

Apêndice A

Parametrização do elipsóide e do politopo

As definições e proposições deste apêndice estão em [Santos (2004)] ou em outros livros de geometria analítica e álgebra linear.

Uma equação quadrática nas variáveis x e y tem a forma

$$ax^2 + bxy + cy^2 + dx + ey + f = 0 \tag{A.1}$$

em que a, b, c, d, e e f são números reais, com a, b e c não simultaneamente nulos.

Esta equação (A.1) representa uma **(seção) cônica**, que são curvas planas obtidas da interseção de um cone circular com um plano. As cônicas não-degeneradas são a elipse, a hipérbole e a parábola e as outras cônicas são chamadas cônicas degeneradas, que incluem um único ponto e um par de retas.

A **elipse** é a curva que se obtém seccionando-se um cone com um plano que não passa pelo seu vértice, não é paralelo a uma reta geratriz (reta que gira em torno do eixo do cone de forma a gerá-lo) e que corta apenas uma das folhas da superfície. Uma maneira equivalente de se definir uma elipse é como um lugar geométrico. Assim, uma elipse é o conjunto dos pontos $P = [x, y]^t$ do plano tais que a soma das distâncias de P a dois pontos fixos F_1 e F_2 (focos) é constante.

Vamos supor que a distância entre os dois focos seja igual a $2c$ e que a soma da distância de cada ponto aos focos seja $2a$. A equação de uma elipse cujos focos são $F_1 = [-c, 0]$ e $F_2 = [c, 0]$ é

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

em que $b = \sqrt{a^2 - c^2}$. A equação de uma elipse cujos focos são $F_1 = [0, -c]$ e $F_2 = [0, c]$ é

$$\frac{x^2}{b^2} + \frac{y^2}{a^2} = 1$$

em que $b = \sqrt{a^2 - c^2}$. Observe que se $F_1 = F_2$ a elipse se reduz à **circunferência** de raio a.

Observe que a equação (A.1) pode ser escrita na forma matricial como:

$$X^t AX + KX + f = 0, \tag{A.2}$$

em que:

$$X = \begin{bmatrix} x \\ y \end{bmatrix}, \quad A = \begin{bmatrix} a & b/2 \\ b/2 & c \end{bmatrix} \quad \text{e} \quad K = \begin{bmatrix} d & e \end{bmatrix}.$$

Fazendo uma mudança de variáveis, conseguimos colocar a equação (A.2) na forma mais simples:

$$X^t BX = d^2. \tag{A.3}$$

Além disso, se esta for a equação de uma elipse, todos os autovalores da matriz B serão positivos e o número positivo d pode ser chamado de raio da elipse.

Vamos, agora, para o $\mathbb{R}^3$. Definimos como **quádricas** as superfícies que podem ser representadas pelas equações quadráticas nas variáveis x, y e z, ou seja, da forma:

$$ax^2 + by^2 + cz^2 + dxy + exz + fyz + gx + hy + iz + j = 0 \tag{A.4}$$

em que a, b, c, d, e, f, g, h, i, $j \in \mathbb{R}$, com a, b, c, d, e, f não simultaneamente nulos.

Um **elipsóide** em $\mathbb{R}^3$ é o conjunto de pontos que, em algum sistema de coordenadas, satisfazem a equação:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1,$$

em que: a, b e c são números reais positivos. Observe que a interseção da elipse com os planos $z = k$ com $\|k\| < c$, $y = k$ com $\|k\| < b$ e $x = k$ com $\|k\| < a$ é uma elipse. Observe também que se $a = b = c$, o elipsóide é uma **bola** ou esfera de raio a.

Observe que podemos também escrever a equação de uma quádrica na forma matricial, como:

$$X^t AX + KX + j = 0, \tag{A.5}$$

em que:

$$X = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad A = \begin{bmatrix} a & d/2 & e/2 \\ d/2 & b & f/2 \\ e/2 & f/2 & c \end{bmatrix} \quad \text{e} \quad K = \begin{bmatrix} g & h & i \end{bmatrix}.$$

Também fazendo uma mudança de variáveis, conseguimos colocar a equação (A.5) na forma mais simples:

$$X^t B X = d^2. \tag{A.6}$$

Além disso, se esta for a equação de um elipsóide, todos os autovalores da matriz B serão positivos e o número positivo d pode ser chamado de raio do elipsóide.

