UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Facultad de Ciencias

"APLICACION DE UN PAQUETE GENERALIZADO DE PROGRAMACION DINAMICA"

T E S I S

Que para obtener el título de

A C T U A R I O

p r e s e n t a

FERNANDO RAUL VIDALES HERNANDEZ

México, D. F. 1990

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO

INTRODUCCION

Como es bien conocido, los métodos de la investigación de operaciones son útiles para obtener políticas óptimas o decisiones adecuadas, que mejoren el funcionamiento de un sistema en general. A la investigación de operaciones la podríamos definir como: la aplicación del método científico a la solución de problemas relacionados con el control de organizaciones o sistemas a fin de que se produzcan soluciones que mejor satisfagan los objetivos fijados. Para hacer más clara la esencia de la investigación de operaciones podemos describir sus características de la siguiente manera: El enfoque principal de la investigación de operaciones es en la toma de decisiones y la planeación; Sus áreas de aplicación son a problemas que tienen que ver con la conducción y coordinación de operaciones y actividades de cualquier organización; En el enfoque metodológico, el proceso de solución comienza con la observación cuidadosa y el planteamiento del problema, para poder construir un modelo que trate de abstraer la esencia del problema; El objetivo es encontrar una solución óptima al problema que estemos tratando.

Antes de aplicar algún método de solución, para algún problema en particular, necesitamos disponer de un modelo. En este sentido el modelar se convierte en una fase importante de la investigación de operaciones. El construir un modelo, ayuda a colocar los aspectos complejos e inciertos de un problema en una estructura lógica que es adecuada para el análisis formal. El modelo especifica las alternativas de decisión y sus consecuencias anticipadas para todos los eventos posibles que puedan ocurrir, indica los datos importantes para analizar las alternativas y conduce a conclusiones generales que tienen sentido

en el contexto del problema. En resumen, un modelo es un vehículo para lograr una visión bien estructurada de la realidad.

Los modelos se pueden clasificar como icónicos, análogos o simbólicos. Los modelos icónicos son la representación física, a escala, de un sistema real. Por ejemplo, un barco de juguete es un modelo icónico de uno real. Los modelos análogos esencialmente requieren la sustitución de una propiedad por otra con el fin de permitir la manipulación del modelo y, después de resolver el problema, la solución se reinterpreta de acuerdo al sistema original. Por ejemplo, un modelo de redes eléctricas puede utilizarse como un modelo análogo para el estudio de flujos en un sistema de transporte. Los modelos simbólicos o matemáticos utilizan un conjunto de símbolos matemáticos y funciones, para representar las variables de decisión y sus relaciones, para describir el comportamiento del sistema. Este tipo de modelos caen en el campo de la llamada Programación Matemática.

El desarrollo de la computación ha permitido, también, el desarrollo de los modelos de simulación y los métodos heurísticos. Los modelos de simulación son programas de computadora que hacen una replica o simulan el comportamiento del sistema real. Generalmente se utilizan para simular sistemas donde tenemos incertidumbre y que, además, son lo suficientemente complejos para poder hacer un modelo matemático de ellos. Cuando el esfuerzo computacional de algún tipo modelo es muy elevado, se disponen de los métodos heurísticos, los cuales, esencialmente emplean reglas intuitivas o lógicas, o bien, utilizan ciertas guías para tratar de generar nuevas estrategias que se traduzcan en soluciones mejoradas para el sistema real. Este tipo de métodos, por su naturaleza, no pretenden obtener soluciones óptimas y son muy utilizados en Inteligencia Artificial.

El gran potencial de aplicación de la investigación de operaciones, ha motivado un creciente desarrollo teórico de esta disciplina, sobre todo en lo que respecta a la programación matemática. El problema general de la programación matemática consiste en optimizar una función (objetivo) f sujeta a restricciones que definen un conjunto de factibilidad S, a

saber:

$$Max \ (o \ Min) \ f(x_1, x_2, \ldots, x_n)$$

$$s.a.$$

$$(x_1, x_2, \ldots, x_n) \in S$$

La teoría de la programación matemática se ha diferenciado por áreas de estudio que proporcionan métodos adecuados de solución al problema anterior, según sean la forma de la función objetivo f y la del conjunto de restricciones que determinan a S, por ejemplo: Cuando la función f es lineal y S está determinado por un conjunto de restricciones lineales, tenemos un problema de programación lineal. Un método de solución, para este tipo de problemas, fué desarrollado por George Dantzig, en 1947, creando el llamado método Simplex, que es el más conocido y desarrollado de la programación matemática por su sencillez y porque, en muchas aplicaciones, un modelo lineal es suficientemente satisfactorio. Cuando la función f es no-lineal o el conjunto S está determinado por restricciones no-lineales tenemos un problema de la programación no lineal, donde Kuhn y Tucker obtuvieron importantes resultados que permitieron el uso práctico de esta técnica, como fué dar condiciones suficientes para la existencia de un óptimo. En el caso, en que la función objetivo y las restricciones sean lineales pero se requieran soluciones enteras, se tiene un problema de programación lineal entera, donde Gomory fué el principal aportador, y donde se han desarrollado diferentes algoritmos de cortes (los más de tipo heurístico) para resolver problemas de este tipo. Estas tres áreas cubren un amplio espectro de los problemas que podemos encontrarnos en una organización o sistema.

LA PROGRAMACION DINAMICA

Otro importante método de solución es la programación dinámica. Cuyos fundamentos formales fueron estudiados ampliamente por Richard Bellman, de quién recibe su nombre. Bellman fué el primero en apreciar el amplio rango de aplicabilidad de esta forma de optimización y publicó sus

primeros trabajos al respecto en 1957. Desde entonces han aparecido otros escritos que han servido para cimentar y extender esos trabajos iniciales y colocarlos en un amplio contexto.

La programación dinámica es una técnica poderosa de optimización útil para resolver una gran variedad de problemas, que en algunos casos no pueden ser solucionados por otras técnicas, o bien, porque resulta ser el método más adecuado de solución como en los problemas de control. Sin embargo en la práctica la programación dinámica no es muy popular, debido a diferentes causas, de las cuales mencionaremos algunas de las más importantes a continuación:

a) El problema de la dimensionalidad de la programación dinámica. Donde el esfuerzo computacional puede llegar a ser muy alto en comparación con otras técnicas.

b) La escasez de paquetes de computo de uso generalizado. Software para aplicaciones generales abundan en las áreas de programación lineal (principalmente), no lineal y redes, en comparación con los paquetes disponibles de programción dinámica.

c) Ante la escasez de paqueteria generalizada, los usuarios (por ejemplo, investigadores) se enfrentan con la dificultad de desarrollar un programa especial para cada problema en particular. Además esta tediosa aplicación de la programación dinámica requiere de una comprensión completa de ésta y de la habilidad necesaria para desarrollar un código eficiente.

Los inconvenientes anteriores si no son todos, sí son los más relevantes, pero no por éstos hay que descalificar a la programación dinámica, por ejemplo el problema de la dimensionalidad, considerado su mayor desventaja, en muchas ocasiones puede ser superado, por lo que vale la pena tomar en cuenta a la programación dinámica en la solución de problemas. También es factible generar un algoritmo general de esta técnica y, además, este algoritmo es particularmente atractivo para ser implementado en microcomputadoras como se verá a continuación.

Varios proyectos de diseño y problemas operacionales pueden ser vistos como un proceso secuencial de decisión por etapas. La programación dinámica es ideal para resolver problemas de multietapas donde el esfuerzo computacional tiene aproximadamente un incremento lineal con respecto al número de etapas. Esto es ideal para microcomputadoras donde el tiempo de proceso es importante.

Cuando las variables de decisión son discretas y enteras, como ocurre frecuentemente en problemas de planeación y diseño, entonces tenemos un posible campo de utilización de la programación dinámica, ya que esta es aplicable a un amplio rango de problemas de programación entera. Esto constituye una ventaja para procesos en microcomputadoras, puesto que no se requiere un alto grado de precisión, en comparación con la programación lineal en donde el grado de precisión requerido es elevado, para la inversión de matrices y cálculos con números reales.

En la programación dinámica la relación funcional entre la función objetivo y las restricciones no tienen condiciones, pues, éstas pueden ser no lineales, no convexas y a veces hasta discontinuas. Las áreas de ingeniería y dirección, en particular, pueden ser áreas de aplicación de la programación dinámica, ya que, se requieren técnicas de optimización que puedan incorporar sistemas tan realistas como sea posible, más que forzar a simplificaciones o idealizaciones del sistema de interés.

Varios sistemas realistas son altamente regulados, lo que significa un gran número de restricciones y un número reducido de posibles políticas de decisión, o sea, un número pequeño de combinaciones de decisiones discretas factibles. En este tipo de casos la programación dinámica constituye una ventaja con respecto a otros métodos dado que el tiempo de proceso es reducido.

A diferencia de otras técnicas de la programación matemática, la programación dinámica produce políticas de decisión "retroalimentadas", lo cual permite flexibilidad en el seguimiento (control) de complejos sistemas, donde la decisión óptima factible de una etapa está condicionada a la situación (estado) del sistema en ese momento. Este

tipo de políticas flexibles son atractivas, particularmente, para resolver problemas estocásticos y de múltiples objetivos.

Un algoritmo de programación dinámica es ideal para ser implementado en microcomputadoras. Pues como habíamos mencionado, una desventaja de la programación dinámica, y quizá la mayor, es el problema del dimensionamiento, donde enormes requerimientos de memoria principal pueden ser requeridos en problemas con varias variables de estado. Sin embargo, hasta la fecha la mayoría de los trabajos de la programación dinámica han sido implementados en los principales sistemas de computo, con una alta precisión (hasta palabras de sesenta bits) pero con limitaciones en la memoria principal (e.g., menos de 300 k bytes). Por otra parte las microcomputadoras pueden tener palabras de 16 bits, lo cual es sufuciente precisión para la mayoría de aplicaciones de la programación dinámica, pero una capacidad mayor de direccionamiento en memoria principal, que en algunos casos puede ser hasta de 3 Mb. Con microprocesadores tales como el Intel 80286 existe un potencial de direccionamiento de 32 Mb de memoria virtual.

Con respecto a las aplicaciones en situaciones reales, la programación dinámica se encuentra más limitada que otras áreas de optimización de la investigación de operaciones, ya que el tamaño de los problemas reales frecuentemente es muy grande para ser atacados por esta técnica, debido al gran esfuerzo computacional requerido. Sin embargo lo anterior no quiere decir que la programación dinámica deba ser vista como una área de interés teórico exclusivamente, puesto que, por un lado, existen problemas reales que se han modelado con programación dinámica, por ejemplo podemos citar un análisis sobre la reserva petrolera de E.E.U.U. y la reducción de importaciones del mismo, hecho por Chao y Manne [1]; el estudio realizado por Kilmer, Spreen y Tilley sobre la localización óptima de plantas empacadoras de cítricos para el este de Florida [2]; también, un trabajo de planeación para la generación de energía eléctrica, elaborado por Evans y Morin [3]. Todos estos trabajos se han modelado con programación dinámica. Por otro lado, para aumentar el rango de aplicación de la programación dinámica, se estan desarrollando métodos apropiados para bajar el grado de complejidad computacional de

esta técnica, de los cuales podemos mencionar a la programación dinámica incremental, desarrollada por Larson [4]; también hay un método híbrido de programación dinámica con ramificación y acotamiento, el cual ha sido propuesto por Morin y Marsten [5] (el trabajo sobre la generación de energía eléctrica, de la referencia [3], está diseñado para utilizar este método híbrido); por último, podemos mencionar un método heurístico para aplicarse a la programación dinámica multi- objetivo, propuesto por Kornbluth [6]. Por lo anterior podemos concluir que la programación dinámica puede ser útil en la práctica, justificándose así su estudio dentro de la investigación de operaciones.

Antes de definir el objetivo de este trabajo, como experiencia personal podría decir que la motivación inicial que me condujo a desarrollar este trabajo fué darle un poco de publicidad a la programación dinámica (lo que es un objetivo en sí), pués, observé que para mis compañeros este método no era de su agrado, en comparación con otros de la investigación de operaciones, debido a la dificultad de modelar los problemas y en consecuencia la dificultad de encontrar la función recursiva.

El objetivo de esta tesis es: mostrar la estructura y componentes de la programación dinámica en su forma determinista y estocástica presentar el paquete CSUDP (esto es un paquete generalizado de programación dinámica) y su aplicación. El trabajo se desarrolla de la manera siguiente: en el capítulo 1 se presentan los principios básicos de la programación dinámica, explicados por medio del modelo determinista. En el capítulo 2 se describe la programación dinámica estocástica y su aplicación en los procesos de Markov con ganancias. En el capítulo 3 se presenta el paquete CSUDP, describiendo sus características y funcionamiento, así como opciones de uso. Por último en el capítulo 4 se dan ejemplos de aplicaciones de la programación dinámica; estos ejemplos estan resueltos a "mano" y por medio del paquete (CSUDP) con el fin de que las soluciones también sirvan como ejemplo a los otros capítulos.

CAPITULO 1

LA PROGRAMACION DINAMICA DETERMINISTICA

Los fenómenos que suceden en la naturaleza en un sentido general, están condicionados por causas o leyes que el hombre siempre a pretendido conocer o descubrir a lo largo de la historia. De esta manera, dependiendo del grado de conocimiento que se tenga acerca de un fenómeno en particular, éste se puede conocer con exactitud o no. En este sentido los fenómenos se clasifican en deterministicos o estocásticos. Los problemas de optimización que nos interesan se pueden clasificar también de estas dos formas.

En este capitulo se describen las características y los componentes básicos de la programación dinámica así como algunos aspectos metodológicos que subyacen en dicha técnica. El capitulo se desarrolla como sigue: En la primera sección se describen los conceptos básicos de estado, etapa y decisión entre otros, mientras que en la segunda sección se analizan la relación de recurrencia así como algunos resultados útiles. En la tercera sección se analiza el método de la programación dinámica hacia adelante y hacia atrás. Finalmente, en la cuarta sección se describen los aspectos metodológicos para la formulación de problemas.

1.1 COMPONENTES BASICOS

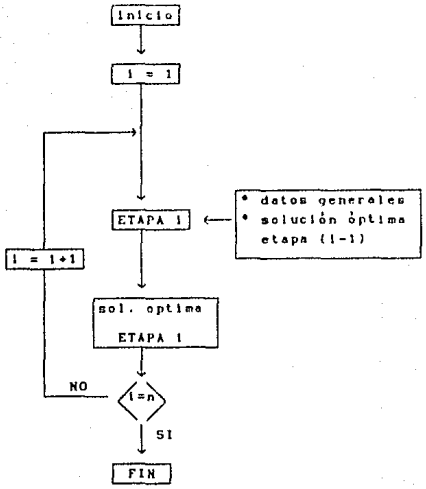
La programación dinámica es un método de optimización que puede resolver una amplia gama de problemas de manera eficiente y satisfactoria. El procedimiento general consiste en dividir el problema original en n subproblemas que se resuelven secuencialmente para obtener una solución óptima que se compone de n acciones o decisiones (una por etapa) denominada política de decisión. Como es de esperarse, el problema original debe tener una cierta estructura matemática.

Dadas las características esenciales del método de la programación dinámica es necesario visualizar el problema original como un sistema secuencial, en el que hay que precisar conceptos tales como etapas, estados, decisiones y la relación de recurrencia. Todo esto nos proporciona las bases para aplicar un algoritmo de programación dinámica.

ETAPA

En un problema dinámico, o susceptible de modelarse con programación dinámica, la estrategia a seguir es dividir el problema principal en n "subproblemas" más simples que llamaremos **etapas**. Estas etapas se resuelven en un orden secuencial y , por lo tanto, para poder determinar la solución óptima de una etapa se requiere disponer de la solución óptima de la etapa anterior. Por ejemplo, en el problema de asignación de recursos financieros a cuatro proyectos (ver ejemplo 3, apartado 4.3), donde el objetivo es en reducir los tiempos de construcción de los cuatro proyectos, el problema principal se descompone en cuatro subproblemas más simples, y que consiste en resolver primero para un sólo proyecto después para dos proyectos (el anterior y uno más) y así sucesivamente. De esta manera estar en la etapa i significa tomar en cuenta los primeros i proyectos.

En la figura 1 se muestra, de manera general, las partes principales de un algoritmo de programación dinámica.



DECISIONES OPTIMAS SECUENCIALES

figura 1

ESTADO

En los problemas de la vida real es frecuente prever las distintas situaciones con las que pudieramos encontrarnos antes de considerar todas las alternativas de acción. Lo anterior se refleja en los problemas planteados con programación dinámica. Especificamente, es posible tener varias situaciones distintas, para cada etapa i, que llamaremos **estados** posibles de la etapa i. Tales estados reflejan cierto tipo de información sobre el problema. Los estados deben ser identificados desde el principio para ser analizados exhaustivamente y se puede decir que su identificación es parte del arte de modelar, el problema, usando programación dinámica.

En general se denotará a S_i como el conjunto de todos los estados posibles de la etapa i, específicamente

$$S_i = \{ x \ / \ x \text{ es un estado posible de la etapa } i \}$$

donde S_i puede ser discreto o continuo. También se tomará a x como la variable de estado de la etapa actual o que estemos trabajando, o bien cuando queramos ser precisos diremos que x_i es la variable de estado de la etapa i. En el problema de asignación de recursos financieros, estar en el estado x_i significa que se disponen de x_i miles de pesos para repartir entre los primeros i proyectos. Dado que los recursos se asignan en partidas de $5,000.00, tenemos que

$$S_i = \{0,\ 5000, 10000, \ldots, 35000\}\ ; \qquad i = 1..3$$

$$S_4 = \{35000\}$$

DECISION

En la programación dinámica para resolver la etapa i se necesita disponer de la solución óptima de la etapa anterior i-1 y esto nos indica una relación entre los estados posibles de cada etapa. Esta relación se da por medio de acciones o decisiones, sobre las cuales se realiza el proceso de optimización ,es decir, para cada estado existe un conjunto de alternativas de decisión que pueden ser adoptadas a partir de ese estado.

Para determinar la decisión óptima de la etapa i la programación dinámica identifica a cuales estados de la etapa i-1 es posible llegar desde el estado x_i y elegir la acción más adecuada que optimice los "recursos" para las primeras i etapas. Por lo anterior podemos concluir que la solución óptima de la etapa i consiste en encontrar una política óptima de decisión (para las primeras i etapas) para cada uno de los estados posibles de la etapa i. Esto es factible sólo si se cumple el principio de optimalidad de Bellman que enunciaremos más adelante.

Supongamos que estamos considerando una decisión u_i y un estado x_i

fijos, de la etapa i, si la acción u_i puede ser adoptada (según el contexto del problema) desde el estado x_i y, además, la consecuencia de esa acción es encontrarse en un estado posible de la etapa anterior i-1, entonces se dirá que dicha acción u_i es una **decisión factible** para el estado x_i o simplemente ,de manera más general, una decisión factible para la etapa i. Definimos a

$$D(x) = \{ u \ / \ u \text{ es una decisión factible para el estado } x \}$$

y

$$D_i = \bigcup_{x \in S_i} D(x)$$

o sea, D_i es el conjunto de decisiones factibles de la etapa i. En general diremos que u es la variable de decisión de la etapa que estemos tratando, o bien, cuando queramos precisar definiremos a u_i como la variable de decisión de la etapa i. También diremos que u_i^* es una decisión óptima para la etapa i. Por ejemplo, en el problema de asignación de recursos financieros, en la etapa 2, si x = 10,000. el conjunto de decisiones factibles para ese estado será

$$D(x) = \{0,\ 5000, 10000\}$$

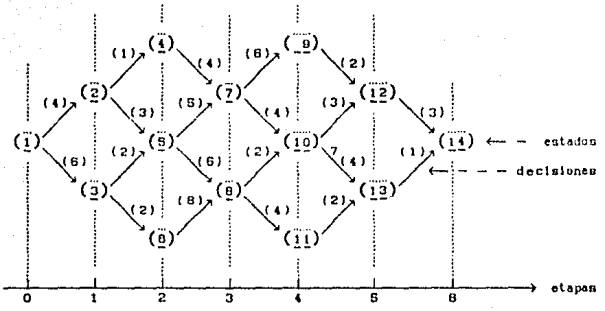
y el conjunto de decisiones factibles para cualquier etapa i es

$$D_i = \{0,\ 5000, \ldots, 35000\} \qquad i = 1..4.$$

Para dar una idea gráfica de como se relacionan los tres conceptos anteriores supongamos que tenemos la red de la figura 2, entonces los nodos corresponden a los estados, los arcos a las decisiones y una etapa se compone del conjunto de nodos, tales que para cada estado posible x, de esa etapa, todos los predecesores de x pertenecen a la etapa anterior. En esta representación el problema usual de la programación dinámica consistirá en encontrar el camino más corto entre el nodo 1 y el nodo 14, si el problema consiste en minimizar, o bien, el camino más largo si queremos maximizar. La distancia entre los nodos viene dada por los números entre paréntesis. Por ejemplo, para la etapa 2 tenemos que

$$S_2 = \{4,\ 5,\ 6\};\ D(4) = \{(2,4)\};\ D(5) = \{(2,5),\ (3,5)\};\ D(6) = \{(3,6)\}$$

$$\text{y} \quad D_2 = \{(2,4),\ (2,5),\ (3,5),\ (3,6)\}$$



RELACION ENTRE DECISIONES Y ESTADOS

figura 2

TRANSICION ENTRE ESTADOS

Hemos visto que la esencia de la programación dinámica es resolver los problemas por etapas donde, por medio de las acciones factibles de cada etapa, existe una relación entre los estados. En otras palabras, conociendo el estado actual y la decisión adoptada, podemos identificar el estado de procedencia. Esta relación la podemos expresar con la llamada función de transición (t):

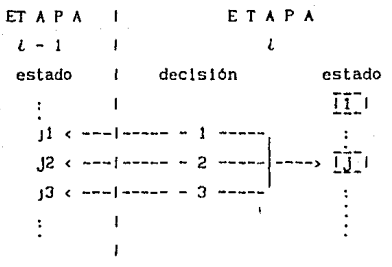
$$x_{i-1} = t_i(x_i, u_i) \qquad (1.1)$$

donde: $x_{i-1} \in S_{i-1}$; $x_i \in S_i$; $u_i \in D_i$; $t_i : U_i \longrightarrow S_{i-1}$;

El conjunto $U_i \subset S_i \times D_i$, se define de la siguiente manera:

$$U_i = \{(x,u) \mid \text{para cada } x,\ x \in S_i, \text{ se tiene que } u \in D(x)\}$$

Gráficamente podemos representar la relación entre estados como se muestra en la figura 3.



TRANSICION ENTRE ESTADOS

figura 3

Por ejemplo, en la red de la figura 2, tenemos que

$$t_i(x_i,u_i) = t_i(k,(j,k)) = j = x_{i-1} \; ; \qquad i = 1..6$$

donde el dominio de t_i es

$$U_i = \{(k,(j,k) \mid k \in S_i \text{ y } (j,k) \in D_i\} \qquad i = 1..6$$

para el problema de asignación de recursos tendríamos

$$t_i(x_i,u_i) = x_i - u_i = x_{i-1} \qquad i = 1,..4$$

el dominio de U_i es

$$U_i = \{(x,u) \mid \text{para cada } x \in S_i \text{ se tiene que } u = x,\ u \in D_i\}$$

$$U_i = \{(x,u) \mid \text{para cada } x \in S_i \text{ se tiene que } u \leq x,\ u \in D_i\} \qquad i = 2..4$$

UTILIDAD

En un problema de decisión generalmente sucede que al realizar una acción ante una situación determinada se incide en un "costo" o bien en una "ganancia", que genéricamente llamaremos utilidad. La utilidad es el recurso que queremos optimizar y que puede ser dinero, tiempo, distancia, disponibilidad de un insumo, etc.. En términos de nuestro lenguaje diremos que $f_i(x,u)$ representa la utilidad obtenida en la etapa i cuando nos encontramos en el estado x y tomamos la decisión u.

$$f_i : U_i \longrightarrow \mathbb{R}$$

Por ejemplo, en la red de la figura 2, $f_3(7,(5,7))= 5$; $f_5(12,(9,12))= 2$; en el problema de asignación de recursos, por ejemplo, $f_2(x,10)= 13$ con $x \geq 10$, $x \in S_2$.