Podemos generalizar a definição de elipsóide para o espaço $\mathbb{R}^n$. A fim de trabalharmos com uma equação que independa do ambiente, vamos considerar a equação na forma matricial. Observe que as equações (A.3) e (A.6) têm a mesma forma, o que muda é a sua dimensão. Portanto, definimos um elipsóide em $\mathbb{R}^n$ como o conjunto dos pontos $X \in \mathbb{R}^n$ que satisfaçam a equação:

$$X^t B X = d^2, \tag{A.7}$$

em que B seja uma matriz simétrica de dimensão n com todos os autovalores positivos e d seja um número real positivo, chamado de raio do elipsóide.

Podemos também transladar o elipsóide de modo que ele fique centrado em qualquer ponto $X^* \in \mathbb{R}^n$. Este novo elipsóide terá como equação:

$$(X - X^*)^t B (X - X^*) = d^2, \tag{A.8}$$

em que B seja uma matriz simétrica de dimensão n com todos os autovalores positivos e d seja um número real positivo.

Esta é a parametrização que usamos neste trabalho. Se a matriz simétrica B tiver autovalores não-negativos, a equação (A.8) corresponderá a equação de um elipsóide em $\mathbb{R}^n$, centrado em X^* com raio d, e o chamaremos ao longo do texto de **elipsóide degenerado** ou apenas de elipsóide. Como um sub-caso, quando a matriz B tiver todos os autovalores positivos, o chamaremos neste texto de **elipsóide não-degenerado**. Se a matriz B for a matriz identidade, esta será a equação de uma **bola** no $\mathbb{R}^n$ de raio d, centrada em X^*.

Agora, vamos justificar a parametrização do hipercubo e do politopo. Procuramos uma parametrização que fosse mais computacional. Um politopo em $\mathbb{R}^n$ é a interseção de semi-espaços limitados por hiperplanos. Portanto, o que

precisamos saber é como parametrizar um hiperplano.

No plano, a equação de uma reta é determinada se forem dados sua inclinação e um de seus pontos. No espaço, a inclinação de um plano é dada por um vetor perpendicular a ele e a equação de um plano é determinada se são dados um vetor perpendicular a ele e um de seus pontos. Do mesmo modo, a equação de um hiperplano em $\mathbb{R}^n$ é determinada se são dados um vetor perpendicular a ele e um de seus pontos.

Seja $N = [N_1, \ldots, N_n)]^t$, um vetor no $\mathbb{R}^n$ perpendicular ao hiperplano, chamado de vetor normal do hiperplano, e $P = [P_1, \ldots, P_n]^t$ um ponto pertencente ao hiperplano. A equação deste hiperplano é:

$$N_1 x_1 + N_2 x_2 + \ldots + N_n x_n + d = 0 \tag{A.9}$$

em que $d = -(N_1 P_1 + N_2 P_2 + \ldots + N_n P_n)$. A equação (A.9) é chamada de equação geral do hiperplano.

Repare que podemos reescrever a equação (A.9) como:

$$N^t(X - P) = 0 \tag{A.10}$$

em que $X = [x_1, \ldots x_n]^t \in \mathbb{R}^n$. Esta é a parametrização do hiperplano que usamos neste trabalho.