1.2 RELACION DE RECURRENCIA

De todas las partes de la programación dinámica el concepto de la relación de recurrencia es el más importante, puesto que, en ella se ven representados todos los demás elementos y sus interrelaciones. Además es con esta relación como podremos encontrar una función para calcular el valor óptimo de todos los estados posibles de cada etapa. Debido a la importancia que tiene esta función, se presentará de manera formal. En lo sucesivo se supondrá que se desea maximizar la siguiente función:

$$J_n(x_n) = \underset{d_n}{\text{Max}} \left\{ \phi_n(f_n(x_n,u_n), f_{n-1}(x_{n-1},u_{n-1}), \ldots, f_1(x_1,u_1)) \right\} \quad (1.2)$$
$$\text{s.a.}$$
$$x_{i-1} = t_i(x_i,u_i); \qquad i= 2,..,n.$$

donde d_n constituye una política de decisión.

Definicion 1.1 Una política de decisión, para las primeras k etapas, de un estado x es una secuencia de decisiones, que podemos representar con

un vector

$$d_k(x) = (u_1, u_2, \ldots, u_k)$$

donde u_i es la decisión de la etapa i, i=1,2,..,k, y x es un estado de la etapa k. Si además u_i es factible para el estado x_i, de la etapa i, para toda i= 1..k, entonces diremos que $d_k(x)$ es una política de decisión factible para el estado x.

Escribiremos únicamente d_k en vez de $d_k(x)$ cuando no existan ambigüedades sobre el estado x de referencia. En general, cuando digamos política de decisión nos referiremos a una política de decisión factible para el estado x de referencia. Por ejemplo, en la red de la figura 2, si tenemos como función de transición a $t_1(k,(j,k)) = j$, la política de decisión $d_4(9)$= ((1,2),(2,5),(5,7),(7,9)) es una política de decisión factible pero $d_4(9)$= ((1,2),(2,5),(5,8),(7,9)) no lo es dado que $t_4(9,(7,9)) = 7$ y el arco (5,8) no es una decisión factible para el nodo 7. Por el mismo razonamiento, en el problema de asignación de recursos financieros, la política $d_3(20)$= (5, 10, 10) no es factible (dado que $t_3(20,10)$= 10 y $t_2(10,10)$= 0.=) en cambio $d_3(20)$= (5, 5, 10) si lo es.

Si queremos aplicar el algoritmo de la programación dinámica se deben verificar las tres condiciones siguientes:

a) Para todo estado x_i y decisión factible u_i de ese estado, debe existir una función de transición

$$x_{i-1} = t_i(x_i, u_i) \qquad i = 1..n$$

b) La función ϕ_n debe ser una función separable. Diremos que la función $\phi_n(f_n(x_n,u_n),\ldots,f_1(x_1,u_1))$ es separable si existe una función ϕ, tal que, para toda i, i=2..n se tiene que

$$\phi_i(f_i(x_i,u_i),\ldots,f_1(x_1,u_1)) =$$

$$= \phi\,(f_i(x_i,u_i),\ \phi_{i-1}(f_{i-1}(x_{i-1},u_{i-1}),\ldots,f_1(x_1,u_1)))$$

c) Se cumpla el principio de optimalidad de Bellman que dice:

Una política de decisión debe tener la propiedad siguiente: dados una decisión y un estado inicial, las decisiones restantes deben constituir una política óptima con respecto al estado resultante de la presente decisión y estado actual.

Proposición 1.2 Sea $J_i(x_i, d_i)$ el valor del estado x_i cuando se sigue una política de decisión d_i. Si

$$J_i(x_i, d_i) = \phi_i(f_i(x_i, u_i), \ldots, f_1(x_1, u_1)), \qquad \forall\ i$$

donde $d_i = (u_i, u_2, \ldots, u_i)$ y ϕ_i es una función separable, entonces $J_i(x_i, d_i)$ se puede representar recursivamente como

$$J_i(x_i, d_i) = \phi(f_i(x_i, u_i), J_{i-1}(x_{i-1}, d_{i-1})) \tag{1.3}$$

Prueba. Por hipótesis tenemos que existe una función ϕ tal que

$$J_i(x_i, d_i) = \phi(f_i(x_i, u_i),\ \phi_{i-1}(f_{i-1}(x_{i-1}, u_{i-1}), \ldots, f_1(x_1, u_1))) \tag{1.4}$$

y por definición tenemos que

$$J_{i-1}(x_{i-1}, d_{i-1}) = \phi_{i-1}(f_{i-1}(x_{i-1}, u_{i-1}), \ldots, f_1(x_1, u_1)) \tag{1.5}$$

por lo tanto sustituyendo la ecuación (1.5) en la ecuación (1.4) obtenemos la ecuación (1.3).

La relación de recurrencia en que se basa la programación dinámica (ecuación (1.3)) consiste en calcular la función $J_i(x_i, d_i)$ (de la etapa actual) en términos de una similar pero de "grado" menor (etapa anterior). Para efecto de saber cuando disponemos de una política óptima de decisión diremos que: d_i^* es una política óptima de decisión para las primeras i etapas, desde un estado x_i, si para toda política d_i tenemos

que

i) $J_i(x_i, d_i^*) \geq J_i(x_i, d_i)$ caso maximizar

ii) $J_i(x_i, d_i^*) \leq J_i(x_i, d_i)$ caso minimizar

En base a esta notación podemos interpretar el principio de optimalidad de Bellman de la siguiente forma : para toda política de decisión d_i de un estado x_i de la etapa i se tiene que

$$d_i(x_i) = (d_{i-1}^*, u_i) = (u_1^*, u_2^*, \ldots, u_{i-1}^*, u_i)$$

donde: d_{i-1}^* constituye una política óptima de decisión para algún estado x_{i-1} de la etapa anterior tal que

$$x_{i-1} = t_i(x_i, u_i)$$

cada vez que se verifique lo anterior diremos que se cumple con la condición de optimalidad.

Proposición 1.3 Sea $J_i(x_i, d_i) = \phi_i(f_1(x_1, u_1), \ldots, f_i(x_i, u_i))$ donde: ϕ_i es una función separable y $d_i = (u_1, u_2, \ldots, u_i)$ cumple con la condición de optimalidad. Si

$$J_i(x_i) = \max_{d_i} \{ \phi_i(f_1(x_1, u_1), \ldots, f_i(x_i, u_i)) \}$$
$$\text{s.a. } x_{j-1} = t_j(x_j, u_j) \quad j = 2..i$$

entonces $J_i(x_i)$ se puede calcular recursivamente como

$$J_i(x_i) = \max_{u_i} \{ \phi(f_i(x_i, u_i), J_{i-1}(x_{i-1})) \} \qquad (1.6)$$
$$\text{s.a. } x_{i-1} = t_i(x_i, u_i)$$

donde la función ϕ (asociada a la función separable ϕ_i), cumple con la siguiente propiedad: si $b \geq c$, entonces $\phi(a,b) \geq \phi(a,c)$ (no decreciente en el segundo argumento), para toda a.

Prueba. Por hipótesis tenemos que

$$J_i(x_i,(d_{i-1},u_i)) = \phi_i(f_i(x_i,u_i),\ldots,f_i(x_i,u_i))$$

como ϕ_i es separable se tiene que

$$J_i(x_i,(d_{i-1},u_i)) = \phi(f_i(x_i,u_i),J_{i-1}(x_{i-1},d_{i-1}))$$

por otro lado como d_i cumple con la condición de optimalidad se tiene que: $d_i = (d^*_{i-1},u_i)$; donde d^*_{i-1} constituye una política óptima para un estado x_{i-1}, por lo tanto tenemos que

$$J_i(x_i,d_i) = J_i(x_i,(d^*_{i-1},u_i)) = \phi\,(f_i(x_i,u_i),J_{i-1}(x_{i-1},d^*_{i-1})) \quad (1.7)$$
$$\text{s.a.}$$
$$x_{i-1} = t_i(x_i,u_i)$$

por definición tenemos que

$$J_i(x_i) = \max_{d_i}\{\,J_i(x_i,d_i)\,\} = J_i(x_i,d^*_i) \quad (1.8)$$

como $d_i = (d^*_{i-1},u_i)$; para toda política d_i (observese que la política óptima d^*_{i-1} no es la misma para toda d_i). En particular para la política óptima d^*_i tenemos que

$$d^*_i = (d^{**}_{i-1},u^*_i)$$

de donde tenemos que

$$J_i(x_i,d^*_i) = J_i(x_i,(d^{**}_{i-1},u^*_i)) \geq J_i(x_i,(d^*_{i-1},u_i))$$

$$= \phi(f_i(x_i,u_i),J_{i-1}(x_{i-1},d^*_{i-1}))$$
$$\text{s.a.}$$
$$x_{i-1} = t_i(x_i,u_i)$$

$$\geq \phi(f_i(x_i,u_i),J_{i-1}(x_{i-1},d_{i-1})) = J_i(x_i,d_i)$$
$$\text{s.a.}$$
$$x_{i-1} = t_i(x_i,u_i)$$

de la desigualdad anterior, observamos claramente que, en la etapa i, basta maximizar sobre el conjunto de decisiones de esa etapa, puesto que

disponemos de la política óptima de cualquier estado resultante (principio de optimalidad), o sea que

$$\max_{d_i} \{ J_i(x_i, d_i) \} = \max_{u_i} \{ J_i(x_i, (d^*_{i-1}, u_i)) \} \qquad (1.9)$$

resumiendo tenemos que

$$J_i(x_i) = \max_{d_i} \{ J_i(x_i, d_i) \} \qquad \text{por (1.8)}$$

$$= \max_{u_i} \{ J_i(x_i, (d^*_{i-1}, u_i)) \} \qquad \text{por (1.9)}$$

$$= \max_{u_i} \{ \phi (f_i(x_i, u_i), J_{i-1}(x_{i-1}, d^*_{i-1})) \} \qquad \text{por (1.7)}$$

s.a.

$$x_{i-1} = t_i(x_i, u_i)$$

$$= \max_{u_i} \{ \phi (f_i(x_i, u_i), J_{i-1}(x_{i-1})) \} \qquad \text{por (1.8)}$$

s.a.

$$x_{i-1} = t_i(x_i, u_i)$$

La ecuación (1.6) es la llamada función recursiva y con ella, como se ha demostrado en la proposición anterior podemos calcular el máximo de la ecuación (1.2) resolviendo recursivamente las n funciones $J_1, J_2, \ldots, J_n$, con la condición de que ϕ_n sea una función separable (con ϕ no decreciente en el segundo argumento) y de que para toda política d_i (i=1..n) se cumpla la condición de optimalidad.

Concluyendo, la programación dinámica consiste en resolver los problemas de optimización por medio de las funciones recursivas que acabamos de presentar. Para los problemas de minimización estas funciones tienen exactamente la misma forma y se puede comprobar fácilmente procediendo de una manera análoga. Hay que observar que como J_i está definida recursivamente en términos de J_{i-1}, cuando i es igual a uno tenemos una función J_0 que correspondería al valor máximo de la etapa cero que no existe, por lo que su valor no se calcula recursivamente sino que se da por definición de acuerdo con las características del problema. La

función J_0 se conoce como la CONDICION DE FRONTERA ya que con ella iniciamos el proceso de solución y generalmente se le define de manera que se cumpla la siguiente igualdad

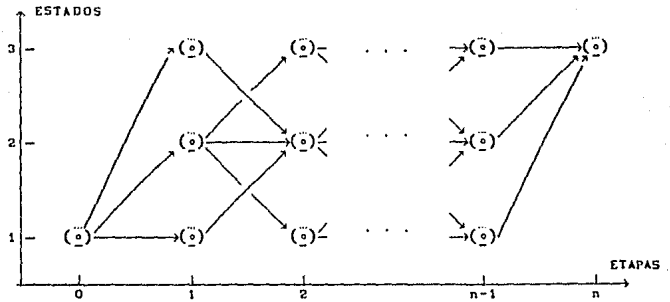
$$\phi_1(f_1(x_1,u_1)) = \phi(f_1(x_1,u_1),J_0(x_0))$$

1.3 FORMA HACIA ATRAS Y FORMA HACIA ADELANTE

En la programación dinámica existen dos formas de resolver los problemas, una de atrás hacia adelante (forward) y otra de adelante hacia atrás (backward). Estas nociones se dan en relación al tiempo o de un origen y un destino. En algunos problemas se pueden aplicar cualesquiera de las dos formas indistintamente y en otros se adecúa una forma más que la otra o simplemente sólo es posible aplicar alguna de las dos, como en los problemas estocásticos, donde sólo es posible aplicar la forma hacia atrás.

Supongamos que tenemos un problema dinámico, que podemos representar estructuralmente con la gráfica de la figura 4; y que es susceptible de ser resuelto por las dos formas. Si aplicamos la forma hacia adelante tendremos las siguientes características: las etapas se numeran en términos de "períodos recorridos" como se muestra en la figura 4. El proceso de solución empieza en el estado (nodo) (0,1) y termina en el estado (n,3). La función recursiva es del tipo de la ecuación (1.6). Por la estructura del problema, en donde las decisiones (arcos) "van" hacia "adelante", y la característica de la programación dinámica de que: dado un estado, se determina desde cuantos estados de la etapa anterior es posible llegar a él. Entonces intuitivamente diremos que aplicar esta forma es como "caminar" hacia adelante "mirando" hacia atrás.

Si desarrollamos la forma hacia atrás, sus características son: las etapas se numeran en términos de "períodos por recorrer" como se muestra en la figura 5. El proceso de solución empieza en el estado (0,3) y termina en el estado (n,1). La función recursiva es del tipo de la ecuación (1.6). Por las mismas razones expuestas para la forma hacia



NUMERACION DE ETAPAS HACIA ADELANTE
(PERIODOS RECORRIDOS)

figura 4

adelante, aquí diremos que intuitivamente éste procedimiento es como "caminar" hacia atrás "mirando" hacia adelante. Por ejemplo, en la red de la figura 2, si las etapas se numeran al revés, entonces el período dos corresponderá a la etapa cuatro, con un conjunto de estados

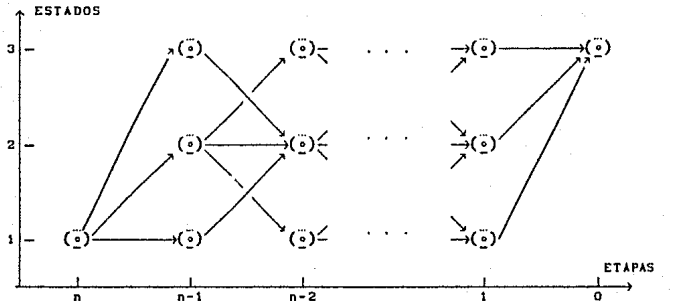
$$S_4 = \{4,5,6\}$$

que es el mismo que teníamos, en un ejemplo anterior, para la etapa dos con la forma hacia adelante, cuando se explicaba lo que eran las etapas, estados y decisiones; pero aquí el conjunto de decisiones factibles de cada estado es diferente, a saber

$$D(4) = \{(4,7)\};\ D(5) = \{(5,7),(5,8)\} \text{ y } D(6) = \{(6,8)\}$$

por lo tanto

$$D_4 = \{(4,7),(5,7),(5,8),(6,8)\}$$



NUMERACION DE ETAPAS HACIA ATRAS
(PERIODOS POR RECORRER)

figura 5

la función de transición para cada etapa es

$$t_i(x_i,u_i) = t_i(k,(k,m)) = m = x_{i-1} \; ; \qquad i = 1..6$$

donde el dominio de t_i es

$$U_i = \{(k,(k,m) \mid k \in S_i \text{ y } (k,m) \in D_i\} \qquad i = 1..6$$

Como comentarios podemos decir que la diferencia entre ambas formas es de concepto únicamente, como se observa comparando: la numeración de etapas (períodos recorridos v.s. períodos por recorrer); como se obtiene la solución (de atrás hacia adelante v.s. de adelante hacia atrás); y la perspectiva del problema ("visión" hacia atrás v.s. "visión" hacia adelante). Mientras que formalmente no existe diferencia alguna ya que ambas formas las podemos trabajar con la misma función recursiva.

La diferencia de concepto entre la forma hacia adelante y hacia atrás es sin embargo muy importante por que, si bien, con ambas formas obtenemos las mismas soluciones, la interpretación de los resultados intermedios es distinta y, por tanto, importante cuando nos interesan éstos. Por ejemplo, en la red de la figura 2 con la forma hacia adelante tendríamos el camino más corto del nodo 1 a cada nodo de la red. En cambio con la forma hacia atrás tendríamos el camino más corto de cada nodo de la red al nodo 14.

También la diferencia de concepto hace que la forma hacia atrás sea más utilizada, pues intuitivamente, esta forma se adapta en general más naturalmente a la estructura de los problemas reales, como podemos observar en el ejemplo de la red, con la forma hacia adelante una decisión es elegir el arco por medio del cual se va a llegar al estado (nodo) en que ya nos encontramos, lo que desde el punto de vista de recorrer un camino es absurdo, pues, el camino a ese estado "ya se recorrió", en cambio con la forma hacia atrás sería elegir el arco por el cual vamos a llegar a un estado del período siguiente.

Las características formales de la forma hacia atrás, si nos basamos en la gráfica de la figura 4 son: las etapas se numeran de manera "natural" hacia adelante, con lo que el proceso de solución empezaría en el estado (n,3) y terminaría en el estado (0,1). La función recursiva se expresaría como

$$J_i(x_i) = \max_{u_i} (\min) \{ \phi_i [f_i(x_i,u_i), J_{i+1}(x_{i+1})] \} \qquad (1.10)$$

s.a.

$$x_{i+1} = t_i(x_i,u_i)$$

con una condición de frontera J_{n+1} . Es fácil mostrar que la ecuación (1.10) tiene las mismas propiedades que la ecuación (1.6), puesto que, sólo se ha efectuado una redefinición de índices. A la ecuación (1.10) la definen en algunos textos de investigación de operaciones como: la forma hacia atrás de la ecuación recursiva de la programación dinámica.

1.4 FORMULACION DE UN PROBLEMA

En este apartado se intenta dar un esquema general de los pasos a seguir en la aplicación de la programación dinámica para el planteamiento y solución de problemas, la secuencia de estos pasos pretende ser únicamente una guía informal de una manera práctica y sistemática de proceder; y de ninguna manera un procedimiento rígido que se deba seguir al pie de la letra.

En primera instancia necesitamos determinar si nuestro problema puede ser resuelto mediante la programación dinámica, para ello necesitamos identificar cuatro elementos básicos que son: las etapas, los estados, las decisiones y la relación de recurrencia de la siguiente manera:

a.1) Definir claramente lo que significa estar en la etapa i del problema.

a.2) Definir el significado de "encontrarse" en el estado x_i de la etapa i.

a.3) Definir las acciones o decisiones factibles.

a.4) Definir el significado que tiene la función recursiva $J_i(x)$ para un estado x_i de la etapa i cuando se sigue una política óptima de decisión.

Posteriormente para poder aplicar el algoritmo de la programación dinámica se necesita:

b.1) Especificar los valores numéricos o nominales de los estados posibles de cada etapa, o sea definir S_i para cada i, $i = 0..n$

b.2) Dar el significado y los valores numéricos o nominales de las decisiones que se pueden efectuar a partir de los estados posibles de cada etapa, o sea definir D_i para cada i, $i = 1..n$.

b.3) Dar explícitamente la función de transición de estados

$$x_{i-1} = t_i(x_i, u_i)$$

b.4) Dar la utilidad asociada a cada decisión posible, para cada uno de los estados, de cada etapa, representada por $f_i(x_i, u_i)$.

b.5) Dar explícitamente la función recursiva

$$J_i(x_i) = \underset{u_i}{\text{Max}}\ (\text{Min})\ \{\ \phi\ (f_i(x_i, u_i), J_{i-1}(x_{i-1}))\}$$

s.a.

$$x_{i-1} = t_i(x_i, u_i)$$

o bien

$$J_i(x_i) = \underset{u_i}{\text{Max}}\ (\text{Min})\ \{\ \phi(f(x_i, u_i), J_{i-1}(t_i(x_i, u_i))\ \}$$

b.6) Definir el valor de la condición de frontera $J_0(x_0)$.

Finalmente, cuando se ha encontrado la función $J_n(x_n^0)$ que estamos buscando se tiene a la mano u_n^*, con ella podemos encontrar su estado asociado $x_{n-1}^0 = t_n(x_n^0, u_n^*)$ e identificar u_{n-1}^* y así, sucesivamente, hasta obtener todas las decisiones u_i^* $(i=1..n)$ que componen la política óptima de decisión. Puede suceder que la solución no sea única. Como ejemplos: de la forma hacia adelante ver el problema 4.2; de la forma hacia atrás ver el problema 4.1.

CAPITULO 2

LA PROGRAMACION DINAMICA ESTOCASTICA

Ante la imposibilidad de llegar a obtener un conocimiento completo de todos los fenómenos y procesos que ocurren en el mundo, ya sean naturales o sociales, determinan que en la vida diaria nos enfrentemos a la incertidumbre en un sinnúmero de situaciones, las cuales, en algunos casos es deseable conocer o controlar. Para tal efecto se han desarrollado técnicas estadísticas que nos ayudan a obtener un conocimiento aproximado de las características de estos fenómenos o procesos. La programación dinámica permite incorporar estos elementos de incertidumbre en su estructura y manejarlos de manera adecuada.

En este capítulo se contempla la aplicación de la programación dinámica en problemas estocásticos, para este fin el capítulo se estructura en tres partes. En la primera parte se define a la ecuación recursiva de la programación dinámica, cuando tenemos incertidumbre en el sistema. En la segunda parte se describen algunas propiedades de las cadenas de Markov, finitas e infinitas, y su relación con la programación dinámica. Por último, la tercera parte es un apéndice que consta de algunos conceptos y propiedades utilizados en la segunda parte.

2.1 CONCEPTOS Y COMPONENTES BASICOS

En el análisis de los problemas deterministicos hemos visto que dado un estado y una decisión es posible conocer el estado resultante en la etapa anterior por medio de la función de transición. Sin embargo no siempre es posible determinar con certeza ese estado resultante, como es el caso de los problemas estocásticos, puesto que el proceso dinámico depende, en parte, de la ocurrencia de un evento que no podemos controlar ni tampoco predecir con exactitud, por ejemplo, supongamos que deseamos determinar la política óptima de órdenes de un inventario, que debe satisfacer la demanda de los próximos N periodos, de tal manera que los costos de operación del inventario sean mínimos. Las órdenes se solicitan al inicio de cada periodo y son surtidas inmediatamente. Resolviendo este problema con la forma hacia atrás de la programación dinámica, definiremos a la etapa i como el número de periodos que faltan por recorrer y al estado x_i como el nivel del inventario al inicio de la etapa i. De esta manera el nivel de inventario del periodo siguiente (etapa anterior) se puede determinar mediante la siguiente relación $x_{i-1} = x_i + u_i - d_i$ donde u_i y d_i representan la cantidad ordenada y la demanda, respectivamente, de la etapa i. De aquí podemos observar que si la demanda es determinística (conocida para todas las etapas del proceso), la parte derecha de la ecuación anterior representa la función de transición del sistema. En cambio si la demanda es desconocida, no podremos determinar el estado resultante, debilitándose así la relación de recurrencia, base de la programación dinámica. Sin embargo, la programación dinámica también puede atacar este tipo de problemas introduciendo algunos cambios para adaptarse a la nueva situación, que discutiremos a continuación.