Este hiperplano divide o espaço em dois semi-espaços. As equações dos dois semi-espaços limitados por hiperplanos da equação (A.10) são:

$$N^t(X - P) \leq 0 \quad \text{e} \quad N^t(X - P) \geq 0. \tag{A.11}$$

Um hiperparalelepípedo em $\mathbb{R}^n$ é um politopo cujos 2^n hiperplanos de sua face são ortogonais dois-a-dois e o hipercubo é um hiperparalelepípedo com todos lados iguais. Podemos pensar também no hipercubo como o conjunto dos pontos $X \in \mathbb{R}^n$ que eqüidistam (distância definida a partir da norma do máximo) de um mesmo ponto X^*. Sua equação seria:

$$||X - X^*||_\infty \leq \epsilon,$$

que chamamos de hipercubo de raio ϵ com centro em X^*.

Considere a base canônica do R^n. Fixe uma coordenada i e considere as duas faces do hipercubo correspondentes a esta coordenada. O vetor normal de cada face do hipercubo é o vetor canônico e_i ou $-e_i$. Além disso, o ponto $X^* + e_i\epsilon$

pertence à face cujo vetor normal é e_i e o ponto $X^* - e_i\epsilon$ pertence à face cujo vetor normal é $-e_i$.

Portanto, a partir da equação (A.11), podemos parametrizar este hipercubo como a interseção dos semi-espaços:

$$\begin{cases} e_i^t(X - (X^* + e_i\epsilon)) \leq 0, & \text{para cada } i \\ -e_i^t(X - (X^* - e_i\epsilon)) \geq 0, & \text{para cada } i. \end{cases}$$

Esta é a parametrização do hipercubo que usamos neste trabalho.

Bibliografia

BAZARAA, M. S., *Programación Lineal y Flujo en Redes.* Limusa Noriega Editores, 1999.

BDMG, *Programa de desenvolvimento para a criação de gado de corte em Minas Gerais.* Relatório Técnico, Banco de Desenvolvimento de Minas Gerais (BDMG), 1970.

BELLMAN, R., *Dynamic Programming.* Princeton University Press, 1957.

BERTSEKAS, D. P., *Dynamic Programming and Optimal Control, volume 1.* Athena Scientific, 1995.

CARVALHO, J. L., *Production investiment and expectations: a study of the United States cattle industry.* Ph.D. thesis, Chicago University, Illinois, USA, 1972.

CHEN, C., *Linear System Theory and Design.* Oxford University Press, 1999.

FARIAS, D. P. de & VAN ROY, B., *The linear programming approach to approximate dynamic programming.* Operations Research, volume 51, number 6, pages 850-865, 2003.

GARFINKEL, R. S. & NEMHAUSER, G. L., *Integer Programming.* John Wiley & Sons, Inc, 1972.

GOLDBARG, M. C. & LUNA, H. P. L., *Otimização Combinatória e Programação Linear.* Editora Campus, 2000.

GONZAGA, C. C., *17º Colóquio Brasileiro de Matemática - Algoritmos de Pontos Interiores para Programação Linear.* Instituto de Matemática Pura e Aplicada do CNPq, 1990.

MASCÓ, R. 0., *Introducción da la Programacion Dinamica (apuntes de clase).* Escuela de Ingeniería Industrial, Facultad de Ciencias Exactas, Ingeniería y Agrimensura, Universidad Nacional de Rosario, Argentina, 2001.

MASCOLO, J. L., *Um estudo econômico da criação de gado de corte no Brasil.* Tese de doutorado, Fundação Getúlio Vargas, 1980.

PHILLIPS, C. L. & NAGLE, H. T., *Digital Control System Analysis and Designer - third edition.* Prentice Hall International Editions, 1995.

SANTOS, R. J. S., *Um Curso de Geometria Analítica e Álgebra Linear.* Imprensa Universitária da UFMG, 2004.

TAKAHASHI, R. H. C., *Notas de aula do curso Otimização Escalar e Vetorial.* Departamento de Matemática, Instituto de Ciências Exatas, Universidade Federal de Minas Gerais, 2005, disponível em `http://www.mat.ufmg.br/ taka`.

TAKAHASHI, R. H. C., CALDEIRA, C. P. & PERES, P. L. D., *A linear dynamic system approach for cattle herd optimal shaping.* International Journal of Systems Science, volume 28, number 9, pages 913-918, 1997.