FUNCION DE TRANSICION

Obsérvese, del ejemplo anterior, que podríamos determinar el nivel inicial del inventario, del periodo siguiente, si conociéramos todos los valores posibles de la demanda. En general tenemos que, si conocemos la totalidad de los diferentes eventos probables (desconocidos) que pueden

llegar a suceder en cada etapa del proceso, tendremos los elementos suficientes para decir: que dada una decisión u_i y un evento probable r_i podemos determinar, a partir de un estado x_i de la etapa i, el estado resultante x_{i-1} de la etapa anterior, o sea

$$x_{i-1} = t_i(x_i, u_i, r_i)$$

FUNCION DE UTILIDAD

Es natural pensar que un evento , del cual depende parte del proceso, también influya en las utilidades del mismo, por tanto, debe ser tomado en cuenta para determinar la utilidad $f_i(x_i, u_i, r_i)$ de cada etapa.

FUNCION RECURSIVA

Procediendo con el mismo criterio que en el caso determinístico, se podría pensar que la función recursiva vendría dada por

$$J_i(x_i) = \underset{u_i}{\text{Max}} \text{ (o Min) } \{ \phi(f_i(x_i, u_i, r_i), J_{i-1}(x_{i-1})) \}$$

s.a.

$$x_{i-1} = t_i(x_i, u_i, r_i)$$

Ahora bien, la finalidad es tener en este momento algún conocimiento aproximado, en términos de valores esperados, de lo que va a suceder con el sistema, por lo tanto, no se obtiene ninguna ventaja definiendo la utilidad de esta manera; puesto que sólo podríamos conocer la posible ganancia máxima (o costo mínimo) después de que se hubieren completado las n etapas del proceso. Sin embargo, disponemos de todos los elementos que requerimos para realizar nuestro objetivo. Para ello es necesario asignar una medida de probabilidad a cada uno de los eventos de tal manera que, en cada etapa, la suma de sus probabilidades sea igual a uno. Tendremos así una variable aleatoria R_i bien definida para cada una

de las etapas y de esta forma definiremos el valor óptimo de un estado como: el valor óptimo esperado para ese estado. Así tenemos que

$$J_i(x_i) = \underset{u_i}{\text{Max}} \text{ (o Min) } \{ E [\phi(f_i(x_i, u_i, R_i), J_{i-1}(x_{i-1}))] \} \qquad (2.2)$$

s.a.

$$x_{i-1} = t_i(x_i, u_i, R_i)$$

o sea

$$J_i(x_i) = \underset{u_i}{\text{Max}} \{ \sum_j P[R_i = r_{ij}] \, \phi(f_i(x_i, u_i, r_{ij}), J_{i-1}(x_{i-1})) \}$$

s.a.

$$x_{i-1} = t_i(x_i, u_i, r_{ij})$$

Es importante hacer notar que en los problemas estocásticos sólo es posible aplicar la forma hacia atrás de la programación dinámica, pues, como lo habíamos visto en el apartado 1.3 (en la diferencia de fondo de ambas formas) la forma hacia atrás es como "caminar" hacia atrás "mirando" hacia adelante, lo que aquí se interpretaría como proyectar hacia el futuro; en cambio, con la forma hacia adelante obtendríamos una "proyección" hacia el pasado, lo que carece de sentido, ya que, supondría conocer con certeza el estado futuro (resultante) y no saber en cual estado presente nos encontramos, lo que claramente no es posible.

2.2 PROCESOS DE MARKOV Y PROGRAMACION DINAMICA

Se ha definido la ecuación recursiva de la programación dinámica estocástica (PDE) extendiendo de manera natural las ideas del capítulo anterior, agregando únicamente una variable aleatoria a las funciones definidas anteriormente de una manera matemática más que conceptual, con la intención de destacar que: lo importante en la aplicación de la programación dinámica es la identificación de etapas, estados y la relación de recurrencia. Sin embargo este punto de vista puede resultar deficiente, en virtud de la variedad de problemas donde es más natural y satisfactorio plantearlos como procesos de Markov. Por otra parte, con

el enfoque anterior, el horizonte a planear debe ser finito y bien definido, perdiéndose de vista la aplicación de la programación dinámica en la búsqueda de políticas estacionarias óptimas, las cuales, en buena medida, amplían el campo de aplicación de la programación dinámica.

Las propiedades y los resultados que se expondrán en lo sucesivo, relacionadas con los procesos de Markov, se darán sin prueba puesto que se necesitaría un desarrollo extenso y formal de la teoría, lo cual está fuera de los objetivos de esta tesis. Al final del capítulo se incluye un apéndice donde se dan algunas definiciones de conceptos y propiedades mencionadas en este apartado. En lo sucesivo supondremos que: el conjunto E es el espacio de estados; y que $p_{ij}^{(n)}$ representa la probabilidad de pasar del estado i al estado j en n etapas.

PROCESOS FINITOS DE MARKOV

Sea $Z_1, Z_2, \ldots, Z_n$ un proceso estocástico, donde la variable aleatoria Z_k (k=1..n) toma valores sobre un conjunto E; y donde puede suceder que el valor de la variable aleatoria Z_k dependa de los valores tomados por de las variables aleatorias anteriores, lo cual, podemos expresar con una probabilidad condicional

$$\text{Prob} \left(Z_{k+1}=J_{k+1} \mid Z_1=J_1, Z_2=J_2, \ldots, Z_k=J_k \right) \qquad k = 1..n$$

Definición 2.1 Un proceso estocástico es un proceso de Markov o una cadena de Markov, si

$$\text{Prob}(Z_{k+1}=J_{k+1} \mid Z_1=J_1, \ldots, Z_k=J_k) = \text{Prob}(Z_{k+1}=J_{k+1} \mid Z_k=J_k)$$

para todo k = 1..n.

Supongamos que $E = \{1, 2, \ldots, m\}$ y que

$$\sum_{j=1}^{m} \text{Prob} \left(Z_{k+1}=J \mid Z_k=i \right) = 1 \qquad \forall \; i, j \in E$$

entonces la probabilidad condicional es una probabilidad de transición, que representaremos como $p_{ij;k}$,cuando queramos identificar la etapa o tiempo en que se realiza la transición

$$\text{Prob}\ (\ Z_{k+1}=j \mid Z_k = i\ ) = p_{ij;k} \qquad k = 1..n-1$$

PROCESOS DE MARKOV CON GANANCIAS

Supongamos que tenemos un proceso de Markov donde se genera una utilidad o costo, $c_k(i,j)$, cuando pasamos del estado i de la etapa k al estado j de la etapa k+1. Si definimos el valor, $f(i)$, de un estado i como el promedio (la esperanza) total de las utilidades de las etapas restantes (incluyendo la etapa k), tendremos que:

$$f_k(i) = \sum_{j=1}^{m} p_{ij;k}\ \phi(c_k(i,j), f_{k+1}(j))$$

Supongamos que disponemos de un conjunto D de decisiones, donde la utilidad y la probabilidad de transición dependen de la acción elegida, entonces el valor óptimo de un estado se puede obtener como

$$J_k(i) = \max_{u\in D} \left\{ \sum_{j=1}^{m} p_{ij;k}(u)\ \phi(c_k(i,j,u), J_{k+1}(j)) \right\} \qquad (2.3)$$

para k = 1..n, con J_{n+1} como condición de frontera. Como podemos observar la ecuación (2.3) tiene la misma forma que la ecuación recursiva (2.2) de la programación dinámica estocástica, solo que en la forma hacia atrás, por lo que podemos utilizar la programación dinámica para resolver este tipo de problemas.

PROCESOS INFINITOS DE MARKOV CON GANANCIAS

Hasta el momento nos hemos enfocado a la búsqueda de políticas de decisión óptimas para procesos, donde el horizonte a planear se encuentra bien delimitado por ser finito. Pero ¿qué sucede cuando ese horizonte no está bien definido, por ser muy extenso?. ¿será posible encontrar una solución teóricamente aceptable?. Una respuesta a la segunda pregunta la podemos encontrar en la búsqueda de políticas estacionarias. Pero antes definiremos algunos conceptos que nos serán de utilidad.

Definición 2.2 Un proceso de Markov es homogeneo o estacionario, si la probabilidad de transición p_{ij} es estacionaria, donde

$$p_{ij} = \text{Prob}\ (\ Z_{n+1} = j \mid Z_n = i) \qquad n \in \mathbb{N} \text{ y } i, j \in E$$

Definición 2.3 Una utilidad es estacionaria si no depende del tiempo, o sea, si

$$c_n(i,j) = c(i,j) \quad \forall \quad n \in \mathbb{N} \text{ y para cada } i, j \in E$$

Considérese un proceso estacionario de Markov de un sistema ergódico completo, con un espacio de estados $E = \{1,2,\ldots,m\}$ y con matriz de transción P, en donde se genera una utilidad estacionaria $c(i,j)$ cada vez que el sistema hace una transición del estado i al estado j. Por último supóngase que el proceso consta de n etapas y que la función ϕ es de la siguiente forma

$$\phi(c(i,j), f(j)) = c(i,j) + f(j)$$

entonces tendremos que

$$f_k(i) = \sum_{j=1}^{m} p_{ij}\, c(i,j) + \sum_{j=1}^{m} p_{ij}\, f_{k+1}(j)$$

Si definimos a $\quad v_i = \sum_{j=1}^{m} p_{ij}\, c(i,j)$

entonces $$f_k(i) = v_i + \sum_{j=1}^{m} p_{ij} f_{k+1}(j)$$

ahora bien, sean

i) $F_k = (f_k(i))$ vector columna $i = 1..m$

ii) $V = (v_i)$ vector columna $i = 1..m$

de esta manera podemos escribir el valor de todos los estados en notación matricial obteniendo que

$$F_k = V + P F_{k+1} \qquad k = 1..n$$

podemos redefinir los indices (haciendo $k = n+1 - i$, $i = 1..n$) de tal manera que

$$F_k = V + P F_{k-1} \qquad k = 1..n$$

como el proceso tiene n-etapas observemos (con los indices redefinidos) que:

$$\begin{aligned} F_1 &= V + P F_0 \\ F_2 &= V + P F_1 = V + P V + P^2 F_0 \\ &\vdots \\ F_n &= V + PV + P^2V + \ldots + P^{n-1}V + P^nF_0 \\ &= (I + P + P^2 + \ldots + P^{n-1})V + P^nF_0 \end{aligned}$$

donde F_n representa la utilidad esperada, de todos los estados, para el proceso completo. Aplicando la igualdad de la ecuación (A.1) tenemos que

$$F_n = (nS_n + Y_n)V + S_nF_0$$

$$F_n = nS_nV + Y_nV + S_nF_0$$

Si tomamos el límite de F_n cuando $n \to \infty$, entonces $F_n \to \infty$, debido al primer sumando de F_n. Pero supongamos que aplicamos el límite obteniendo la siguiente expresión

$$F^{\infty} = nSV + YV + SF_0 \qquad (n \infty) \qquad (2.4)$$

donde $\lim_{n \infty} S_n = S$ y $\lim_{n \infty} Y_n = Y$

La importancia de la ecuación (2.4) radica en que F^{∞} representa un valor asintótico para F_n en sistemas ergódicos.

Cuando decimos que F^{∞} representa un valor asintótico para F_n queremos decir que:

$$f_{n-1}(i) \leq f_n(i) \leq ns_1v_1 + y_1v_1 + s_1f_0(i); \qquad \forall \ n.$$

o bien

$$f_{n-1}(i) \geq f_n(i) \geq ns_1v_1 + y_1v_1 + s_1f_0(i); \qquad \forall \ n.$$

Por ser nSV el elemento de F^{∞} que crece, conforme el número de etapas crece y por el significado de la matriz Y diremos que:

i) $g = SV$ es el vector de ganancias.

ii) $w = YV$ es el vector de sesgos.

Así tenemos que $F^{\infty} = ng + w + SF_0 \qquad (n \quad \infty) \qquad (2.5)$

POLITICA OPTIMA

Según podemos observar, para llegar a la expresión de la ecuación (2.5) hemos supuesto que el proceso y las utilidades generadas en el mismo son estacionarios. La razón de haber hecho estas hipótesis es perfectamente entendible si tomamos en cuenta que el número de etapas tiende a infinito. En consecuencia sería razonable pensar también que una política estacionaria es la adecuada para un proceso de tales

características. De lo anterior surge la siguiente pregunta: ¿Una política estacionaria es la mejor política?. La respuesta es sí, aunque por dificultades técnicas no haremos la prueba pero este hecho se encuentra plenamente justificado por Mine y Osaki (Mine, H. and Osaki,s., Markovian decision Processes).

Como comentario diremos que algunos autores (por ejemplo, White, D. J., Dynamic Programming.1969) consideran que, para sistemas periódicos, una política no-estacionaria es superior a una estacionaria. Quedando lo anterior sujeto al grado de conocimiento que se tenga acerca del proceso, pues Mine y Osaki mostraron también que una política estacionaria es superior a una no estacionaria aplicada aleatoriamente.

Como aquí tenemos incertidumbre en el sistema no podemos contemplar políticas de decisión como las definidas en el capítulo 1. Pero podemos reinterpretar la definición 1.1 de la siguiente manera: si $d_k(x)$ es una política de decisión, $d_k(x) = (u_1, u_2, \ldots, u_k)$, entonces u_i representa la decisión a tomar para el estado x en la etapa i, i=1..k, en vez de representar la decisión de la etapa i únicamente, como en el caso determinístico. O sea, para un proceso de Markov, una política de decisión es aquella donde disponemos de una acción a seguir para cada uno de los estados posibles, en cada una de las etapas. Pero ya que hablamos de políticas estacionarias conviene definir lo que entenderemos por este tipo de políticas.

Definición 2.4 Una política de decisión para un estado x es estacionaria, si la acción a seguir para el estado x es la misma en todas las etapas.

Es decir, si $d_k(x) = (u_1, u_2, \ldots, u_k)$ es una política de decisión estacionaria, entonces tenemos que $u_i = u_j$, para toda $i, j \in \{1..k\}$, o sea que $d_k(x) = (u, u, \ldots, u)$, por lo que a una política de decisión estacionaria la representaremos con un escalar, $d_n(x) = u$, únicamente.

Observación: Aun cuando la definición anterior está dada para un número finito de etapas (por como habíamos definido una política de decisión

para un estado x en el capítulo 2), ésta puede extenderse fácilmente a un número infinito de etapas observando que

$$d(x) = d_{\infty}(x) = \lim_{n \to \infty} d_n(x) = \lim_{n \to \infty} u = u$$

Definición 2.5 Una política de decisión es estacionaria, si es estacionaria para todos los estados posibles del proceso.

A una política estacionaria la representaremos con un vector columna u, donde el i-ésimo elemento de u representa la acción a seguir para el estado i en cualquier etapa del proceso. Con lo que, según la ecuación (2.5) tenemos que

$$F^{\infty}(u) = ng(u) + w(u) + SF_0(u) \qquad (n \longrightarrow \infty) \qquad (2.6)$$

esta ecuación representa la utilidad "esperada" de F_n bajo la política estacionaria u, cuando $n \longrightarrow \infty$.

La ecuación (2.6) es la ecuación que finalmente nos interesaba obtener, pues con ella ya estamos en condiciones de preguntarnos sobre una política estacionaria óptima. Pero antes de definir cuando disponemos de una política óptima de este tipo, haremos un par de observaciones:

1) Como hemos supuesto que el número de etapas tiende a infinito, podemos pensar que el proceso nunca termina, por lo que haremos a $F_0 = 0$, para no tomarlo en consideración y facilitar nuestro análisis, obteniendo así que:

$$F^{\infty}(u) = ng(u) + w(u) \qquad (n \longrightarrow \infty) \qquad (2.7)$$

2) Por otra parte vemos que, a causa del primer término de la ecuación (2.7), $F^{\infty}(u)$ crece indefinidamente bajo cualquier política de decisión. Entonces, si suponemos que nuestro objetivo es maximizar, nos interesará aquella política que haga crecer a $F^{\infty}(u)$ lo más rápido posible.

Definición 2.6 Una política estacionaria u^* es óptima, si para toda política estacionaria u tenemos que

$$g(u^*) \geq g(u)$$

y si para alguna política z tenemos que $g(u^*) = g(z)$, entonces necesariamente se cumple que

$$w(u^*) \geq w(z)$$

donde $g(u^*) \geq g(u)$ y $w(u^*) \geq w(z)$ significa que

$$g_i(u_i^*) \geq g_i(u_i) \quad y \quad w_i(u_i^*) \geq w_i(z_i) \qquad i = 1..m$$

Una política así definida siempre existe, como lo demostraron Mine y Osaki.

LA PROGRAMACION DINAMICA Y POLITICAS ESTACIONARIAS

La ecuación recursiva de la programación dinámica es útil para encontrar políticas estacionarias de ganancia óptima, puesto que, si tenemos un sistema de estados ergódico completo de un proceso homogeneo de Markov con utilidades estacionarias, entonces la ecuación (2.8) converge hacia una política estacionaria.

$$J_k(i) = \max_u \{ v_i(u) + \sum_{j=1}^{m} p_{ij}(u)\, J_{k-1}(j) \} \qquad (2.8)$$

donde $v_i(u) = \sum_{j=1}^{m} p_{ij}(u) c(i,j)$ es la utilidad esperada cuando se parte de un estado i y $c(i,j)$ es la utilidad estacionaria que se genera cuando el sistema pasa de un estado i a un estado j con probabilidad p_{ij}.

Además el algoritmo de la programación dinámica resulta muy práctico para la búsqueda de este tipo de políticas, pues la experiencia computacional a mostrado que su convergencia puede ser muy rápida (incluso de tres o cuatro iteraciones únicamente).

Hastins (Hastings, N. A. J.. Operations Research) nos ofrece una forma de saber que tan cerca nos encontramos de una política estacionaria óptima, de la siguiente manera.

Sean:

$$\Theta_k(i) = J_k(i) - J_{k-1}(i)$$

$$\Theta_{k,min} = \min_i \{\Theta_k(i)\}$$

$$\Theta_{k,max} = \max_i \{\Theta_k(i)\}$$

g_k la ganancia que se obtiene mediante la política estacionaria determinada con el algoritmo en la etapa k.

g^* la ganancia con la política estacionaria óptima.

entonces tenemos que

$$\Theta_{k,min} \leq g \leq g^* \leq \Theta_{k,max}$$

por lo tanto

$$g^* - g \leq \Theta_{k,max} - \Theta_{k,min}$$

Para concluir haremos la observación de que el algoritmo garantiza que hemos encontrado una política estacionaria de ganancia óptima, si $\Theta_{k,max} - \Theta_{k,min} = 0$. Pero no garantiza que ésta sea la política estacionaria óptima de la definición 2.6, pues no sabemos nada acerca el sesgo. Sin embargo para fines prácticos la relevancia de este hecho es menor cuanto mayor sea el número de etapas.

APENDICE

(CAPITULO 2)

En esta parte trataremos de formalizar en definiciones ciertos conceptos, contenidos en el capítulo, útiles para la exposición de los procesos de Markov. Puesto que un proceso de Markov es un tipo de proceso estocástico, empezaremos por definir a éste.

Definición A.1 Sea $E \subset \mathbb{R}$ un conjunto donde toma sus valores una variable aleatoria Z y sea T un conjunto de índices. Diremos que la sucesión de variables aleatorias $Z_{t_1}, Z_{t_2}, Z_{t_3}$ ... constituyen un proceso estocástico, si

i) $t_k \in T \qquad k = 1,2,3,\ldots$

ii) $t_k < t_{k+1}$

Diremos que el conjunto E es el espacio de estados del proceso. También dirémos que:

i) $Z = \{ Z_k \ / \ k \in \mathbb{N} \}$ es un proceso estocástico infinito (de tiempo discreto) .

ii) Si $T \subset \mathbb{N}$ y T es finito entonces $Z = \{ Z_k \ / \ k \in T \}$ es un proceso estocástico finito. Si la cardinalidad de T es de *n* elememtos diremos que el proceso se compone de *n* etapas.

Pensando en estados y etapas de un proceso estocástico, haremos las siguientes definiciones, donde, por comodidad, supondremos que el conjunto de estados, E, es finito; $E = \{ 1, 2, 3, \ldots, m \}$.

Definición A.2. Sea $p_{ij}^{(n)} = P[Z_{k+n} = j \mid Z_k = i]$; la probabilidad de pasar del estado *i* al estado *j* en *n* etapas. Diremos que esta probabilidad es una probabilidad de transición, si para cada estado *i* fijo, $i \in E$,

tenemos que

$$\sum_{j\in E} P[Z_{k+n} = j \mid Z_k = i] = 1$$

Además se dice que $p_{ij}^{(n)}$ es estacionaria si se tiene que

$$p_{ij}^{(n)} = \sum_{j\in E} P[Z_{k+n} = j \mid Z_k = i] = \sum_{j\in E} P[Z_n = j \mid Z_0 = i]; \quad \text{para cada } k \in \mathbb{N}$$

o sea que la probabilidad de transición $p_{ij}^{(n)}$ es independiente del tiempo.

Definición A.3 Sea P una matriz cuadrada de m x m y sea p_{ij} el elemento (i, j) de P, entonces diremos que P es una matriz de transición si

$i)$ $\quad 0 \le p_{ij} \le 1 \qquad \forall\ i, j \in E$

$ii)$ $\quad \sum_{j=1}^{m} p_{ij} = 1 \qquad \forall\ i \in E$

Es costumbre decir que p_{ij} representa la probabilidad de pasar del estado i al estado j en un paso (o etapa).

Proposición A.4 Si P es una matriz de transición entonces:

$i)$ P^n es una matriz de transición. Y si el proceso tiene probabilidades estacionarias, entonces el elemento (i, j) de P^n representa la probabilidad de pasar del estado i al estado j en n-etapas. Observese que en este caso podemos cálcular recursivamente a $p_{ij}^{(n)}$.

$$p_{ij}^{(1)} = p_{ij}$$

$$p_{ij}^{(2)} = \sum_{k=1}^{m} p_{ik}\, p_{kj}$$

$$\vdots$$

$$p_{ij}^{(n)} = \sum_{k=1}^{m} p_{ik}^{(n-1)} p_{kj}$$

ii) La matriz P es Cesaro sumable o sea

$$\lim_{n \to \infty} \frac{(P+P^2+ \ldots +P^n)}{n} = S$$

Donde S es la llamada matriz fija asociada con la matriz P. El elemento (i,j) de S representa la probabilidad de encontrarnos en el estado j en alguna etapa escogida aleatoriamente, habiendo empezado el sistema en el estado i.

Sea
$$Y = \sum_{n=0}^{\infty} (P^n - S) \qquad (A.1)$$

entonces Y se define como la matriz fundamental asociada con la matriz P. Donde el elemento (i,j) de Y representa el promedio de visitas "extras" (de más o de menos) al estado J, para un sistema que empieza en el estado i y continúa indefinidamente. Y es una especie de desviación (sesgo).

Hablar de estados y de probabilidades de transición entre ellos, nos induce a descubrir diferentes tipos de estados, los cuales, según sus propiedades, se pueden clasificar de la siguiente forma

- ESTADOS
  - TRANSITORIOS
  - RECURRENTES
    - ABSORBENTES
    - PERIODICOS
    - ERGODICOS

Antes de definir cada estado considerese la siguiente probabilidad

$$q_i^{(n)} = P[Z_n = i \mid Z_0 = i;\ Z_k \neq i,\ k = 1..n-1]$$

$q_i^{(n)}$ representa la probabilidad de que el primer retorno al estado i se efectúe exactamente en n etapas, dado que el estado inicial del proceso es ese estado i.

Definición A.5 Un estado i es transitorio si

$$\sum_{n=1}^{\infty} q_i^{(n)} < 1$$

La definición anterior quiere decir que cuando hemos abandonado un estado transitorio, el retorno a éste puede no volver a ocurrir durante el resto del proceso. Por ejemplo, en la matriz (a) de la figura 7, el estado 1 es transitorio.

Definición A.6 Un estado i es recurrente si no es transitorio, o sea si

$$\sum_{n=1}^{\infty} q_i^{(n)} = 1$$

Lo anterior significa que está garantizada la permanecia o al menos un eventual regreso al estado original.

Definición A.7 Un estado i es absorbente si $p_{ii} = 1$.

La definición anterior significa que cuando caemos en un estado absorbente, permaneceremos en él durante el resto del proceso. Por ejemplo, en la matriz (b) de la figura 7, el estado 2 es absorbente.

Definición A.8 Un estado i es periódico, de periodo α. Si para cada tiempo k, que no pertenece al conjunto $\{\alpha, 2\alpha, 3\alpha, \ldots\}$, se tiene que $p_{ii}^{(k)} = 0$. Donde α es el máximo común divisor del conjunto de las $n \geq 1$ tales que $p_{ii}^{(n)} > 0$.

Como podemos observar de la definición anterior un estado periódico es aquel que no puede ocurrir más que en ciertos periodos de tiempo. Por ejemplo, en la matriz (a), los estados 2, 3 y 4 son periódicos, de periodo 2.

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}
\begin{array}{c}
\begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\
\left[\begin{array}{cccc}
\frac{1}{2} & 0 & 0 & \frac{1}{2} \\
0 & 0 & \frac{1}{2} & \frac{1}{2} \\
0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0
\end{array}\right]
\end{array}
\qquad (a)
$$

$$
\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}
\begin{array}{c}
\begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\
\left[\begin{array}{cccc}
0 & 0 & \frac{1}{2} & \frac{1}{2} \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 \\
\frac{1}{2} & 0 & 0 & \frac{1}{2}
\end{array}\right]
\end{array}
\qquad (b)
$$

figura 7

Definición A.9 Un estado i es **aperiódico**, si el periodo es uno. Además si el promedio de primer retorno al estado i es finito, o sea

$$\sum_{n=1}^{\infty} n q_i^{(n)} < \infty$$

entonces se dice que el estado i es **ergódico**.

En la definición anterior vemos que un estado ergódico puede ocurrir en cualquier momento del proceso. Por ejemplo, en la matriz (b), los estados 1, 3 y 4 son ergódicos.

Definición A.10 Diremos que un **sistema es ergódico** bajo una política de decisión, si todos sus estados son ergódicos o si existen sólo algunos estados transitorios bajo esa política.

Definición A.11 Diremos que un **sistema es ergódico completo**, si es ergódico bajo cualquier política de decisión.

Proposición A.12 Si P es una matriz de transición de un sistema ergódico, entonces P tiene las siguientes propiedades:

i) $\lim_{n \to \infty} \frac{(P + P^2 + \ldots + P^n)}{n} = \lim_{n \to \infty} P^n = S$; donde S tiene todos sus renglones iguales.

ii) $SP = PS = S$

iii) $S = S^2 = S^n$

CAPITULO 3

EL PAQUETE CSUDP

Existen varios aspectos significativos de la programación dinámica que pueden ser efectivamente explotados en un software generalizado de aplicación, este es el caso del paquete CSUDP (Colorado State University Dynamic Programming). El código generalizado CSUDP de programación dinámica requiere más del usuario que, digamos, un paquete de programación lineal, es decir, un usuario de un paquete de este tipo no necesita tener conocimiemto del método simplex para poder utilizarlo. En cambio el uso de software generalizado de programación dinámica requiere cuando menos una comprensión de este método para poder utilizarlo.

La finalidad de este capítulo es describir las características y funcionamiento del programa CSUDP, para ello el capítulo se desarrolla de la siguiente manera: la primera sección presenta una breve historia del paquete y algunas de sus características principales; la segunda sección muestra en detalle las diferentes opciones del CSUDP; la tercera sección describe las subrutinas STATE, OBJECT y READIN, que son programadas por el usuario; por último, la cuarta sección se refiere a las facilidades del CSUDP para resolver problemas estocásticos unidimensionales.

3.1 CARACTERISTICAS GENERALES

El autor del CSUDP, John W. Labadie, se percató de la necesidad de disponer de un software generalizado de programación dinámica mientras enseñaba cursos de postgrado de ingeniería en la Universidad del estado de Colorado (CSU). El había percibido que varias personas, quienes podían hacer un uso efectivo de la programación dinámica en distintas áreas como sistemas de planeación, dirección y diseño, fueron desalentadas a utilizarla por la falta de un código general.

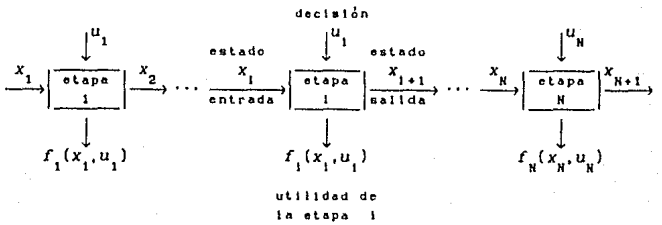
Primeramente, en 1975, Labadie desarrolló un pequeño código en FORTRAN para resolver problemas de asignación de recursos acompañándolo con un programa general para resolver problemas de control óptimo. Estos dos programas fueron implementados juntos dentro de un programa llamado DYNPRO, en 1977, los cuales fueron mejorados para incluir problemas estocásticos y también se les anexó un eficiente tomador de datos. Un código multidimensional llamado INCDP fue entonces desarrollado bajo la dirección de Labadie y posteriormente combinado junto con DYNPRO dentro de un código comprensible llamado CSUDP (Labadie and Shafer, 1980), desde entonces los esfuerzos se han concentrado en adaptar el paquete al uso en microcomputadoras, como se describe en Labadie, Et. (1984) y Labadie and Chou(1986).

La versión actual del CSUDP ha sido utilizada como una herramienta en cursos a nivel de postgrado y licenciatura en la universidad de Colorado (CSU) en: las áreas de ingeniería civil e ingeniería mecánica; los departamentos de Ingeniería de Agricultura y Química; así como en sistemas de ingeniería en general; el departamento de Matemáticas y Economía de Agricultura. Además el paquete ha sido muy utilizado en investigaciones de sistemas de recursos hidráulicos y el manejo de recursos naturales. El código ha sido distribuido a nivel mundial a varias universidades y empresas privadas, e implementado en una gran variedad de equipos de computo.

La más reciente versión del CSUDP está escrita en FORTRAN 77, específicamente para la IBM (computadora personal) y el compilado se

realiza con el sistema microsoft fortran versión 3.20, el cual es necesario para compilar las subrutinas que elabora el usuario y crear el código objeto, que debe ligarse con el código general. Dos versiones del código objeto del CSUDP están disponibles, una de las versiones requiere de un coprocesador INTEL 8087 MATH y la otra puede trabajarse sin este requerimiento. Para el uso del paquete, se debe tener un mínimo de 256k bytes de memoria principal pero para problemas grandes se recomienda disponer de 640k bytes para emular, en memoria principal, el acceso a disco.

El programa CSUDP ha sido diseñado para dar la máxima flexibilidad al usuario para resolver una amplia gama de problemas de decisión secuencial. Para este fin el usuario debe desarrollar tres subrutinas (READIN, STATE y OBJECT), que esencialmente permiten simular el comportamiento del sistema y calcular el valor de las ganancias o costos de cada etapa así como el valor de la función objetivo.



TIPICA ESTRUCTURA POR ETAPAS
PARA PROBLEMAS SECUENCIALES DE DECISION

figura 8

En la figura 8 se representa un típico problema secuencial por etapas, donde una ganancia o costo se produce en cada etapa. La transición de

una etapa a la otra se efectua como la transición de estados (x_i) de etapas contiguas, mediante decisiones independientes (u_i) ejecutadas en cada etapa. O sea, la relación entre etapas se debe expresar como una relación entre estados. Esta relación la definimos con la función de transición de estados

$$x_{i+1} = t_i(x_i, u_i),$$

El paquete CSUDP resuelve problemas de multietapas con la forma "hacia atrás" (backward) de la función recursiva. Sin embargo es posible, para el usuario, resolver problemas en la forma "hacia adelante" (forward, donde $x_{k-1} = t_k(x_k, u_k)$, k = 1..N) invirtiendo los estados de entrada y salida; y redefiniendo índices (i=N+1-k; para obtener la función de transición $x_{i+1} = t_i(x_i, u_i)$, i=1,...,N). Problemas de una dimensión como los ilustrados en la figura 8 son resueltos mediante el procedimiento estandar de la programación dinámica, mientras que problemas con vectores de estado multidimensionales son resueltos por el método de la programación dinámica incremental (Larson 1968).

El CSUDP puede resolver también problemas estocásticos, y lo puede hacer de dos formas distintas, con respecto a la transición entre etapas. Una forma es tener una estructura de variables aleatorias independientes (por ejemplo, en un problema de control de inventarios la demanda esperada es independiente del nivel del inventario). La otra consiste en suponer que la ocurrencia de cada estado, de una etapa, depende de los estados de la etapa anterior según una función de distribución de probabilidad (procesos de Markov). Ciertas clases de problemas de decisión, vistos como procesos estacionarios de Markov, pueden ser resueltos por aproximaciones sucesivas. Donde políticas estacionarias pueden ser halladas en un número finito de iteraciones, en sistemas ergódicos y periódicos.

OPCIONES DE LA FUNCION OBJETIVO

Para problemas deterministicos, los siguientes tipos de la función objetivo pueden ser considerados :

TIPO 1

$$J_1(x_1) = \underset{d_N}{\text{Min (o Max)}} \left\{ \sum_{i=1}^{N} f_i(x_i, u_i) \right\} \qquad (3.1)$$

TIPO 2

$$\left.\begin{aligned} J_1(x_1) &= \text{Min} \left\{ \underset{d_N}{\text{Max}} \left\{ f_1(x_1,u_1), \ldots, f_N(x_N,u_N) \right\} \right\} \\ J_1(x_1) &= \text{Max} \left\{ \underset{d_N}{\text{Min}} \left\{ f_1(x_1,u_1), \ldots, f_N(x_N,u_N) \right\} \right\} \end{aligned}\right\} \qquad (3.2)$$

TIPO 3

$$J_1(x_1) = \underset{d_N}{\text{Min (o Max)}} \left\{ \prod_{i=1}^{N} f_i(x_i, u_i) \right\} \qquad (3.3)$$

Donde $d_N = (u_1, \ldots, u_N)$ es una política de decisión (ver capítulo 1).

En el CSUDP todas las funciones $f_i(x_i, u_i)$ deben ser positivas, para el uso de la función del tipo 3. Como el CSUDP define a la condición de frontera, $J_{N+1}(x_{N+1}) = 0$, entonces cualquier beneficio o costo adicional del estado x_{N+1} se puede incluir en la función de utilidad $f_N(x_N, u_N)$.

Note que utilizando el CSUDP, en la forma hacia atrás, la etapa i representa el periodo i del sistema (indexado en forma natural), mientras que, en la forma hacia adelante, la etapa i representa el número de periodos que faltan por recorrer en la evolución del proceso. El usuario debe introducir los datos tomando lo anterior en cuenta.

LAS ECUACIONES DE TRANSFORMACION DE ESTADOS

La ecuación de transición de estados es suministrada por el usuario en la subrutina STATE, en la forma general:

$$x_{i+1} = t_i(x_i, u_i), \qquad (i=1..N)$$

suponiendo que tenemos como función recursiva a:

$$J_i(x_i) = \underset{u_i}{\text{Max}} \left\{ \phi(f_i(x_i,u_i), J_{i+1}(x_{i+1})) \right\} \qquad (3.4)$$

$$\text{s.a } x_{i+1} = t_i(x_i, u_i)$$

En algunos casos puede ser ventajoso expresar la ecuación de transición de estados en forma invertida, es decir resolverla para la variable de decisión u_i, en vez de la variable de estado x_{i+1}, es decir, expresarla como

$$u_i = t_i^{-1}(x_i, x_{i+1}), \qquad i=1..N$$

de esta manera la optimización se realiza directamente sobre la variable de estado x_{i+1}, con lo que la ecuación recursiva se expresaría como

$$J_i(x_i) = \max_{x_{i+1}} \{\phi(f_i(x_i, u_i), J_{i+1}(x_{i+1}))\} \qquad (3.5)$$
$$\text{s.a } u_i = t_i^{-1}(x_i, x_{i+1})$$

obteniendose también, con esta nueva forma, el valor óptimo de la función objetivo, como lo demuestra la siguiente proposición:

Propocisión 3.1 Las ecuaciones (3.4) y (3.5) son equivalentes.

Prueba. (⇒) Supongamos que tenemos la ecuación (3.4), a ésta la podemos expresar como

$$J_i(x_i) = \max_{u_i} \{\phi(f_i(x_i, u_i), J_{i+1}(t_i(x_i, u_i))\} \qquad (3.6)$$

en la ecuación (3.6) vemos claramente como la variable u_i afecta a ambos argumentos de la función $\phi(\cdot)$. Ahora bien supongamos que existe una relación uno a uno entre u_i y x_{i+1}, entonces podemos expresar a u_i como función de x_i y x_{i+1} o sea

$$u_i = t_i^{-1}(x_i, x_{i+1}), \qquad i=1..N \qquad (3.7)$$

la función (3.7) se cumple para toda u_i, en particular para el valor óptimo u_i^*, a la cual le corresponde uno y sólo un valor de la variable de estado x_{i+1} que denotaremos por x_{i+1}^*. De donde tenemos que

$$\max_{u_i} \{\phi(f_i(x_i, u_i), J_{i+1}(t_i(x_i, x_i)))\} = \phi(f_i(x_i, u_i^*), J_{i+1}(t_i(x_i, u_i^*)))$$

$$= \phi(f_i(x_i, t_i^{-1}(x_i, x_{i+1}^*)), J_{i+1}(x_{i+1}^*))$$

como la función (3.7) es uno a uno entonces no existe un estado x'_{i+1} tal que

$$\phi(f_i(x_i, t_i^{-1}(x_i, x_{i+1}^*)), J_{i+1}(x_{i+1}^*)) < \phi(f_i(x_i, t_i^{-1}(x_i, x'_{i+1})), J_{i+1}(x'_{i+1}))$$

por lo tanto tenemos que

$$\phi(f_i(x_i, t_i^{-1}(x_i, x_{i+1}^*)), J_{i+1}(x_{i+1}^*)) = \underset{x_{i+1}}{\text{Max}} \{\phi(f_i(x_i, t_i^{-1}(x_i, x_{i+1})), J_{i+1}(x_{i+1}))\}$$

$$= \underset{x_{i+1}}{\text{Max}} \{\phi(f_i(x_i, u_i), J_{i+1}(x_{i+1}))\}$$

$$\text{s.a.}$$

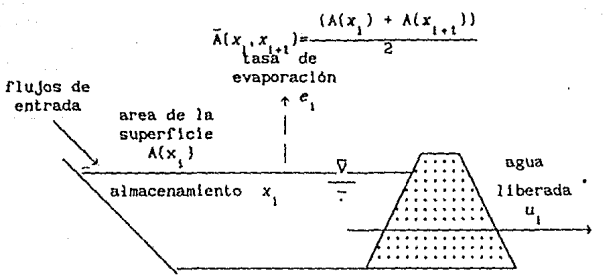
$$u_i = t_i^{-1}(x_i, x_{i+1})$$

el recíproco se demuestra de una manera análoga.

El resultado anterior nos muestra que para pasar de la forma estandar a la invertida o viceversa únicamente se necesita modificar la función de transición de estados, sin tener que modificar la función de utilidad ni tampoco la ecuación recursiva, lo cual es muy conveniente. También puede apreciarse en la ecuación (3.5) que cuando optimizamos sobre la variable x_{i+1} (la variable de estado de la etapa calculada previamente), la variable de decisión u_i pasa a ser una variable dependiente. Una ventaja particular de esta forma es que las relaciones de transición entre estados se pueden expresar como funciones implícitas del estado resultante, lo cual a veces es conveniente, por ejemplo, considérese el problema de operación de una presa (ver figura 9), donde la relación de transición de estados está dada por

$$x_{i+1} = x_i - u_i + r_i - e_i \bar{A}(x_i, x_{i+1}) \qquad (3.8)$$

donde: x_i es lo almacenado en la presa al principio del mes i; u_i es la cantidad de agua liberada para el consumo durante el periodo; r_i es el flujo no regulado del mes; e_i es la tasa de evaporación (por ejemplo, metros cúbicos por mes); y $\bar{A}$ es el area promedio de la superficie del vaso de la presa durante todo el mes i.



EJEMPLO DEL SISTEMA DE TRANSFORMACION DE ESTADO-BALANCE DE LA MASA DEL VASO

figura 9

donde: A(x) significa el área de la superficie del vaso, al nivel de almacenamiento x, tomada de los datos elevación-área-capacidad efectuados en el lugar. Como podemos ver, x_{i+1} aparece en ambos lados de la ecuación de transición (3.8) y se podría requerir de un proceso iterativo para determinar x_{i+1}, sin embargo esta ecuación se invierte fácilmente, quedando: $u_i = x_i - x_{i+1} + r_i - e_i \bar{A}(x_i, x_{i+1})$. De esta manera, se efectúa la optimización directamente con respecto a x_{i+1}.

En problemas unidimensionales el usuario tiene la opción de definir la función de transformación de estados entre la forma invertida o la forma estandar (no invertida). Pero en los problemas multidimensionales la función de transición siempre debe ser expresada en la forma invertida.

3.2 DESCRIPCION DE RESTRICCIONES Y OPCIONES

En el CSUDP, las variables de estado y decisión se representan de forma numérica, así tenemos que las restricciones para estas variables se

expresan con una cota inferior y una cota superior, esto es:

$$x_{i,min} \leq x_i \leq x_{i,max} \qquad (i=1,2,..,N+1) \qquad (3.9)$$

$$u_{i,min} \leq u_i \leq u_{i,max} \qquad (i=1,2,..,N) \qquad (3.10)$$

donde (3.9) y (3.10) representan el conjunto de estados S_i y el conjunto de decisiones D_i respectivamente de la etapa i (ver capítulo 1). Nótese que para los estados se define una etapa de más, esto es porque a la condición de frontera se le considera como una etapa.

De ser necesario, restriciones adicionales pueden ser consideradas en una o varias etapas del proceso. Por ejemplo, supongamos que tenemos un número s de restricciones adicionales para cada etapa, las cuales podemos representar como un vector de $\mathbb{R}^s$ donde $h_i(x_i,u_i) \leq 0$, $i= 1,..,N$; entonces requerimos de una función de penalización $w_i(\cdot)$, que se asigna a una variable PNALTY

$$\text{PNALTY}=\begin{cases} w_i(h_i(x_i,u_i)) & \text{si } h_{ij}(x_i,u_i) > 0, \text{ para algún } j=1,..,s \\ 0 & \text{en otro caso} \end{cases}$$

donde $w_i(\cdot)$ es una función especificada por el usuario en la subrutina OBJECT. De esta forma, el CSUDP suma automáticamente el término PNALTY a la función de utilidad, $f_i(x_i,u_i)$, en problemas de minimización (o lo resta en problemas de maximización). Esta operación tiene efecto sólo en la etapa i, pues el CSUDP resta (o suma) dicho término antes de pasar a la etapa siguiente. Para el caso en que se desee acarrear la penalización, en cálculos subsecuentes del proceso, el CSUDP permite sumar (o restar) directamente el término PNALTY a la función de utilidad.

El usuario debe tener cuidado al definir las cotas, pues el CSUDP suspende el proceso de optimización, si ha ocurrido una infactibilidad, desplegando un mensaje donde declara la etapa en la cual ha ocurrido el error. En estos casos el usuario debe relajar una o más cotas y probar

de nuevo. A menudo es útil, además de redefinir las cotas más allá de los límites deseables, incluir penalizaciones en la función objetivo con el fin de descubrir las violaciones a los límites originales. Las cotas relajadas permitirán al CSUDP completar la solución. Así el usuario podrá determinar si el problema ha sido planteado incorrectamente.

DISCRETIZACION

Como se vio en el párrafo anterior, los conjuntos de estados (S_i) y de decisiones (D_i) posibles se proporcionan en forma de intervalos, por medio de las variables x_i y u_i respectivamente. Estas variables sólo pueden tomar valores discretos, para poder ser manejados por el paquete. Aún cuando teóricamente las variables de estado o de decisión fueran continuas se puede definir una discretización, lo suficientemente fina, que proporcione una solución aproximada, útil para problemas prácticos. También se requiere definir los estados y decisiones de manera que dicha discretización sea uniforme en ambas variables, lo que generalmente es posible.

Una discretización uniforme para la variable de estado se efectúa mediante la constante DELX, que es la misma en todas las etapas. Si para algún problema esto fuera inapropiado, entonces es posible escalar los estados y las ecuaciones de transición de estados, de manera acorde, en la en la subrutina STATE. Por su parte, la variable de decisión es discretizada mediante la constante DELU.

Cuando es utilizada la forma invertida (lo que quiere decir que la optimización se efectua con la variable de estado x_{i+1}) el valor resultante de la variable de decisión (recuerde que en el caso invertido se tiene que $u_i = t_i^{-1}(x_i, x_{i+1})$) es redondeado a un valor que concuerde con la discretización efectuada mediante la variable DELU. Por lo tanto, lo mejor es hacer a DELU lo más pequeño que se pueda, a menos que las decisiones deban ocurrir en incrementos discretos determinados. Por supuesto, en este último caso lo mejor es utilizar la forma no invertida, cuando sea posible.

La selección de DELX es extremadamente importante, dado que afecta el tiempo de ejecución, los requerimientos de memoria y la precisión de la solución. El dimensionamiento del CSUDP requiere, para los problemas unidimensionales, que DELX y las Cotas de los intervalos sean seleccionados de tal manera que se cumpla la siguiente desigualdad:

$$\frac{(x_{i,max} - x_{i,min})}{DELX} \leq 101 \qquad i = 1,..,N$$

Cuando el esfuerzo computacional es demasiado grande se dispone de una opción de "empalme", con la cual el usuario puede reducir en gran medida el número de cálculos. En esta modalidad, se especifica un intervalo "burdo" inicial con discretización DELXI, y otro intervalo final con discretización DELXF, donde DELXF < DELXI. Tambien se selecciona un parámetro SPLICE > 1. Con estos tres valores (DELXI, DELXF y SPLICE) el CSUDP realiza una serie de iteraciones, donde para cada iteración k se tiene que:

a) Después de haber encontrado una solución óptima sobre todas las etapas en la iteración k-1, se define un nuevo "corredor" más estrecho al rededor de esta solución y se calcula una nueva constante de discretización DELX, para la iteración k, de la siguiente forma:

$$DELX^{(k)} = \frac{DELX^{(k-1)}}{SPLICE}, \quad \text{donde} \quad DELX^{(0)} = DELXI \qquad (3.11)$$

b) El proceso se detiene cuando $DELX^{(k)} \leq DELXF$.

El uso de la opción de "empalme" puede reducir significativamente el tiempo de ejecución, pero existe el riesgo de no encontrar la solución óptima global si no se procede con sumo cuidado en la definición de los parámetros.

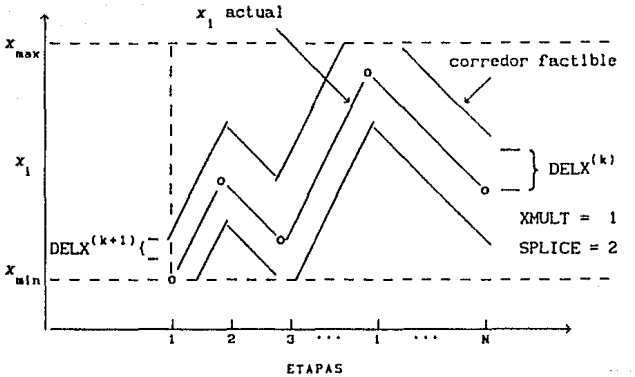


figura 10

Con esta opción de empalme es posible controlar el tamaño del corredor que se forma al rededor de la solución óptima de la iteración previa (o sea el tamaño de la nueva región de estados factibles) por medio de un parámetro entero XMULT ≥ 1. Esto es, para la iteración k, los estados x_i serán restringidos a variar en el siguiente intervalo

$$\text{Max} \left\{ x_i^{*(k-1)} - \text{XMULT} \cdot \text{DELX},\ x_{i,min} \right\} \le x_i$$

$$x_i \le \text{Min} \left\{ x_i^{*(k-1)} + \text{XMULT} \cdot \text{DELX},\ x_{i,max} \right\}$$

donde $x_i^{*(k-1)}$ es la solución óptima encontrada en la iteración anterior (k-1). Si la solución óptima de la iteración k ($x_i^{*(k)}$) se encuentra sobre la frontera del "corredor", en cualesquiera de las etapas, entonces el empalme no ocurre en la siguiente iteración. Sólo si el estado resultante, de cada etapa, se encuentra completamente en el

interior del corredor se reduce DELX. La figura 10 ilustra la forma como trabaja el "empalme" y de como los estados están restringidos a permanecer dentro del corredor definido para la iteración k.

Si en una iteración, el valor de la función objetivo es una repetición del valor de la función objetivo de la iteración previa, entonces se asume que ha ocurrido una convergencia a un óptimo local, terminando el proceso de optimización. En otro caso, se define un nuevo corredor con XMULT incrementos por arriba y por abajo de la solución $x_i^{*(k)}$ encontrada y el proceso continúa. Debe notarse que la opción de empalme sólo es ventajosa cuando se buscan políticas óptimas abiertas. Las políticas retroalimentadas estarán restringidas al pequeño número de estados del "corredor" final y por lo tanto podrían no ser de utilidad. Por último, si se sospecha haber encontrado un punto silla en vez de un óptimo local, se dispone de una opción de rompimiento de empates , la cual podría hacer que el proceso de solución continúe.

FUNCION DE UTILIDAD OPTIMA

Para problemas unidimensionales, las funciones $J_i(x_i)$ evaluadas con el CSUDP representan la ganancia máxima (costo mínimo) total para un proceso secuencial de decisión que empieza en la etapa i, en el estado x_i, y que continúa hasta la etapa N. Esto es, la función $J_i(x_i)$ representa una función recursiva de la forma hacia atrás, donde las etapas están indexadas de forma natural, o sea

$$J_i(x_i) = \max_{u_i} \{\phi(f_i(x_i,u_i), J_{i+1}(x_{i+1}))\} \qquad (i=1,2,..,N)$$

donde $\phi(\cdot)$ puede representar a cualesquiera de los tipos de las ecuaciones (3.1), (3.2) y (3.3). La optimización está sujeta a:

i) $x_{i+1}=t(x_i,u_i)$

ii) $x_{i,min} \le x_i \le x_{i,max} \qquad (i=1,2,..,N+1)$

iii) $u_{i,min} \le u_i \le u_{i,max} \qquad (i=1,2,..,N)$

e indirectamente (si hubiera términos de penalización)

i) $\quad h_i(x_i, u_i) \leq 0 \qquad (i=1,2,..,N)$

La solución del proceso comienza con la condición de frontera: $J_{N+1}(x_{N+1})=0$, para cualquier estado x_{N+1}. En caso de tener costos terminales para los estados X_{N+1}, éstos pueden ser incluidos en la etapa N.

Obsérvese que el valor $J_{i+1}(x_{i+1})$ está almacenado sólo para los valores discretos de las x_{i+1}, definidos en la etapa $i+1$, por lo que no existe garantía de que la función de transición, $x_{i+1}= t_i(x_i, u_i)$, calculada caiga exactamente en alguno de esos valores discretos. En estos casos el CSUDP efectúa una interpolación lineal (figura 11) a menos que la opción de empalme haya sido utilizada.

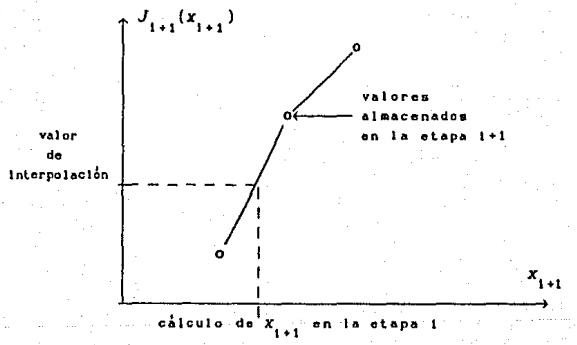


figura 11

Para los problemas resueltos en la forma invertida (función de transición $u_i = t_i^{-1}(x_i, x_{i+1})$) no existe la necesidad de interpolar, puesto que, el CSUDP garantiza que la optimización se realiza únicamente sobre aquellos valores x_{i+1} evaluados en la etapa $(i+1)$.

Si el valor de la función de utilidad $f(\cdot)$ involucra valores tabulados para los valores discretos de la variable de decisión u, entonces la interpolación de $f(\cdot)$ debe ser codificada por el usuario en la subrutina OBJECT.

En el caso unidimensional la optimización se efectúa sobre todos los valores discretos u_i (ó x_{i+1}) para asegurar que se alcanza un óptimo global. El paquete almacena sólo estados factibles para los cálculos de la siguiente etapa.

Al término de cada etapa, los valores: $J_i(x_i)$ y $u_i^*(x_i)$ (o $x_{i+1}^*(x_i)$ para el caso invertido); son almacenados para cálculos posteriores. Como $J_i(x_i)$ será utilizada inmediatamente en la etapa siguiente $(i-1)$, ésta se guarda siempre en memoria principal, mientras que las políticas óptimas $u_i^*(x_i)$ se almacenan en un archivo en disco, puesto que sólamente son utilizadas durante el "rastreo de regreso" después de que todas las etapas han sido calculadas.

El CSUDP obtiene la política óptima de cada estado (x_1) de la etapa uno, aplicando el "rastreo de regreso" de la siguiente forma:

Forma no invertida:

Paso 1. Se selecciona un valor determinado de x_1; $i=1$.

Paso 2. Se lee $u_i^*(x_i)$ del archivo de disco (donde: de ser necesario, x_i se redondea al valor más cercano del intervalo generado por DELX).

Paso 3. Se llama a la subrutina STATE para obtener x_{i+1}^*; $x_{i+1} = x_{i+1}^*$.

Paso 4. Si: $i = N$ entonces terminar. En otro caso $i = i+1$; ir al paso 2.

Forma invertida:

Paso 1. Se selecciona un valor determinado de x_1; $i=1$.

Paso 2. Se lee $x^{*}_{i+1}(x_i)$ del archivo de disco.

Paso 3. Se llama a la subrutina STATE para obtener u^{*}_{i}; $x_{i+1} = x^{*}_{i+1}$.

Paso 4. Si: $i = N$ entonces terminar. En otro caso $i = i+1$; ir al paso 2.

Este proceso se repite para cada valor que puede tomar la variable x_1 en la primera etapa, o sea, para todos los estados iniciales. Para los problemas no invertidos o estandar la subrutina OBJECT es llamada durante el proceso de rastreo para obtener una determinación precisa de la utilidad total, dado que $J_1(x_1)$ podria contener alguna inexactitud, por ejemplo, debido al posible uso de la interpolación durante el proceso de óptimización.

PROCEDIMIENTO PARA ROMPER EMPATES

El CSUDP dispone de una opción para romper empates en el caso de que no haya una solución única. El paquete guarda la primera solución encontrada, o bien, puede guardar la última solución, de acuerdo con la opción que el usuario haya elegido. De esta forma es posible determinar si se ha encontrado una solución única, repitiendo el procedimiento para romper empates. Nótese que cuando la solución no es única, el programa sólo puede determinar dos de ellas.

PROBLEMAS MULTIDIMENSIONALES

Los problemas multidimensionales (donde la dimensión del vector de estados es mayor que uno) la optimización se realiza mediante el método de la programación dinámica incremental (Larson, [4]), por lo tanto deben proporcionarse al programa trayectorias iniciales de estado factibles $x_{ij}^{(0)}$, fijando siempre el tamaño del corredor que circunda a las soluciones sucesivas $x_{ij}^{(k)}$ con el parámetro XMULT = 1. Esto significa

que se evalúan en cada iteración un máximo de 3^m estados discretos factibles, donde m es la dimensión del vector de estado x_i. En la iteración k tenemos:

$$\max [x_{ij}^{(k-1)} - DELX,\ x_{ij,min}] \leq x_{ij} \leq \min [x_{ij}^{(k-1)} + DELX,\ x_{ij,max}]$$

$$(i=1,..,N;\quad j=1,..m)$$

donde el vector de estado en la etapa i es $x_i = (x_{ij})$, $j=1,..,m$. Los estados se restringen entonces a permanecer dentro del corredor correspondiente de la iteración k. Si la solución obtenida en la iteración actual es la misma de la etapa anterior, se asume que se ha convergido a un óptimo local discreto. De no ser este el caso, se define un corredor nuevo con un incremento $DELX^{(k)}$ hacia arriba y hacia abajo de la nueva solución $x_i^{(k)}$, repitiendose el proceso como se había explicado anteriormente. De igual forma la opción para romper empates puede usarse para continuar el proceso, si se sospecha haber encontrado un punto silla en vez de un óptimo local. También, en este caso,está disponible la opción de empalme, por medio de la cual, una vez que se ha presentado la convergencia para un determinado DELX, se calcula, por medio de la ecuación (3.11), una DELX más refinada y el proceso vuelve a repetirse.

No necesariamente las soluciones sucesivas $x_{ij}^{(k)}$ han de producir valores factibles u_{ij} en la forma invertida (única forma permitida en el caso multidimensional), por lo que se tiene la facilidad de utilizar términos de penalización que permitan violaciones mínimas al conjunto de decisiones factibles. Un término de penalización comunmente utilizado es:

$$\left.\begin{aligned} &PNTYU = 100\ [(\max\ (0, u_{ij,min} - u_{ij}))/DELU]^2 \\ &\text{ó} \\ &PNTYU = 100\ [(\max\ (0, u_{ij} - u_{ij,max}))/DELU]^2 \end{aligned}\right\}$$

El uso de los términos de penalización permite generar, en principio, estados no factibles los cuales se espera convergan a estados factibles. Existe el peligro de que no haya convergencia si las trayectorias iniciales son extremadamente malas.

En la versión actual del código únicamente puede usarse la forma invertida y una dimensión máxima de cinco para el vector de estados. También se debe recalcar que las dimensiones del vector de decisión y del vector de estado deben ser iguales. Si para ciertos problemas existiera disparidad, ésta se puede remediar fácilmente añadiendo variables "falsas" (dummy) de estados o decisiones, según sea lo apropiado.

3.3 SUBRUTINAS QUE PROPORCIONA EL USUARIO

Al explicar el funcionamiento y la estructura de la programación dinámica, se han identificado ciertos elementos comunes a todos los problemas, suceptibles de ser solucionados por medio de esta técnica, a saber: las etapas, los estados, las decisiones, la función de utilidad, la función de transición y la función recursiva.

Con excepción de las funciones de utilidad y transición, las características de los demás elementos (como son: número de etapas, intervalos para las variables de estado y de decisión, y tipo de función recursiva) se proporcionan por medio de un archivo de datos, el cual se construye por medio del programa ejecutable SETUP, que va pidiendo los distintos datos y los formatea de manera adecuada para que puedan ser entendidos por el CSUDP.

Como es de esperarse, las funciones de transición y utilidad pueden variar, en mucho, para cada problema en particular, dado que, en la programación dinámica estas funciones no requieren de una estructura definida. Por lo tanto no son suceptibles de ser construidas por medio de datos o, en caso de ser posible, podría resultar un proceso muy arduo. Por tal motivo estas funciones se suministran directamente por medio de subrutinas codificadas en fortran 77.

SUBRUTINA STATE

El propósito de esta subrutina es suministrar la Función de Transición de Estados

$$x_{i+1} = t_i(x_i, u_i) \qquad \text{forma estandar}$$

ó

$$u_i = t_i^{-1}(x_i, x_{i+1}) \qquad \text{forma invertida}$$

donde las siguientes variables en FORTRAN se reservan para el uso del código del CSUDP y que representan lo que a continuación se lista:

I = etapa actual.

X = x_i estado de la etapa actual.

X1 = x_{i+1} estado de la etapa siguiente.

U = u_i decisión de la etapa actual.

J = entero para la tabulación de los valores discretos de x_i (*e.i.*, J=1 corresponde a $x_{i,min}$; J=2 a $x_{i,min}$+DELX, etc.)

K = entero para la tabulación de los valores discretos de x_{i+1} (*e.i.*, K=1 corresponde a $x_{i+1,min}$; J=2 a $x_{i+1,min}$+DELX, etc.)

L = entero para la tabulación de los valores discretos de u_i (*e.i.*, L=1 corresponde a $u_{i,min}$; L=2 a $u_{i,min}$+DELU, etc.)

Todas las variables anteriores sólo pueden tener el uso definido arriba. Para esta subrutina y la subrutina OBJECT, en los problemas unidimensionales debe incluirse la siguiente etiqueta COMMON:

```
COMMON/ONEDEM/X,X1,U,F,I,J,K,L,R,PNALTY
```

Variables de entrada y de salida de la subrutina STATE son:

- Para el caso no invertido:

  variables de entrada = I, J, X, L y U

  variable de salida = X1 ;

- Para el caso invertido:

  variables de entrada = I, J, K, X y X1

  variable de salida = U

Note que en los problemas unidimensionales cualquier variable aleatoria que deba aparecer en la subrutina STATE y/o en la subrutina OBJECT se introduce a las subrutinas vía la etiqueta COMMON/ONEDM.

Para los problemas multidimensionales debe incluirse la siguiente etiqueta COMMON:

```
COMMON/MULTDM/X(5),X1(5),U(5),F,I,J(5),K(5),L(5),PNALTY
```

Para resolver problemas hacia adelante, la etapa actual se define como las etapas que faltan por recorrer, de tal manera que:

$$I = N - II + 1$$

donde II es la etapa actual. Por lo tanto el estado de entrada X es ahora x_{i+1}, el estado de salida X1 es x_i, y asimismo la J y la K se cambian en la definición.

SUBRUTINA OBJECT

El propósito de la subrutina OBJECT es suministrar al CSUDP la función de utilidad $f_i(x_i,u_i)$, para esto se utiliza la variable F, que el paquete reconoce como el valor de la función de utilidad de la etapa actual. El parámetro F es el parámetro de salida de esta subrutina.

Inmediatamente después de la subrutina STATE se hace una llamada a la subrutina OBJECT únicamente si U para el caso invertido o X1 para el caso no invertido son factibles. Los presentes valores se pasan a la subrutina OBJECT para que calcule la función objetivo. También están disponibles para esta subrutina los valores enteros J,K y L, así como otros cálculos de la subrutina STATE, ya que, ésta es la solicitada en primer lugar. Lo anterior permite que se tabulen los datos de las subrutinas STATE y OBJECT, si así se desea.

SUBRUTINA READIN

De ser necesario proporcionar datos adicionales a las subrutinas STATE y OBJECT, por ejemplo: valores tabulados de la función de utilidad; es posible hacerlo con la subrutina READIN.

Los datos suministrados con esta opción pueden ser leídos de un archivo

de datos ya existente, o bien, concatenar éstos al final del archivo de datos del CSUDP (creados con el programa SETUP), aunque el usuario deberá formatearlos. Si este fuera el caso, los datos seran leidos de la unidad de archivo 5.

Los datos leidos con esta subrutina pueden pasarse a las subrutinas STATE y OBJECT via etiquetas, con bloque COMMON, creadas por el usuario. De cualquier forma, aun cuando no se requieran de datos adicionales, siempre se deberá declarar la subrutina READIN. Si éste fuera el caso, es posible construir la subrutina con declaraciones no ejecutables, por ejemplo

```
SUBROUTINE READIN
RETURN
END
```

3.4 PROBLEMAS ESTOCASTICOS Y EXTENSIONES

En la presente versión del código, únicamente pueden ser resueltos problemas estocásticos unidimensionales. Para esta clase de problemas, la función objetivo sólo puede ser del tipo 1 y 3, es decir:

$$\text{Tipo 1} \quad \min_{u_i} \text{(o Max)}\ E\left[\sum_{i=1}^{N} f_i(x_i, u_i, R_i)\right] \tag{3.12}$$

$$\text{Tipo 3} \quad \min_{u_i} \text{(o Max)}\ E\left[\prod_{i=1}^{N} f_i(x_i, u_i, R_i)\right] \tag{3.13}$$

donde E [·] representa la esperanza y R_i es una variable aleatoria discreta con función de distribución conocida, la cuál puede ser independiente, o bien, estar condicionada a los resultados de la etapa anterior (cadenas de Markov). Como comentario se hace notar que en problemas estocásticos el óptimo no está garantizado cuando se utilizan funciones objetivo del tipo 2 (Nemhauser, 1970), ver ecuación (3.2).

Aquí también es posible utilizar la forma no invertida de la función de transición ($u_i = t_i^{-1}(x_i, x_{i+1}, R_i)$, $(i=1,2,..,N)$). Lo interesante en éste último planteamiento es observar que la variable de control u_i se considera, en este caso, como una variable aleatoria en vez de x_{i+1}, ya que sobre ésta última se realiza el proceso de optimización, lo que supondría que el factor aleatorio es perfectamente pronosticable en la etapa i, o bien, que es un fenómeno distribuido a lo largo de la etapa i pero considerado de manera abultada, por ejemplo: en el problema de operación del vaso de una presa, si se suministra el nivel (óptimo) $x^*_{i+1}(x_i)$ (x^*_{i+1} como función del nivel x_i inicial) que debe tener la presa al final del mes, entonces el operador puede ajustar las entregas diarias, basándose en las entradas al vaso observadas a diario, para obtener así un nivel lo más cercano posible a x^*_{i+1}, al principio del siguiente mes.

Normalmente para los problemas estocásticos no es posible dar una política de decisión explícita como resultado del proceso de rastreo hacia adelante, puesto que x_i es una variable aleatoria, por lo tanto deben disponerse de todas las políticas de control $u^*_i(x_i)$. Sin embargo, en la forma invertida, un tipo de rastreo hacia atrás puede aplicarse por que las $x^*_{i+1}(x_i)$ óptimas son almacenadas, ya que aquí los elementos aleatorios son las desiciones u_i. La función recursiva para la forma no invertida es:

$$J_i(x_i) = \underset{u_i}{\text{Max(o Min)}} \left\{ \sum_{k=1}^{q_i} p_i(R_{ik}=r_{ik}) \left[\phi \left(f_i(x_i, u_i, r_{ik}), J_{i+1}(x_{i+1,k}) \right) \right] \right\}$$

s.a.

$$x_{i+1,k} = t_i(x_i, u_i, r_{ik})$$

donde la función $\phi(\cdot)$ corresponde a uno de los tipos de relaciones dadas en (3.12) y (3.13). El subindice "k" en $x_{i+1,k}$ significa que x_{i+1} también es una variable aleatoria. Para la forma invertida tenemos:

$$J_i(x_i) = \underset{x_{i+1}}{\text{Max(o Min)}} \left\{ \sum_{k=1}^{q_i} p_i(R_{ik}=r_{ik}) \left[\phi \left(f_i(x_i, u_{ik}, r_{ik}), J_{i+1}(x_{i+1}) \right) \right] \right\}$$

s.a.

$$u_{ik} = t_i^{-1}(x_i, x_{i+1}, r_{ik})$$

en este caso la decisión se considera ahora aleatoria. En ambos casos, las probabilidades, $p_i(R_{ik}=r_{ik})$, son independientes para cada posible valor de R_{ik}, donde

$$\sum_{k=1}^{q_i} p_i(R_{ik}=r_{ik})=1 \qquad \text{para } i=1,2,\ldots,N$$

EL CSUDP puede aceptar también probabilidades condicionadas a la variable aleatoria anterior:

$$J_i(x_i,r_{i-1})=\underset{u_i(x_{i+1})}{\text{Max(o Min)}}\Big\{\sum_{k=1}^{q_i} p_i(R_{ik}=r_{ik}|R_{i-1}=r_{i-1})\left[\phi(f_i(x_i,u_i,r_{ik}),\right.$$

$$\left.,J_{i+1}(x_{i+1},r_{ik}))\right]\Big\}$$

donde

$$\sum_{k=1}^{q_i} p_i(R_{ik}=r_{ik}|R_{i-1}=r_{i-1}) = 1, \text{ para cada valor } r_{i-1} \; ; \; i=1..N$$

ANALISIS DE RIESGO

Una característica útil del programa para los problemas estocásticos es su capacidad para incluir riesgos condicionados (es decir, condicionados al estado actual x_i del sistema). Con estos riesgos las restricciones pueden violarse con cierta probabilidad sin que el CSUDP declare una solución no factible. Siempre y cuando esta probabilidad no rebase una probabilidad máxima, especificada por el usuario.

En la forma no invertida o estándar el usuario especifica los parámetros P_{min} (P_L) y P_{max} (P_H) de manera que las x_i y las u_i dadas se consideran factibles mientras P_L y P_H cumplan con lo siguiente:

$$\sum_{k\in L_i} P_i(R_{ik}=r_{ik}) \geq P_L, \; i=1,\ldots,N \qquad (3.14)$$

$$\sum_{k\in H_i} P_i(R_{ik}=r_{ik}) \leq P_H, \; i=1,\ldots,N \qquad (3.15)$$

donde

$$L_i = \{\, k \mid k=1,\ldots,q_i;\ x_{i+1,k} = t_i(x_i,u_i,r_{ik}) < x_{i,min} \,\}$$

$$H_i = \{\, k \mid k=1,\ldots,q_i;\ x_{i+1,k} = t_i(x_i,u_i,r_{ik}) > x_{i,max} \,\}$$

para toda $i=1,\ldots,N$

Esto es, se permiten los niveles de riesgo P_L para violar la cota inferior en el estado x_{i+1} y P_H para violar la cota superior en el mismo estado x_{i+1}. Obviamente, si $P_L = P_H = 0$, las cotas superior e inferior deben satisfacerse para todos los posibles valores de R_{ik}. Para la forma invertida, tenemos:

$$L_i = \{\, k \mid k=1,\ldots,q_i;\ u_{ik} = t_i^{-1}(x_i,x_{i+1},r_{ik}) < u_{i,min} \,\}$$

$$H_i = \{\, k \mid k=1,\ldots,q_i;\ u_{ik} = t_i^{-1}(x_i,x_{i+1},r_{ik}) > u_{i,max} \,\}$$

para toda $i=1,\ldots,N$

La versión actual del código trunca las políticas u_i (forma invertida) o los estados buscados x_{i+1} (forma no invertida) hacia las cotas violadas siempre y cuando (3.14) y (3.15) se satisfagan; y la función objetivo se calcula con esos valores truncados. Probabilidades condicionales también pueden considerarse en (3.14) y (3.15).

CAPITULO 4

PROBLEMAS DE APLICACION

En este capitulo se presentan algunas aplicaciones de la programación dinámica. Como se ha mencionado, éste método de optimización es muy poderoso, sobre todo, en problemas donde se requiera ejercer un control sobre las etapas del sistema. En este capitulo se presentan cuatro problemas, a saber: el problema de reemplazo, un problema de la planeación de la producción, un problema de asignación de recursos y un problema de inventario estocástico. En los ejemplos de estos problemas, la solución se presenta de dos formas: una efectuada a mano, cuyos cálculos se encuentran resumidos en tablas; y la otra forma es la solución obtenida por medio del paquete CSUDP. Con excepción del problema de inventario, cuya solución se presenta, únicamente, por medio del programa CSUDP.

4.1 EL PROBLEMA DE REEMPLAZO

En las áreas de producción de bienes y servicios son determinantes los medios materiales, con los cuales estas industrias realizan su objetivo. Las principales caraterísticas de un bien material, que representan un interés económico son: el valor del bien al momento de comprarlo o costo de adquisición; los recursos empleados para mantener en buenas condiciones el bien o costo de mantenimiento; el tiempo máximo estimado de productividad o vida útil del bien; la paulatina pérdida de valor, que el bien sufre, debido al desgaste por el uso y paso del tiempo.

Si un bien material es imprescindible para el funcionamiento de una empresa, éste debe ser reemplazado al final de su vida útil o antes, con el fin de tener una continuidad en la producción o servicio prestado. Esta sustitución representa un costo, por lo que es de interés disponer de una política de reemplazo que minimice costos. En síntesis podemos decir que el problema de reemplazo consiste en encontrar el momento o tiempos, en que se debe sustituir un bien por otro de las mismas características de manera que, en un período determinado de tiempo, los costos asociados a todo el proceso sean mínimos.

El modelo básico de reemplazo, a considerar, consiste en: revisar periódicamente una máquina para repararla o reemplazarla por otra. El costo de reparación depende del tiempo que la máquina lleva produciendo, o bien, si se decide desecharla comprar una nueva. Se desea tener una política de reemplazo que minimice los costos para los próximos n períodos, al final de los cuales la máquina que esté en operación es desechada. Teniendo en cuenta que una máquina tiene como máximo una vida útil de m períodos. Podemos proceder a definir algunos parámetros y variables de decisión como sigue: sea ca := costo de adquisición de una máquina nueva; $co(x)$:= costo de mantenimiento de una máquina con x períodos de uso; $vr(x)$:= valor de rescate de una máquina con x períodos de uso; y u la variable de decisión

donde $$u := \begin{cases} 0 & \text{si se reemplaza} \\ 1 & \text{si se mantiene} \end{cases}$$

Entonces el problema de reemplazo se plantearía como sigue:

$$\text{Minimizar} \quad \sum_{i=1}^{n} c_i(x_i, u_i) - vr(x_{n+1}) \qquad (4.1)$$

s.a

$$x_{i+1} = t(x_i, u_i) \qquad i = 1 \,..\, n$$

$$1 \le x_i \le m \qquad i = 1 \,..\, n+1$$

$$x_i \in Z \quad \text{y} \quad u_i \in \{0, 1\}$$

donde:

$$c_i(x_i,u_i) = \begin{cases} ca - vr(x_i) & \text{si} \quad u_i = 0 \\ co(x_i) & \text{si} \quad u_i = 1 \end{cases}$$

$$t\ (x_i,u_i) = \begin{cases} 1 & \text{si} \quad u_i = 0 \\ x_i + 1 & \text{si} \quad u_i = 1 \end{cases}$$

Aquí x_i es la edad de la máquina al inicio del intervalo de tiempo a planear. Este problema se puede resolver con programación dinámica, por lo que, procediendo como se indica en el apartado 1.4, este modelo básico planteado en la forma hacia atrás quedaría como sigue:

- Las etapas son los periodos de tiempo que faltan por recorrer, al principio de los cuales se puede efectuar el reemplazo o no.

- Los estados (x_i) representan la edad o periodos de uso de la máquina, al principio de cada etapa.

- $J_i(x_i)$ representará la política de reemplazo de costo mínimo, para los últimos i periodos, cuando la máquina tiene en ese momento una edad x_i.

Si la variable de decisión u_i es:

$$u_i = \begin{cases} 0 & \text{si se reemplaza} \\ 1 & \text{si se mantiene} \end{cases}$$

y el costo total $c_i(x_i,u_i)$ viene dado por:

$$c_i(x_i,u_i) = \begin{cases} ca - vr(x_i) & \text{si } u_i = 0 \\ co(x_i) & \text{si } u_i = 1 \end{cases}$$

entonces la ecuación recursiva es

$$J_i(x_i) = \min_{u_i} \{ c_i(x_i,u_i) + J_{i-1}(x_{i-1}) \}$$

s.a

$$x_{i-1} = (x_i)(u_i) + 1$$

$$1 \le x_i \le m, \qquad i = 1 \,..\, n$$

$$x_i \in Z \ , \ u_i \in \{0,1\}$$

Ejemplo 1: una máquina en una planta de producción es inspeccionada anualmente y se le da mantenimiento o bien es reemplazada por una máquina nueva. El costo de reparación y el valor de rescate de la máquina según su antigüedad se muestran en la siguiente tabla:

datos problema de reemplazo (en miles)

	EDAD (años)	1	2	3	4
(CO)	COSTO MANTENIMIENTO	7	3	9	-
(VR)	VALOR DE RESCATE	10	5	2	0

El costo de una máquina nueva es de $20,000. El tiempo que resta de vida a toda la planta son cinco años, al final de los cuales la máquina que esté funcionando es desechada. Determine un plan de reemplazo a costo mínimo si:

a) la máquina cumple un año en su próximo aniversario
b) la máquina cumple dos años en su próximo aniversario
c) la máquina cumple tres años en su próximo aniversario
d) la máquina cumple cuatro años en su próximo aniversario

Solución:
Estar en la etapa i significa que faltan i años por recorrer; estar en el estado x_i de la etapa i, significa que la máquina tiene la edad x_i al inicio del año $(n+1) - i$; $J_i(x_i)$ representa el costo (mínimo) de una política óptima de reemplazo para los próximos i años, de una máquina de edad x_i. La decisión a tomar en la etapa i es:

$$u_i = \begin{cases} 0 & \text{reemplazar} \\ 1 & \text{mantener} \end{cases}$$

El costo asociado con el presente estado y acción es:

$$c_i(x_i, u_i) = \begin{cases} 20 - VR(x_i) & \text{si } u_i = 0 \\ CO(x_i) & \text{si } u_i = 1 \end{cases}$$

la ecuación de recurrencia es

$$J_i(x_i) = \min_{u_i} \{ c_i(x_i, u_i) + J_{i-1}(x_{i-1}) \}$$

s.a.

$$x_{i-1} = (x_i)(u_i) + 1 \qquad i = 1 \ldots 5$$

$$1 \leq x_i \leq 4 \qquad i = 0 \ldots 5$$

la condición de frontera es $J_0(x_1) = -VR(x_1)$. En base a esta notación estamos buscando: a) $J_5(1)$; b) $J_5(2)$; c) $J_5(3)$ y d) $J_5(4)$.

Los cálculos resumidos se muestran en las tablas siguientes:

	u_1			
x_1	0	1	$J_1(x_1)$	u_1^*
1	0	2	0	0
2	5	1	1	1
3	8	9	8	0
4	10	-	10	0

	u_2			
x_2	0	1	$J_2(x_2)$	u_2^*
1	10	8	8	1
2	15	11	11	1
3	18	19	18	0
4	20	--	20	0

	u_3			
x_3	0	1	$J_3(x_3)$	u_3^*
1	18	18	18	0,1
2	23	21	21	1
3	26	29	26	0
4	28	--	28	0

	u_4			
x_4	0	1	$J_4(x_4)$	u_4^*
1	28	28	28	0,1
2	33	29	29	1
3	36	37	36	0
4	38	--	38	0

	u_5			
x_5	0	1	$J_5(x_5)$	u_5^*
1	38	36	36	1
2	43	39	39	1
3	46	47	46	0
4	48	--	48	0

Procediendo como se indica en el apartado 1.4, tenemos que la solución óptima esta dada por:

La solución óptima para el inciso: a) es

año	acción	costo
1	mantener	7
2	mantener	3
3	reemplazar	18
4	mantener	7
5	mantener	3
6	rescate	-2
		36

para el inciso b) hay dos soluciones una de ellas es:

año	acción	costo
1	mantener	3
2	reemplazar	18
3	reemplazar	10
4	mantener	7
5	mantener	3
6	rescate	-2
		39

para los incisos c) y d) hay tres politicas de decisión, que son las mismas, una de ellas es:

año	acción	c) costo	d) costo
1	reemplazar	18	20
2	reemplazar	10	10
3	reemplazar	10	10
4	mantener	7	7
5	mantener	3	7
6	rescate	-2	-2
		46	48

Solución por medio del paquete CSUDP:

Las subrutinas STATE, OBJECT y READIN, que proporciona el usuario, se compilan en un archivo de fortran77, para este ejemplo tenemos:

(REMP.FOR: nombre del archivo en este ejemplo)

```
C **   PROBLEMA DE REEMPLAZO
C **
       SUBROUTINE STATE
       COMMON/ONEDM/X,X1,U,F,I,J,K,L,R,PNALTY
C **   FORMA NO INVERTIDA
       X1 = X * U + 1.
       RETURN
       END
C **
       SUBROUTINE OBJECT
       COMMON/ONEDM/X,X1,U,F,I,J,K,L,R,PNALTY
       DIMENSION VR(4),CO(4)
       DATA VR/10.,5.,2.,0./
       DATA CO/7.,3.,9.,100./
       IX = IFIX(X)
       IF (U.EQ.0.)  F = 20. - VR(IX)
       IF (U.EQ.1.)  F = CO(IX)
       IF (I.EQ.5)   F = F - VR(IX+1)
       RETURN
       END
C **
       SUBROUTINE READIN
       RETURN
       END
```

Una vez compiladas, sin errores, estas subrutinas se ligan con el programa objeto DYNPRO (REMP+DYNPRO), para obtener el programa ejecutable (REMP.EXE). Antes de ejecutar este programa, se debe crear por medio del programa SETUP.EXE, un archivo de datos donde se especifiquen las caracteristicas de la optimización. (Recuerde que los datos adicionales de un problema, como pueden ser costos etc., se pueden concatenar, debidamente formateados, al final del archivo de datos creado por SETUP y éstos se leen con el dispositivo de entrada número 5; o bien , los datos pueden ser leidos de un archivo ya existente de datos; o bien, como en este ejemplo, definirlos directamente en las subrutinas, lo que generalmente no es aconsejable, ya que,si cambiara alguno de los datos del problema habría que complar y ligar nuevamente).

(ejemplo de como funciona el programa SETUP)

```
**************************************************
* GENERALIZED DYNAMIC PROGRAMMING FOR MICROCOMPUTER *
*                  PROGRAM CSUDP                 *
*              COLORADO STATE UNIVERSITY         *
*                  JOHN W. LABADIE               *
**************************************************

ENTER 0: TO CREATE NEW DATA FILE
ENTER 1: TO READ AND EDIT AN EXISTING DATA FILE
0

TYPE IN DATA FILE NAME : REMP.DAT     (nombre del archivo de datos
                                       que estamos creando)

ENTER TITLE FOR THIS PROBLEM (UP TO 70 CHARACTERS):
Problema de Reemplazo
IF MINIMIZATION PROBLEM, ENTER 1
IF MAXIMIZATION PROBLEM, ENTER -1:
1
ENTER 1: IF TYPE 1 OBJECTIVE (ADDITIVE)
ENTER 2: IF TYPE 2 OBJECTIVE (MAX(MIN) OR MIN(MAX))
ENTER 3: IF TYPE 3 OBJECTIVE (MULTIPLICATIVE):
1
ENTER DIMENSION FOR STATE VECTOR:
1
ENTER 0: IF STATE EQUATION IN NONINVERTED FORM
ENTER 1: IF STATE EQUATION IN INVERTED FORM
0
IS THIS A STOCHASTIC PROBLEM? (Y OR N)
n
ENTER NUMBER OF STAGES:
5
```

```
PRINTOUT OPTIONS FOR OPTIMAL POLICIES:
ENTER 1: FOR OPTIMAL OPEN LOOP POLICIES ONLY
ENTER 2: FOR OPTIMAL FEEDBACK POLICIES ONLY
ENTER 3: FOR BOTH OPTIONS
3
ENTER 0: IF SELECTING FIRST TIE IN OPTIMIZATION
ENTER 1: IF SELECTING LAST  TIE IN OPTIMIZATION
0
ENTER DELU: DISCRETIZATION INTERVAL FOR DECISION(S) U:
1.
DO YOU WANT SPLICING IN THE STATE SAPCE? (Y OR N):
n
ENTER STATE DISCRETIZATION INTERVAL DELX:
1.
UPPER AND LOWER BOUNDS ON STATE AND DECISION VARIABLES

FOR  STATE   DIMENSION  NO.            1...
   IN HOW MANY STAGES DO THE LOWER BOUNDS
   ON THE STATE    VARIABLE CHANGE? (BE SURE TO
   ALWAYS INCLUDE THE 1ST STAGE IN YOUR COUNT: 1
   ENTER STAGE NO. AND CORRESPONDING BOUND:  1 1
FOR  STATE   DIMENSION  NO.            1...
   IN HOW MANY STAGES DO THE UPPER BOUNDS
   ON THE STATE    VARIABLE CHANGE? (BE SURE TO
   ALWAYS INCLUDE THE 1ST STAGE IN YOUR COUNT: 1
   ENTER STAGE NO. AND CORRESPONDING BOUND:  1 4
FOR DECISION DIMENSION NO.             1...
   IN HOW MANY STAGES DO THE LOWER BOUNDS
   ON THE DECISION VARIABLE CHANGE? (BE SURE TO
   ALWAYS INCLUDE THE 1ST STAGE IN YOUR COUNT: 1
   ENTER STAGE NO. AND CORRESPONDING BOUND:  1 0
FOR DECISION DIMENSION NO.             1...
   IN HOW MANY STAGES DO THE UPPER BOUNDS
   ON THE DECISION VARIABLE CHANGE? (BE SURE TO
   ALWAYS INCLUDE THE 1ST STAGE IN YOUR COUNT: 1
   ENTER STAGE NO. AND CORRESPONDING BOUND:  1 1

ENTER:    FOR:
  0       TERMINATING DATA EDITING.
  1       NEW PROBLEM TITLE.
  2       MINIMIZING OR MAXIMIZING OBJECTIVE.
  3       CHANGING TYPE OF OBJECTIVE FUNCTION.
  4       INVERTED 0 NONINVERTED STATE EQUATION.
  5       STOCHASTIC OR DETERMINISTIC PROBLEM.
  6       CHANGING NO. OF STAGES.
  7       CHANGING NO. OF ITERATIONS (MULTIDIM. ONLY).
  8       DIFFERENT PRINTOUT OPTION.
  9       CHANGING THE TIE BREAKING OPTION.
 10       NEW DECISION SPACE INCREMENT DELU.
 11       NEW STATE SPACE INCREMENT DELX.
 12       ALTERING BOUNDS ON VARIABLES.
 13       NEW INITIAL STATE TRAJECTORY (MULTIDIM. ONLY).
 14       NEW PROBABILITY DISTRIBUTIONS (STOCH. ONLY).
0   Stop - Program terminated.
```

En seguida se despliega la solución del problema en los dos formatos disponibles de impresión, que son: políticas abiertas, equivalentes a la parte derecha de las tablas, hechas a mano, donde se resumen los cálculos; y políticas retroalimentadas, equivalentes a la solución cuando se ha efectuado el rastreo de regreso de las etapas.

Title : Problema de Reemplazo

```
**************************************
*                                    *
*        1 Dimensional Problem       *
*                                    *
*        Minimization Problem        *
*                                    *
*     Objective is Summation Type    *
*                                    *
*     Deterministic Optimization     *
*                                    *
*   Problem Assumed Non-Invertible   *
*                                    *
*        First Tie Value Taken       *
*                                    *
*       Splicing will not Occur      *
*                                    *
*        Number of Stages =  5       *
*                                    *
**************************************
```

Interval for X = 1.000
Interval for U = 1.000

Upper and Lower Bounds on X(I+1) and U(I)

I	XMIN(I)	XMAX(I)	UMIN(I)	UMAX(I)
1	1.000	4.000	.0000	1.000
2	1.000	4.000	.0000	1.000
3	1.000	4.000	.0000	1.000
4	1.000	4.000	.0000	1.000
5	1.000	4.000	.0000	1.000
6	1.000	4.000		

Optimal Policies

```
************
*          *
* Stage  5 *
*          *
************
```

X(5)	1.0000	2.0000	3.0000	4.0000
U(X(5)	1.0000	1.0000	1.0000	.00000
X*(6)	2.0000	3.0000	4.0000	1.0000
F(X(5)	2.0000	1.0000	9.0000	13.000

Optimal Policies

```
************
*          *
* Stage  4 *
*          *
************
```

X(4)	1.0000	2.0000	3.0000	4.0000
U(X(4)	1.0000	1.0000	.00000	.00000
X*(5)	2.0000	3.0000	1.0000	1.0000
F(X(4)	8.0000	12.000	20.000	22.000

Optimal Policies

```
************
*          *
* Stage  3 *
*          *
************
```

X(3)	1.0000	2.0000	3.0000	4.0000
U(X(3)	.00000	.00000	.00000	.00000
X*(4)	1.0000	1.0000	1.0000	1.0000
F(X(3)	18.000	23.000	26.000	28.000

Optimal Policies

```
************
*          *
* Stage  2 *
*          *
************
```

X(2)	1.0000	2.0000	3.0000	4.0000
U(X(2)	.00000	1.0000	.00000	.00000
X*(3)	1.0000	3.0000	1.0000	1.0000
F(X(2)	28.000	29.000	36.000	38.000

Optimal Policies

```
************
*          *
* Stage  1 *
*          *
************
```

X(1)	1.0000	2.0000	3.0000	4.0000
U(X(1)	1.0000	1.0000	.00000	.00000
X*(2)	2.0000	3.0000	1.0000	1.0000
F(X(1)	36.000	39.000	46.000	48.000

OPTIMAL SOLUTION FOR X(1) = 1.00000

I	X*	U*	F*
1	1.000000	1.000000	7.000000
2	2.000000	1.000000	3.000000
3	3.000000	.0000000	18.00000
4	1.000000	1.000000	7.000000
5	2.000000	1.000000	1.000000
6	3.000000		

MINIMUM INTERPOLATED OBJECTIVE VALUE = 36.00000
MINIMUM OBJECTIVE VALUE = 36.00000

OPTIMAL SOLUTION FOR X(1) = 2.00000

I	X*	U*	F*
1	2.000000	1.000000	3.000000
2	3.000000	.0000000	18.00000
3	1.000000	.0000000	10.00000
4	1.000000	1.000000	7.000000
5	2.000000	1.000000	1.000000
6	3.000000		

MINIMUM INTERPOLATED OBJECTIVE VALUE = 39.00000
MINIMUM OBJECTIVE VALUE = 39.00000

OPTIMAL SOLUTION FOR X(1) = 3.00000

I	X*	U*	F*
1	3.000000	.0000000	18.00000
2	1.000000	.0000000	10.00000
3	1.000000	.0000000	10.00000
4	1.000000	1.000000	7.000000
5	2.000000	1.000000	1.000000
6	3.000000		

MINIMUM INTERPOLATED OBJECTIVE VALUE = 46.00000
MINIMUM OBJECTIVE VALUE = 46.00000

```
OPTIMAL SOLUTION FOR X(1) = 4.00000

I       X*              U*              F*

1     4.000000        .0000000        20.00000
2     1.000000        .0000000        10.00000
3     1.000000        .0000000        10.00000
4     1.000000        1.000000        7.000000
5     2.000000        1.000000        1.000000
6     3.000000

MINIMUM INTERPOLATED OBJECTIVE VALUE =    48.00000
MINIMUM OBJECTIVE VALUE =    48.00000
```

4.2 UN PROBLEMA DE LA PLANEACION DE LA PRODUCCION

Para cualquier industria, en particular la de producción de bienes, una condición necesaria para su subsistencia es que sea rentable. Además si se trata de una empresa con fines lucrativos, las ganancias obtenidas por ésta deben ser atractivas para sus dueños y para ello se necesita de una organización eficiente.

La cantidad a producir de un bien determinado depende de varios factores tales como la capacidad de producción instalada, materia prima disponible, capacidad de almacenaje, la demanda esperada del bien, etc.. Si se estiman o conocen con anticipación los principales factores a que nos referimos se puede diseñar un plan de producción (que además sirva de control) que repercuta positivamente en el objetivo económico de la empresa.

En base a lo anterior diremos que el problema de la planeación de la producción consiste en determinar, para un intervalo de tiempo, la cantidad de bienes que se han de producir bajo ciertas condiciones materiales y de mercado, de tal manera que se minimicen costos o se maximicen ganancias.

El modelo básico determinístico consiste en planear la producción de un artículo para un horizonte de tiempo de n periodos, donde la demanda es conocida en cada periodo. La demanda puede ser satisfecha únicamente por

la producción del periodo actual o por productos que hayan sido fabricados en periodos anteriores y que estén almacenados. Suponemos que el modelo incluye costos de producción y de almacenaje y que el nivel de inventario debe ser cero al principio y al final del horizonte. El objetivo es llevar un control de la producción de tal manera que se satisfaga la demanda a un costo mínimo.

Para este problema definiremos a: d_i como la demanda en el periodo i; $cp_i(u)$ el costo de producir u unidades durante el periodo i; y $ca_i(x)$ como el costo de tener almacenadas x unidades al inicio del periodo i. Entonces el problema consiste en:

$$\text{Minimizar} \quad \sum_{i=1}^{n} (cp_i(u_i) + ca_i(x_i))$$

s.a.

$$x_1 = x_{n+1} = 0$$

$$x_{i+1} = x_i + u_i - d_i \qquad i = 1 \,..\, n$$

$$u_i \geq 0 \quad y \quad u_i \in Z \qquad i = 1 \,..\, n$$

En programación dinámica, utilizando la forma hacia atrás pero con las etapas indexadas hacia adelante, el problema se plantearía de la manera siguiente: estar en la etapa i, significa estar en el periodo i; estar en el estado x_i de la etapa i, significa tener un nivel de inventario x_i al inicio del periodo i; $J_i(x_i)$ es el costo mínimo de un programa de producción para los últimos (n+1) - i periodos, a partir de un nivel de inventario de x_i; y la variable de decisión u_i es el número de artículos que se producen en la etapa i. Así tenemos que la ecuación recursiva viene dada por:

$$J_i(x_i) = \text{Min} \left\{ cp_i(u_i) + ca_i(x_i) + J_{i+1}(x_{i+1}) \right\}$$

s.a.

$$x_{i+1} = x_i + u_i - d_i \qquad i = 1..n$$

$$x_1 = x_{n+1} = 0$$

$$u_i \geq 0 \text{ y } u_i \in Z$$

con una CONDICION DE FRONTERA de $J_{n+1}(0) = 0$.

Ejemplo 2: un constructor de yates tiene los pedidos mensuales que se muestran en la tabla siguiente:

	ORDENES DE YATES					
	(1)	(2)	(3)	(4)	(5)	(6)
MES	FEB	MAR	ABR	MAY	JUN	JUL
No. de YATES	1	2	5	3	2	1

El fabricante puede construir hasta cuatro yates en cualquier mes y puede almacenar a lo más tres. Si se construye al menos un bote en un mes, se tiene un sobre costo de $40,000.00 más $100,000.00 por cada yate construido. El costo de almacenamiento por unidad es de $10,000.00. Suponga que todas las órdenes deben ser satisfechas y que el número de unidades almacenadas es cero al principio y al final del periódo. ¿En que meses y en que cantidades los botes se deben construir si los costos deben ser minimizados?.

Solución:

Estar en la *etapa* t significa estar en el mes t. Mientras que estar en el *estado* x_t de la etapa t significa que se tiene un nivel de inventario de x_t yates, al inicio del mes t. Por otra parte, $J_t(x_t)$ es el costo (mínimo) de una política óptima de producción, para los últimos $7-t$ meses, cuando se tiene un inventario de x_t yates en el mes t.

sean: u_t el número de yates que se producen en el mes t; d_t la demanda en el mes t; $c_t(x_t,u_t)$ el costo total de tener un inventario de x_t yates, al inicio del mes t, y de producir una cantidad de u_t en el mismo mes. Entonces

$$c_t(x_t,u_t) = \begin{cases} 10(x_t) & \text{si } u_t = 0 \\ 10(x_t)+100(u_t)+40 & \text{si } u_t > 0 \end{cases} \quad \text{(cifras en miles)}$$

La ecuación recursiva viene dada por

$$J_t(x_t) = \min_{u_t} \{ c_t(x_t,u_t) + J_{t+1}(x_{t+1}) \}$$

s.a.

$$x_{t+1} = x_t + u_t - d_t \qquad t = 1 \ldots 6$$

$$0 \le x_t \le 3, \qquad t = 2 \ldots 6$$

$$x_1 = x_7 = 0; \quad 0 \le u_t \le 4; \quad x_t, u_t \in Z; \; t=1..6$$

En términos de ésta notación estamos buscando $J_1(0)$ y la condición de frontera es $J_7(0) = 0$

Los cálculos se muestran en las siguientes tablas:

x_6	u_6= 0	1	2	3	4	$J_6(x_6)$	u_6^*
0	-	140	-	-	-	140	1
1	0	-	-	-	-	10	0 *

x_5	u_5= 0	1	2	3	4	$J_5(x_5)$	u_5^*
0	-	-	380	350	-	350	3 *
1	-	290	260	-	-	260	2
2	160	170	-	-	-	160	0
3	40	-	-	-	-	40	0

x_4	u_4= 0	1	2	3	4	$J_4(x_4)$	u_4^*
0	-	-	-	690	700	690	3 *
1	-	-	600	610	610	600	2
2	-	510	520	520	500	500	4
3	380	430	430	410	-	380	0

x_3	u_3= 0	1	2	3	4	$J_3(x_3)$	u_3^*
1	-	-	-	-	1140	1140	4 *
2	-	-	-	1050	1060	1050	3
3	-	-	960	970	970	960	2

x_2	u_2= 0	1	2	3	4	$J_2(x_2)$	u_2^*
0	-	-	-	1480	1490	1480	3
1	-	-	1390	1400	1410	1390	2
2	-	1300	1310	1320	-	1300	1
3	1170	1220	1230	-	-	380	0 *

x_1	u_1= 0	1	2	3	4	$J_1(x_1)$	u_1^*
0	-	1620	1630	1640	1610	1610	4 *

la solución óptima es:

mes	yates por fabricar	C O S T O INV.	PROD.	TOTAL
febrero	4	0	440	440
marzo	0	30	0	30
abril	4	10	440	450
mayo	3	0	340	340
junio	3	0	340	340
julio	0	10	0	10
				1610

Solución por medio del paquete CSUDP:

```
C **  TEMA: PLANECION DE LA PRODUCCION
C **
      SUBROUTINE STATE
      COMMON/ONEDM/X,X1,U,F,I,J,K,L,R,PNALTY
      DIMENSION D(6)
      DATA D/1.,2.,5.,3.,2.,1./
C **  D(I) ES LA DEMANDA EN EL MES I (I=1,..,6)
      X1 = X + U - D(I)
      RETURN
      END
C **
      SUBROUTINE OBJECT
      COMMON/ONEDM/X,X1,U,F,I,J,K,L,R,PNALTY
      DIMENSION D(7)
C **  LOS COSTOS ESTAN EN MILES
      IF (U.EQ.0.0) THEN
      F = 10.*X
      ELSE
      F = 100.*U + 10.*X + 40.
      ENDIF
      RETURN
      END
C **
      SUBROUTINE READIN
      RETURN
      END
```

```
 Title :  Problema de la Planeacion de la Producción

          **************************************
          *        1 Dimensional Problem       *
          *         Minimization Problem       *
          *     Objective is Summation Type    *
          *      Deterministic Optimization    *
          *   Problem Assumed Non-Invertible   *
          *        First Tie Value Taken       *
          *       Splicing will not Occur      *
          *        Number of Stages =  6       *
          **************************************
```

Interval for X = 1.000
Interval for U = 1.000

Upper and Lower Bounds on X(I+1) and U(I)

I	XMIN(I)	XMAX(I)	UMIN(I)	UMAX(I)
1	.0000	.0000	.0000	4.000
2	.0000	3.000	.0000	4.000
3	.0000	3.000	.0000	4.000
4	.0000	3.000	.0000	4.000
5	.0000	3.000	.0000	4.000
6	.0000	3.000	.0000	4.000
7	.0000	.0000		

OPTIMAL SOLUTION FOR X(1) = .000000

I	X*	U*	F*
1	.0000000	4.000000	440.0000
2	3.000000	.0000000	30.00000
3	1.000000	4.000000	450.0000
4	.0000000	3.000000	340.0000
5	.0000000	3.000000	340.0000
6	1.000000	.0000000	10.00000
7	.0000000		

MINIMUM INTERPOLATED OBJECTIVE VALUE = 1610.000
MINIMUM OBJECTIVE VALUE = 1610.000

4.3 PROBLEMA DE ASIGNACION DE RECURSOS

En el desarrollo de cualquier actividad se requiere disponer de las condiciones materiales adecuadas para poder realizarla, por ejemplo, para producir cierto tipo de articulos es necesario disponer de un local, maquinaria, mano de obra, materias primas etc.. En general, el problema de asignación de recursos consiste en distribuir los recursos escasos de tal manera que se optimice una función de utilidad asociada a éstos. En general estos recursos pueden ser
materias primas, capital, mano de obra, tiempo, espacio, capacidad etc..

En el modelo básico de asignación de recursos se tienen m recursos que se distribuyen entre n usos diferentes, donde $g_i(u_{1i}, u_{2i}, \ldots, u_{mi})$ es la utilidad obtenida por asignar una combinación determinada de los

recursos, u_{ji}, al i-ésimo uso. Se dispone únicamente de una cantidad c_j del recurso j, j = 1 ..m. De esta manera, el problema se representaría como:

$$\text{Maximizar (Minimizar)} \quad \sum_{i=1}^{n} g_i(u_{1i}, u_{2i}, \ldots, u_{mi})$$

s.a.

$$u_{j1} + u_{j2} + \ldots + u_{jn} \leq c_j \qquad j = 1 \ldots m$$

En las más de las veces la utilidad se mide a través de un producto que utiliza los recursos y no directamente por medio de éstos. Con lo que el modelo básico se reinterpretaría como: asignar m recursos a n productos, donde $g_i(\omega_i)$ es la utilidad obtenida por una cantidad ω_i del producto i; y $u_j(\omega_i)$ es la cantidad que se necesita del recurso J (J = 1 .. m) para poder tener una cantidad ω_i del producto i (i = 1 .. n). Se dispone únicamente de una cantidad c_j del producto J, J = 1..m. De esta forma, el problema se representa como:

$$\text{Maximizar} \quad \sum_{i=1}^{n} g_i(\omega_i)$$

s.a.

$$u_j(\omega_1) + u_j(\omega_2) + \ldots + u_j(\omega_n) \leq c_j \qquad j = 1 \ldots m$$

El planteamiento en programación dinámica es el siguiente:

Estar en la etapa i, significa que los recursos disponibles se distribuyen entre los primeros i usos (productos). Estar en el estado $(x_{1i}, x_{2i}, \ldots, x_{mi})$ de la etapa i, significa que se dispone de una cantidad x_{ji} del J-ésimo recurso, J = 1..m, para distribuirse entre los primeros i usos (productos). $J_i(x_{1i}, x_{2i}, \ldots, x_{mi})$ es la utilidad óptima que se puede obtener, distribuyendo las cantidades disponibles de los m recursos entre los primeros i usos (productos). La ecuación recursiva para el primer caso viene dada por:

$$J_i(x_{1i}, \ldots, x_{mi}) = \max_{u_{ji}} \left\{ g_i(x_{1i}, \ldots, x_{mi}) + J_{i-1}(x_{1(i-1)}, \ldots, x_{m(i-1)}) \right\}$$

s.a.

$$x_{j(i-1)} = x_{ji} - u_{ji} \qquad j = 1 \ldots m$$

$$x_{ji} \leq c_j \qquad j = 1 \ldots m$$

$$0 \leq u_{ji} \leq x_{ji} \qquad j = 1 \ldots m$$

para el segundo caso la ecuación recursiva es:

$$J_i(x_{1i}, \ldots, x_{mi}) = \max_{u_j} \{ g_i(\omega_i) + J_{i-1}(x_{1(i-1)}, \ldots, x_{m(i-1)}) \}$$

s.a.

$$x_{j(i-1)} = x_{ji} - u_j(\omega_i) \qquad j = 1 \ldots m$$

$$x_{ji} \leq c_j \qquad j = 1 \ldots m$$

$$0 \leq u_j(\omega_i) \leq x_{ji} \qquad j = 1 \ldots m$$

Ejemplo 3: Una compañía constructora tiene cuatro proyectos en construcción, de acuerdo con la actual distribución de la mano de obra, equipo y materiales los cuatro proyectos pueden ser términados en 15, 20, 18, y 25 semanas. El administrador quiere reducir los tiempos para terminar los proyectos y ha decidido repartir entre todos los proyectos un adicional de $35,000.00. Los tiempos de terminación como funciones del fondo adicional asignado a cada proyecto se muestran en la tabla siguiente:

FONDOS ADICIONALES (X 1000)	TIEMPO DE TERMINACION (SEMANAS) PROYECTOS			
	1	2	3	4
0	15	20	18	25
5	12	18	15	21
10	10	13	12	18
15	8	11	10	16
20	7	9	8	14
25	6	8	8	12
30	5	7	7	11
35	4	7	6	10

Supóngase que los fondos adicionales sólo pueden ser distribuidos en bloques de $5,000. ¿Como podrían ser repartidos los $35,000 pesos, de manera que los cuatro proyectos se realicen en un tiempo total mínimo?

Solución:

Estar en la etapa i, significa considerar únicamente los primeros i proyectos; y estar en el estado x_i de la etapa i, significa que se asignan x_i miles de pesos a los primeros i proyectos. Por otra parte $J_i(x_i)$ es el tiempo mínimo total de duración de los primeros i proyectos dado que se le asignan x_i miles de pesos. La variable de decisión u_i significa que asignamos u_i miles de pesos al proyecto i. $t_i(x_i, u_i)$ es el tiempo de duración del proyecto i dado que se le asignan u_i miles de pesos de un disponible de x_i. En términos de esta notación estamos buscando $J_4(35)$. La ecuación de recurrencia viene dada por

$$J_i(x_i) = \min_{u_i} \{ t_i(x_i, u_i) + J_{i-1}(x_{i-1}) \}$$

s.a.

$$x_{i-1} = x_i - u_i$$

$$0 \leq u_i \leq x_i$$

La condición de frontera es $J_0(x_i) = 0$.

Los cálculos se muestran en las tablas siguientes:

ETAPA 1

x_1	$u_1 = 0$	5	10	15	20	25	30	35	$J_1(x_1)$	u_1^*
0	15	-	-	-	-	-	-	-	15	0
5	-	12	-	-	-	-	-	-	12	5
10	-	-	10	-	-	-	-	-	10	10
15	-	-	-	8	-	-	-	-	8	15
20	-	-	-	-	7	-	-	-	7	20
25	-	-	-	-	-	6	-	-	6	25
30	-	-	-	-	-	-	5	-	5	30
35	-	-	-	-	-	-	-	4	4	35

ETAPA 2

x_2	$u_2 = 0$	5	10	15	20	25	30	35	$J_2(x_2)$	u_2^*
0	35	-	-	-	-	-	-	-	35	0
5	32	31	-	-	-	-	-	-	31	5
10	30	28	28	-	-	-	-	-	28	5, 10
15	28	26	25	26	-	-	-	-	25	10
20	27	24	23	23	24	-	-	-	23	10, 15
25	26	23	21	21	21	23	-	-	21	10, 15, 20
30	25	22	20	19	19	20	22	-	19	15, 20
35	24	21	19	18	17	18	19	22	17	20

ETAPA 3

x_3	$u_3 = 0$	5	10	15	20	25	30	35	$J_3(x_3)$	u_3^*
0	53	-	-	-	-	-	-	-	53	0
5	49	50	-	-	-	-	-	-	49	0
10	46	46	47	-	-	-	-	-	46	0, 5
15	43	43	43	45	-	-	-	-	43	0, 5, 10
20	41	40	40	41	44	-	-	-	40	5, 10
25	39	38	37	38	40	43	-	-	37	10
30	37	37	35	35	37	39	42	-	35	10, 15
35	35	34	33	33	34	36	38	41	33	10, 15

ETAPA 4

x_4	$u_4 = 0$	5	10	15	20	25	30	35	$J_4(x_4)$	u_4^*
35	58	56	55	56	57	58	60	63	55	10

la solución óptima está dada por

PROYECTO	CANTIDAD ASIGNADA
1	5,000
2	10,000
3	10,000
4	10,000

Solución por medio del paquete:

```
C **  PROBLEMA DE ASIGNACION DE RECURSOS
C **  FINANCIEROS A CUATRO PROYECTOS EN CONSTRUCCION
C **
      SUBROUTINE STATE
      COMMON/ONEDM/X,X1,U,F,I,J,K,L,R,PNALTY
      X1 = X - U
      RETURN
      END
C **
      SUBROUTINE OBJECT
      COMMON/ONEDM/X,X1,U,F,I,J,K,L,R,PNALTY
      COMMON/TRANS/XTEMP(8,4)
      F = XTEMP(L,I)
      RETURN
      END
C **
      SUBROUTINE READIN
      COMMON/TRANS/XTEMP(8,4)
      OPEN (15,FILE='DATOS.as1',STATUS='OLD',FORM='FORMATTED')
      REWIND 15
      DO 10 JJ=1,8
      READ (15,110) (XTEMP(JJ,II), II = 1,4)
  110 FORMAT(10X,5F5.1)
   10 CONTINUE
      CLOSE (15)
      RETURN
      END
```

```
**************************************
*                                    *
*        1 Dimensional Problem       *
*                                    *
*         Minimization Problem       *
*                                    *
*     Objective is Summation Type    *
*                                    *
*      Deterministic Optimization    *
*                                    *
*    Problem Assumed Non-Invertible  *
*                                    *
*        First Tie Value Taken       *
*                                    *
*       Splicing will not Occur      *
*                                    *
*        Number of Stages =  4       *
*                                    *
**************************************

        Interval for X =  5.000
        Interval for U =  5.000
```

Upper and Lower Bounds on X(I+1) and U(I)

I	XMIN(I)	XMAX(I)	UMIN(I)	UMAX(I)
1	35.00	35.00	.0000	35.00
2	.0000	35.00	.0000	35.00
3	.0000	35.00	.0000	35.00
4	.0000	35.00	.0000	35.00
5	.0000	.0000		

OPTIMAL SOLUTION FOR X(1) = 35.0000

I	X*	U*	F*
1	35.00000	5.000000	12.00000
2	30.00000	10.00000	13.00000
3	20.00000	10.00000	12.00000
4	10.00000	10.00000	18.00000
5	.0000000		

MINIMUM INTERPOLATED OBJECTIVE VALUE = 55.00000
MINIMUM OBJECTIVE VALUE = 55.00000

Ejemplo 4: una fábrica puede producir mercancías del tipo A, B y C obteniendo las ganancias que se muestran en la tabla 1. Los artículos requieren de las materias primas X e Y en las cantidades que se muestran en la tabla 2

TABLA 1
Ganancias que se obtienen por vender distintas cantidades de mercancías

	PRODUCTO		
NUMERO DE PRODUCTOS	A (1)	B (2)	C (3)
1	5	6	10
2	9	12	18
3	12	17	25

TABLA 2
requerimientos de materias primas

	P R O D U C T O S					
	A		B		C	
NUMERO DE PRODUCTOS	X	Y	X	Y	X	Y
1	2	1	1	1	1	2
2	3	1	2	2	2	3
3	3	2	3	2	2	4

se disponen únicamente de 4 unidades del material X y de 4 unidades del material Y. ¿ Cuántos artículos de cada tipo deben producirse para obtener la máxima ganancia ?

Estar en la etapa i, significa que se reparten las unidades disponibles de las materias primas X e Y entre los primeros i productos. Encontrarse en el estado $\mathbf{x} = (x_1, x_2)$ de la etapa i, significa que dispongo de x_1 unidades del material X y x_2 unidades del material Y para repartirlos entre los primeros i productos. Asimismo $J_i(\mathbf{x})$ es la ganancia máxima que se puede obtener entre los primeros i productos dado el estado $\mathbf{x}$ de la etapa i. La variable de decisión es $u = (u_1, u_2)$ donde u_1 y u_2 es la cantidad de material X e Y, respectivamente, que se utiliza para producir artículos del tipo i. Por otra parte $g_i(u)$ es la ganancia que se obtiene por asignar la combinación, u, de recursos para producir artículos del tipo i en la etapa i. En términos de esta notación estamos buscando $J_3((4,4))$. La ecuación recursiva viene dada por:

$$J_i(\mathbf{x}) = \max_{u} \{ g_i(u) + J_{i-1}(x - u) \}$$

s.a.

$$0 \le x_k \le 4, \qquad k = 1,2$$

$$0 \le u_k \le x_k, \qquad k = 1,2$$

Dado que si no se fabrican artículos no hay ganancias, entonces la condición de frontera viene dada por $J_0(\mathbf{x}) = 0$ para todo estado $\mathbf{x}$.

Antes de elaborar los cálculos vamos a resumir en una tabla los posibles valores de g_i para las distintas combinaciones de material X e Y asignadas al producto i

producto A (1)					g_i
si	$u_1 \le 1$	ó	$u_2 = 0$		0
si	$u_1 = 2$	y	$u_2 \ge 1$		5
si	$u_1 \ge 3$	y	$u_2 = 1$		9
si	$u_1 \ge 3$	y	$u_2 \ge 2$		12

producto B (2)	g_2
si $u_1 = 0$ ó $u_2 = 0$	0
si $\{u_1=1$ y $u_2\geq 1\}$ ó $\{u_1\geq 1$ y $u_2=1\}$	6
si $u_1 = 2$ y $u_2 \geq 2$	12
si $u_1 \geq 3$ y $u_2 \geq 2$	17

producto C (3)	g_3
si $u_1 = 0$ ó $u_2 \leq 1$	0
si $\{u_1=1$ y $u_2\geq 2\}$ ó $\{u_1\geq 1$ y $u_2=2\}$	10
si $u_1 \geq 2$ y $u_2 = 3$	18
si $u_1 \geq 2$ y $u_2 \geq 4$	25

ETAPA 1

		(PRODUCTOS TERMINADOS)						
x_1	x_2	0	1	2	3	$J_1(x)$	u_1^*	u_2^*
[0,1]	[0,4]	0	-	-	-	0	0	0
2	0	0	-	-	-	0	0	0
2	[1,4]	0	5	-	-	5	2	1
[3,4]	0	0	-	-	-	0	0	0
[3,4]	1	0	5	9	-	9	3	1
[3,4]	[2,4]	0	5	9	12	12	3	2

ETAPA 2

		(PRODUCTOS TERMINADOS)						
x_1	x_2	0	1	2	3	$J_2(x)$	u_1^*	u_2^*
0	[0,4]	0	-	-	-	0	0	0
1	0	0	-	-	-	0	0	0
1	[1,4]	0	8	-	-	8	1	1
2	0	0	-	-	-	0	0	0
2	1	6	8	-	-	8	1	1
2	[2,4]	6	8	12	-	12	2	2
3	0	0	-	-	-	0	0	0
3	1	8	8	-	-	8	0	0
3	[2,4]	12	11	12	17	17	3	2
4	0	0	-	-	-	0	0	0
4	1	9	8	-	-	9	0	0
4	2	12	15	12	17	17	3	2
4	[3,4]	12	18	17	17	18	1	1

ETAPA 3

		(PRODUCTOS TERMINADOS)						
x_1	x_2	0	1	2	3	$J_3(x)$	u_1^*	u_2^*
4	4	18	27	24	25	27	1	2

la solución óptima está dada por

Hacer:	0	art.	del	producto	A
	3	"	"	"	B
	1	"	"	"	C

Solución con el paquete

```
C **  PROBLEMA DE ASIGNACION DE RECURSOS
C **  FINANCIEROS A CUATRO PROYECTOS EN CONSTRUCCION
C **
      SUBROUTINE STATE
      COMMON/MULTDM/X(5),X1(5),U(5),F,I,J(5),K(5),L(5),PNALTY
      DO 10 JD = 1,2
      U(JD) = X(JD) - X1(JD)
   10 CONTINUE
      RETURN
      END
C **
      SUBROUTINE OBJECT
      COMMON/MULTDM/X(5),X1(5),U(5),F,I,J(5),K(5),L(5),PNALTY
      GO TO (1,2,3), I
    3 IF ((U(1).LE.1.0) .OR. (U(2).EQ.0.0)) THEN
        F = 0.0
        RETURN
      ELSE
        IF ((U(1).EQ.2.0) .AND. (U(2).GE.1.0)) THEN
          F = 5.0
          RETURN
        ELSE
          IF ((U(1).GE.3.0) .AND. (U(2).GE.2.0)) THEN
            F = 12.0
            RETURN
          ELSE
            F = 9.0
            RETURN
          ENDIF
        ENDIF
      ENDIF
    2 IF ((U(1).EQ.0.0) .OR. (U(2).EQ.0.0)) THEN
        F = 0.0
        RETURN
      ELSE
        IF ((U(1).EQ.2.0) .AND. (U(2).GE.2.0)) THEN
          F = 12.0
          RETURN
        ELSE
          IF ((U(1).GE.3.0) .AND. (U(2).GE.2.0)) THEN
            F = 17.0
            RETURN
          ELSE
            F = 6.0
```

```
          RETURN
        ENDIF
      ENDIF
    ENDIF
  1 IF ((U(1).EQ.0.0) .OR. (U(2).LE.1.0)) THEN
      F = 0.0
      RETURN
    ELSE
      IF ((U(1).GE.2.0) .AND. (U(2).EQ.3.0)) THEN
        F = 18.0
        RETURN
      ELSE
        IF ((U(1).GE.2.0) .AND. (U(2).GE.4.0)) THEN
          F = 25.0
          RETURN
        ELSE
          F = 10.0
          RETURN
        ENDIF
      ENDIF
    ENDIF
    RETURN
    END
    SUBROUTINE READIN
    RETURN
    END
```

Title : PROBLEMA DE ASIGNACION DE MATERIAS PRIMAS

```
**************************************
*                                    *
*        2 Dimensional Problem       *
*                                    *
*         Maximization Problem       *
*                                    *
*     Objective is Summation Type    *
*                                    *
*      Deterministic Optimization    *
*                                    *
*      Problem Assumed Invertible    *
*                                    *
*        First Tie Value Taken       *
*                                    *
*        Number of Stages =  3       *
*                                    *
**************************************
```

```
Initial Interval for X =  1.000
  Final Interval for X =  1.000
Order of Accuracy for U =  1.000
          SPLICE =  1.000
Maximum No. of Iterations =     20
     IPRINT Option =      2
```

Upper and Lower Bounds on X(I+1,JD) and U(I,JD)

I	JD	XMIN(I,JD)	XMAX(I,JD)	UMIN(I,JD)	UMAX(I,JD)
1	1	4.000	4.000	.0000	2.000
1	2	4.000	4.000	.0000	4.000
2	1	.0000	4.000	.0000	3.000
2	2	.0000	4.000	.0000	2.000
3	1	.0000	4.000	.0000	3.000
3	2	.0000	4.000	.0000	2.000
4	1	.0000	4.000		
4	2	.0000	4.000		

Initial Trajectories :

I	X(I,1)	X(I,2)
1	4.000000	4.000000
2	4.000000	4.000000
3	4.000000	4.000000
4	4.000000	4.000000

Iteration 1

Current Trajectories :

I	X(I,1)	X(I,2)
1	4.000000	4.000000
2	4.000000	4.000000
3	3.000000	3.000000
4	3.000000	3.000000

Decision Variables

I	U(I,1)	U(I,2)
1	.0000000	.0000000
2	1.000000	1.000000
3	.0000000	.0000000

Objective Value = 6.0000000

Iteration 2

Current Trajectories :

I	X(I,1)	X(I,2)
1	4.000000	4.000000
2	4.000000	4.000000
3	2.000000	2.000000
4	2.000000	2.000000

Decision Variables

I	U(I,1)	U(I,2)
1	.0000000	.0000000
2	2.000000	2.000000
3	.0000000	.0000000

Objective Value = 12.000000

Iteration 3

Current Trajectories :

I	X(I,1)	X(I,2)
1	4.000000	4.000000
2	4.000000	3.000000
3	1.000000	1.000000
4	1.000000	1.000000

Decision Variables

I	U(I,1)	U(I,2)
1	.0000000	1.000000
2	3.000000	2.000000
3	.0000000	.0000000

Objective Value = 17.000000

Iteration 4

Current Trajectories :

I	X(I,1)	X(I,2)
1	4.000000	4.000000
2	3.000000	2.000000
3	.0000000	.0000000
4	.0000000	.0000000

Decision Variables

I	U(I,1)	U(I,2)
1	1.000000	2.000000
2	3.000000	2.000000
3	.0000000	.0000000

Objective Value = 27.000000

*** Optimal Solution ***

Iteration 4

Final Trajectories :

I	X(I,1)	X(I,2)
1	4.000000	4.000000
2	3.000000	2.000000
3	.0000000	.0000000
4	.0000000	.0000000

Decision Variables

I	U(I,1)	U(I,2)
1	1.000000	2.000000
2	3.000000	2.000000
3	.0000000	.0000000

```
Objective Function Values

   I        F(I)

****************************************************************************
        *
   1    *   10.00000
   2    *   17.00000
   3    *   .0000000

Objective Value =   27.000000
```

4.4 PROBLEMA DE UN INVENTARIO ESTOCASTICO

En un sistema productivo, el manejo adecuado de los inventarios es necesario cuando se tienen recursos escasos y se desea hacer un uso eficiente de la infraestructura disponible. El problema básico de inventarios consiste en determinar un nivel de inventario óptimo que minimice los costos de operación, o bien, que maximice las ganancias que se obtienen por la venta de productos, considerando los costos por mantener el inventario. Las restricciones de este tipo de problemas pueden ser: capacidad de almacenamiento; capacidad de compra; problemas de operación, según el volumen de los productos, etcétera. También, otro tipo de restricciones externas, como son: demanda esperada; tiempo de entrega de las mercancías ordenadas; cantidad mínima por orden; etcétera. Además podemos encontrarnos con políticas de operación ya definidas, por ejemplo: cubrir el total de la demanda o penalizar por la demanda no cubierta debido a que se terminen las existencias o mantener siempre un mínimo de inventario para las emergencias, etc.. En general el manejo de inventarios es sencillo cuando la demanda de artículos es determinística, sin embargo, el problema se complica cuando tenemos que hacer frente a una demanda estocástica.

Debido a: la gran variedad de problemas de inventarios que pueden existir, las diferentes políticas de operación definidas para un problema en particular y demás restricciones; no se planteará un problema de inventario tipo en general, sino que se propone directamente, como ilustración, un ejemplo donde la demanda es estocástica, que es el caso interesante.

Ejemplo 5. Un revendedor puede ordenar tanques de gas al inicio de cada semana y recibir la entrega al principio de la otra. Cada cilindro cuesta $10 y es vendido en $18. Además hay un cargo fijo de $5 por cada orden de entrega.

Existe un costo de almacenamiento de $1 por cada tanque que permanece en bodega durante una semana completa. La demanda de los tanques en la semana n es una variable aleatoria R_n que toma el valor r con probabilidad $p_n(r)$ ($p_n(r) = P[R_n = r]$) como se muestra en la tabla siguiente.

Inventario estocástico
probabilidad de la demanda $p_n(r)$

demanda	semana n					
r	1	2	3	4	5	6
0	0.3	0.5	0.3	0.1	0.1	0.5
1	0.3	0.5	0.5	0.2	0.2	0.4
2	0.4	0	0.2	0.4	0.3	0.1
3	0	0	0	0.3	0.4	0

La demanda que no es satisfecha inmediatamente se pierde. Los tanques que no son vendidos al final de la sexta semana tienen un valor de rescate de $8 . Determínese una política óptima para elaborar las órdenes.

Solución:
Estar en la etapa i significa estar en la semana i. El estado x_i de la etapa i ,es el nivel de inventario al inicio de la semana i. La variable de decisión u_i es ordenar u_i cilindros al inicio de la etapa i. $g_i(x_i,u_i,r)$ es la ganancia obtenida en la etapa i, dada una demanda de r tanques, considerando una existencia de x_i y que se efectúa una orden de u_i tanques. $J_i(x_i)$ es la ganancia máxima esperada cuando nos encontramos en la semana i con un nivel de inventario de x_i

Como no se menciona un nivel de inventario inicial, no se tiene una función $J_1(x_1)$ especifica que calcular. La utilidad por etapa viene dada por las ganancias de la demanda menos el costo de almacenamiento de los tanques que no se venden menos el costo de hacer una orden, o sea que

$$g_1(x_1,0,r) = \begin{cases} 18r - (x_1 - r) & \text{si } 0 \le r \le x_1 \\ 18x_1 & \text{si } r > x_1 \end{cases}$$

$$g_1(x_1,u_1,r) = \begin{cases} 18r - (x_1 - r) - (10u_1+5) & \text{si } 0 \le r \le x_1 \\ 18x_1 - (10u_1+5) & \text{si } r > x_1 \end{cases}$$

$$x_{1+1} = \begin{cases} (x_1-r)+u_1 & \text{si } 0 \le r \le x_1 \\ u_1 & \text{si } r > x_1 \end{cases} \quad (1)$$

la ecuación viene dada por

$$J_1(x_1) = \max_{u_1} E\,[\, g_1(x_1,u_1,r) + J_{1+1}(x_{1+1}) \,]$$

de donde

$$E\,[J_{1+1}(x_{1+1})] = \sum_{r=0}^{x_1-1} J_{1+1}(x_1-r+u_1)\ P\ [R_1 = r] + \sum_{r=x_1}^{N} J_{1+1}(u_1)\ P\ [R_1 = r]$$

$$= \sum_{r=0}^{x_1} J_{1+1}(x_1-r+u_1)\ P\ [R_1 = r] + J_{1+1}(u_1)\ P\ [R_1 > x_1]$$

$$E\,[g_1(x_1,u_1,r)] = \sum_{r=0}^{x_1} [18r - (x_1-r) - (10u_1+5)]\ P\ [R_1 = r] +$$

$$+ \sum_{r=x_1+1}^{N} [18x_1 - (10u_1+5)]\ P\ [R_1 = r]$$

$$= 18 \left\{ \sum_{r=0}^{x_1} r\ P\ [R_1 = r] + x_1 \sum_{r=x_1+1}^{N} P\ [R_1 = r] \right\} -$$

$$- \left\{ x_1 \sum_{r=0}^{x_1} P\ [R_1 = r] - \sum_{r=0}^{x_1} r\ P\ [R_1 = r] \right\} - (10u_1+5)$$

por lo tanto

$$E\,[g_i(x_i,u_i,r)] = 18\sum_{r=1}^{x_i} P\,[R_i \geq r] - \sum_{r=1}^{x_i} P\,[R_i < r] - (10u_i+5)$$

resumiendo la ecuación recursiva quedaría como

$$J_i(x_i) = \max_{u_i} \left\{ 18\sum_{r=1}^{x_i} P\,[R_i \geq r] - \sum_{r=1}^{x_i} P\,[R_i < r] - (10u_i+5)I_{\{1,2,..\}}(u_i) + \right.$$

$$\left. + \sum_{r=0}^{x_i} J_{i+1}(x_i-r+u_i)\; P\,[R_i = r] + J_{i+1}(u_i)\; P\,[R_i > x_i] \right\} \qquad (2)$$

donde

$$I_{\{1,2,..\}}(u_i) = \begin{cases} 1 & \text{si} \quad u_i > 0 \\ 0 & \text{si} \quad u_i = 0 \end{cases}$$

Solución:

Ejemplo 6 Muchas veces este tipo de problemas se plantean en términos de probabilidades de transición entre los estados posibles de etapas contiguas (cadenas de Markov). Para plantear el problema anterior en estos términos definiremos a $p_i(x_i,x_{i+1},u_i)$ como la probabilidad de transición de pasar del estado x_i al estado x_{i+1} tomando la acción u_i, para ser $p_i(x_i,x_{i+1},u_i)$ probabilidad de transición, debe cumplir con las siguientes condiciones:

1) $0 \leq p_i(x_i,x_{i+1},u_i) \leq 1$

2) $\sum_{x_{i+1}} p_i(x_i,x_{i+1},u_i) = 1$ para cada x_i y u_i fijas

ahora bien para deducir $p_i(x_i,x_{i+1},u_i)$, a partir de los datos disponibles del ejemplo 5, supongase que $u_i = 0$, entonces la probabilidad de pasar a un estado x_{i+1} desde un estado x_i es igual a la probabilidad de la demanda requerida

$$p_i(x_i,x_{i+1},0) = P\,[R_i = x_i - x_{i+1}] = P\,[R_i = r]$$

s.a.

$$x_{i+1} > 0;\; x_{i+1} \leq x_i;\; r = x_i - x_{i+1}$$

cuando el estado resultante x_{i+1} es cero puede ser que la demanda haya excedido a la oferta, por lo que se tiene que

$$p_i(x_i,0,0) = P\,[R_i \geq x_i]$$

resumiendo se tiene que

$$p_i(x_i,x_{i+1},0) = \begin{cases} P\,[R_i = x_i - x_{i+1}] & \text{si } 0 < x_{i+1} \leq x_i \\ P\,[R_i \geq x_i] & \text{si } x_{i+1} = 0 \text{ y } x_i \geq 0 \\ 0 & \text{en otro caso} \end{cases}$$

nótese que $p_i(x_i,x_{i+1},0)$ cumple con las condiciones 1) y 2), ahora bien si $u_i > 0$ de la ecuación (1) se tiene que $x_{i+1} = x_i - r + u_i$ despejando a r tenemos que

$$r = x_i - x_{i+1} + u_i$$

tomando probabilidades se obtiene

$$P\,[R_i = r] = P\,[R_i = x_i - x_{i+1} + u_i]$$

$$= P\,[R_i = x_i - (x_{i+1} - u_i)] = p_i(x_i,(x_{i+1} - u_i),0)$$

por lo tanto

$$p_i(x_i,x_{i+1},u_i) = p_i(x_i,(x_{i+1} - u_i),0)$$

sustituyendo en la ecuación (2) se tiene que

$$J_i(x_i) = \max_{u_i} \Big\{ 18 \sum_{r=1}^{x_i} P\,[R_i \geq r] - \sum_{r=1}^{x_i} P\,[R_i < r] - (10u_i + 5) I_{\{1,2,..\}}(u_i) + \sum_{x_{i+1}=u_i}^{x_i+u_i} J_{i+1}(x_{i+1})\; p_i(x_i,x_{i+1},u_i) \Big\}$$

o bien

$$J_i(x_i)=\max_{u_i}\left\{ 18\sum_{r=1}^{x_i} P\{R_i \geq r\} - \sum_{r=1}^{x_i} P\{R_i < r\} - (10u_i+5)I_{\{1,2,..\}}(u_i) + \right.$$

$$\left. + \sum_{r=0}^{x_i} J_{i+1}(x_i - r + u_i)\, p_i(x_i,(x_i - r),0) \right\}$$

la condición de frontera está dada por el valor de rescate de los tanques $J_0(x_0) = 8x_0$. Los cálculos para este problema son muy extensos para poder consignarlos aquí en tablas, pero la política óptima de elaboración de las órdenes se muestra a continuación:

semana	estado	acción
I	x_i	u_i
1	0	2
	$x_i \geq 1$	0
2	0	5
	$x_i \geq 1$	0
3	0	5
	1	5
	2	4
	$x_i \geq 3$	0
4	0	3
	1	3
	2	3
	3	2
	$x_i \geq 4$	0
5	$x_i \geq 0$	0
6	$x_i \geq 0$	0

En seguida se presenta la solución obtenida por el paquete CSUDP, la cual es idéntica a la solución que se acaba de presentar. Primero se muestra la codificación de las subrutinas STATE, OBJECT y READIN; después se proporciona el listado de soluciones. Por cuestiones de espacio se enlistan únicamente los primeros cinco estados de cada etapa pero, en realidad, el número de estados considerados para el proceso de solución, los muestra el paquete en listado correspondiente.

```
      SUBROUTINE STATE
      COMMON/ONEDM/X,X1,U,F,I,J,K,L,R,PNALTY
      X1 = X + U - R
      IF (R.GT.X) X1 = U
      RETURN
      END


      SUBROUTINE OBJECT
      COMMON/ONEDM/X,X1,U,F,I,J,K,L,R,PNALTY
      COMMON/DATO/IN
C **  DONDE N1 ES EL NUMERO DE ETAPAS
      IF (R.LT.X) THEN
       F = 18.*R - (X - R)
       IF (I.EQ.IN) F = F + 8*(X - R)
      ELSE
       F = 18.*X
      END IF
      IF (U.GT.0) F = F - (10.*U+5)
      RETURN
      END


      SUBROUTINE READIN
      COMMON/DATO/IN
      REWIND 5
      READ (5,100)IN
      WRITE (*,200)IN
  100 FORMAT(/,45X,I5)
  200 FORMAT(/,'    NUMERO DE ETAPAS',I5,/)
      RETURN
      END
```

Title : PROBLEMA DE INVENTARIO ESTOCASTICO

1 Dimensional Problem

Maximization Problem

Objective is Summation Type

Stochastic Optimization

Problem Assumed Non-Invertible

First Tie Value Taken

Splicing will not Occur

Number of Stages = 6

Interval for X = 1.000
Interval for U = 1.000

Risk Level for XMIN(I+1) = .0000
Risk Level for XMAX(I+1) = .0000

Stochastic Analysis - Random Variables and Probabilities

1	.000(.300)	1.000(.300)	2.000(.400)	
2	.000(.500)	1.000(.500)		
3	.000(.300)	1.000(.500)	2.000(.200)	
4	.000(.100)	1.000(.200)	2.000(.400)	3.000(.300)
5	.000(.100)	1.000(.200)	2.000(.300)	3.000(.400)
6	.000(.500)	1.000(.400)	2.000(.100)	

Upper and Lower Bounds on X(I+1) and U(I)

I	XMIN(I)	XMAX(I)	UMIN(I)	UMAX(I)
1	.0000	5.000	.0000	5.000
2	.0000	10.00	.0000	5.000
3	.0000	15.00	.0000	5.000
4	.0000	20.00	.0000	5.000
5	.0000	25.00	.0000	5.000
6	.0000	30.00	.0000	5.000
7	.0000	35.00		

Optimal Policies

Stage 6					
X(6)	.00000	1.0000	2.0000	3.0000	4.0000	5.0000
U(X(6)	.00000	.00000	.00000	.00000	.00000	.00000
F(X(6)	.00000	12.500	20.600	27.600	34.600	41.600

Optimal Policies

Stage 5					
X(5)	.00000	1.0000	2.0000	3.0000	4.0000	5.0000
U(X(5)	.00000	.00000	.00000	.00000	.00000	.00000
F(X(5)	.00000	17.350	32.960	45.630	54.160	60.600

Optimal Policies

Stage 4					
X(4)	.00000	1.0000	2.0000	3.0000	4.0000	5.0000
U(X(4)	3.0000	3.0000	3.0000	2.0000	.00000	.00000
F(X(4)	10.630	27.583	42.233	53.132	65.031	76.132

Optimal Policies

Stage 3					
X(3)	.00000	1.0000	2.0000	3.0000	4.0000	5.0000
U(X(3)	5.0000	5.0000	4.0000	.00000	.00000	.00000
F(X(3)	21.132	35.862	46.442	56.673	67.622	78.082

Optimal Policies

Stage 2

X(2)	.00000	1.0000	2.0000	3.0000	4.0000	5.0000
U(X(2)	5.0000	.00000	.00000	.00000	.00000	.00000
F(X(2)	23.082	36.997	48.652	58.057	67.647	77.352

Optimal Policies

Stage 1

X(1)	.00000	1.0000	2.0000	3.0000	4.0000	5.0000
U(X(1)	2.0000	.00000	.00000	.00000	.00000	.00000
F(X(1)	23.652	39.556	53.827	64.712	74.072	82.623

C O N C L U S I O N E S

En este trabajo se ha presentado una herramienta importante de optimización: la Programación Dinámica. Se analizó su estructura general, describiendo sus componentes y poniendo énfasis en su propiedad recursiva, que, como se observa en los ejemplos desarrollados, reduce en gran medida el esfuerzo computacional de problemas de tipo combinatorio. También se destacó lo adecuado de seguir un procedimiento ordenado para el planteamiento de problemas, pues de esta forma se identifican con mayor facilidad sus componentes para definir la función recursiva, con la cual se realizan los cálculos. Aquí es importante destacar otra propiedad de la programación dinámica, que es la de ser intuitiva, es decir, para resolver un problema por este método necesitamos, preponderantemente, "echar mano" de la intuición y no de estructuras matemáticas elaboradas.

Otro punto desarrollado es ver cómo la estructura de la programación dinámica se adapta muy bien a problemas que tienen incertidumbre en su sistema, mostrando así la versatilidad de este modelo, sobre todo, porque puede combinarse con otras herramientas como se mostró en el caso de las cadenas de Markov con ganancias, en donde es útil para buscar políticas de ganancia óptima.

Posteriormente se presenta el paquete CSUDP, que permite resolver problemas de programación dinámica de manera general. Se describieron las características del paquete y el modo de aplicarlo a problemas prácticos. Se pudo observar que el paquete es poco atractivo en su presentación y complicado en su uso, en comparación con otros paquetes similares de modelos de óptimización diferentes. Pero no se le puede descalificar por esta circunstancia, dadas sus características de uso general dentro de la programación dinámica, ya que paquetes de este tipo no abundan. Por último se proporcionaron algunos ejemplos de aplicación que ilustran las posibilidades prácticas de la programación dinámica y del paquete presentado en esta tesis.

Con lo expuesto en este trabajo, vemos que la programación dinámica es una herramienta importante de la investigación de operaciones que debe tomarse en cuenta en la solución de problemas dinámicos, como pueden ser los problemas de control. Aunque la aplicación de la programación dinámica a problemas reales es escasa, considero que esta técnica puede popularizarse más, conforme se difundan paquetes generalizados de programación dinámica, como el presentado en este trabajo, o bien, porque se presente combinada con otros métodos como son: el de ramificación y acotamiento (Branch and Bound) y el de la programación dinámica incremental.

BIBLIOGRAFIA

1] BELLMAN, RICHARD, *DYNAMIC PROGRAMMING*, PRINCETON UNIVERSITY PRESS, NEW JERSEY 1965.

2] DENARDO, ERIC V., *DYNAMIC PROGRAMMING (Models and Aplications)*, PRENTICE HALL, INC., NEW JERSEY 1982.

3] HASTINGS, N.A.J., *DYNAMYC PROGRAMMING (With Management Aplications)*, CRANE, R. & COMPANY, INC., NEW YORK 1973.

4] HILLIER, F. and LIEBERMAN, G., *OPERATIONS RESEARCH*, HOLDEN-DAY, INC., 2ª Ed., SAN FRANCISCO 1974.

5] LABADIE, J.W., *DYNAMIC PROGRAMMING WITH THE MICRO COMPUTER*, COLORADO STATE UNIVERSITY 1986.

6] MINE, H. and OSAKI, S., *MARKOVIAN DECISION PROCESSES*, AMERICAN ELSEVIER PUBLISHING CO., INC., NEW YORK 1970.

7] MOSKOWITZ, H. and WRIGHT, G.P., *INVESTIGACION DE OPERACIONES*, PRENTICE-HALL HISPANOAMERICANA S.A., MEXICO 1986.

8] PRAWDA, JUAN, *METODOS Y MODELOS DE INVESTIGACION DE OPERACIONES*, ED. LIMUSA, MEXICO 1989.

9] TAHA, HAMDY A., *OPERATIONS RESEARCH (an Introduction)*, MACMILLAN PUBLISHING CO., INC., 2ª Ed., NEW YORK 1976.

R E F E R E N C I A S

[1] HUNG-PO CHAO and ALAN S. MANNE, *Oil Stockpiles and Import Reductions: A Dynamic Programming Aproach*, Operations Research, 31, No. 4, 632-651. 1983.

[2] RICHARS L. KILMER, THOMAS SPREEN and D.S. TILLEY, *A Dynamic Plant Location Model: The East Florida Fresh Citrus Packing Industry*, American Journal of Agricultural Economics, 65, No.4, 730-37, Nov 1983, en Operations Research/Management Science, XXIV, No.2, 99-100, 1984.

[3] GERALD W. EVANS and THOMAS L. MORIN, *Hybrid Dynamic Branch-and-Bound Strategies for Electric Power Generation Planning*, IIE Transactions, 18, No. 2, 138-47, 1986, en OP/MS, XXVII, No. 1, 25, 1987.

[4] ROBERT E. LARSON, *STATE INCREMENT DYNAMIC PROGRAMMING*, AMERICAN ELSEVIER PUBLISHING COMPANY, INC., NEW YORK 1968.

[5] THOMAS L. MORIN and ROY E. MARSTEN, *Branch-and-Bound Strategies for Dynamic programming*, Operations Research, 24, NO. 4, 611-27, 1976.

[6] J.S.H. KORNBLUTH, *Multiple Objective Dynamic Programming*, Computers and Operations Research, 18, No. 4, 517-24, 1986, en OP/MS, XXVII, No. 8, 445-47,1987.