Universidad de Vigo

Universidade de Vigo

Trabajo de Teoría

SISTEMAS INTELIGENTES

-

Programación Dinámica

SI – 52

ÍNDICE

Introducción 3
Marco Teórico 4
Definición de Programación Dinámica 4
Principio de Optimalidad de Bellman 4
Aplicaciones Generales de la Programación Dinámica 4
AgentSpeak y su Utilidad en Sistemas Inteligentes 4
Formulación del Problema de Inventarios 6
Descripción formal del problema 6
Variables, parámetros y restricciones 6
Tipos de costes 6
Ejemplo de función objetivo 7
Modelización y Resolución con Programación Dinámica 8
Planteamiento de etapas, estados y decisiones 8
Ecuación de recurrencia 8
Explicación del algoritmo de PD paso a paso 8
Ventajas frente a otros métodos 9
Ejemplo Numérico Detallado: Inventario con Programación Dinámica 10
Planteamiento del caso 10
Resolución manual paso a paso 10
Explicación de la lógica del agente 12
Código AgentSpeak 12
Ejemplo de ejecución y resultados 14
Ventajas de usar agentes para este tipo de problemas 14
Diagrama de Interacción 15
Agente Almacén 15
Agente Gestor 15
Interacción entre Almacén y Gestor 16
Propósito del Sistema 16
Diagrama 16
Conclusiones 18
Bibliografía 19

INTRODUCCIÓN

La gestión de inventarios es clave para cualquier empresa que maneje productos, ya que un control eficiente permite equilibrar la oferta y la demanda, evitando tanto faltantes como excesos y mejorando la rentabilidad.

La creciente volatilidad de la demanda y la complejidad de las cadenas de suministro hacen imprescindible optimizar el inventario mediante técnicas avanzadas como la programación dinámica y el uso de sistemas inteligentes. Una gestión inadecuada puede derivar en pérdidas económicas y pérdida de competitividad, por lo que la optimización de inventarios es hoy un objetivo prioritario.

Este trabajo analiza y aplica la programación dinámica para resolver problemas de inventarios, buscando políticas óptimas de pedido que minimicen los costes totales. Se presenta una formulación matemática, un caso práctico y la implementación de la solución en un agente inteligente, evaluando su eficacia en contextos empresariales reales.

MARCO TEÓRICO

DEFINICIÓN DE PROGRAMACIÓN DINÁMICA

La programación dinámica es una metodología matemática y computacional desarrollada por Richard Bellman en la década de 1950, orientada a la resolución de problemas de optimización que pueden ser descompuestos en una secuencia de subproblemas interrelacionados. Su principal característica es la capacidad de abordar problemas complejos dividiéndolos en etapas o decisiones sucesivas, donde la solución óptima global se construye a partir de las soluciones óptimas de los subproblemas.

La programación dinámica se aplica a situaciones en las que las decisiones tomadas en un momento afectan el estado futuro del sistema, permitiendo así la optimización secuencial a lo largo del tiempo.

PRINCIPIO DE OPTIMALIDAD DE BELLMAN

El principio de optimalidad, formulado por Richard Bellman, establece que una política óptima tiene la propiedad de que, independientemente del estado y la decisión inicial, las decisiones restantes deben constituir una política óptima respecto al estado resultante de la primera decisión. La solución óptima de un problema de decisión secuencial se puede obtener resolviendo de manera óptima cada subproblema que surge tras cada decisión.

Matemáticamente, este principio permite expresar la función de valor óptima de un estado como una relación recursiva, conocida como ecuación de Bellman:

$$V(x_0) = \max_{a_0}\{F(x_0, a_0) + \beta V(x_1)\}$$

donde x_0 es el valor óptimo a partir del estado inicial x_0, $F(x_0, a_0)$ es la recompensa inmediata al tomar la acción a_0 y β es un factor de descuento que pondera el valor futuro.

APLICACIONES GENERALES DE LA PROGRAMACIÓN DINÁMICA

La programación dinámica se utiliza en una amplia variedad de campos, incluyendo la ingeniería, la economía, la informática y la investigación operativa. Entre sus aplicaciones más destacadas se encuentran la planificación de rutas óptimas en redes de transporte, la gestión de inventarios, la asignación de recursos, el control óptimo de procesos, la toma de decisiones financieras y la resolución de problemas combinatorios como la mochila o el alineamiento de secuencias en bioinformática.

AGENTSPEAK Y SU UTILIDAD EN SISTEMAS INTELIGENTES

AgentSpeak es un lenguaje de programación basado en la lógica, diseñado para modelar el comportamiento de agentes inteligentes dentro de sistemas multiagente. Inspirado en la arquitectura BDI (Belief-Desire-Intention), AgentSpeak permite especificar creencias, deseos y planes de acción de un agente, facilitando la toma de decisiones autónoma y adaptativa en entornos dinámicos. AgentSpeak resulta especialmente útil para implementar

agentes que resuelven problemas complejos de manera distribuida, como la gestión de inventarios, la coordinación de robots o la negociación automática.

FORMULACIÓN DEL PROBLEMA DE INVENTARIOS

El problema de inventarios se centra en determinar la política óptima de pedidos y almacenamiento de productos, de modo que se minimicen los costes totales asociados al sistema, garantizando al mismo tiempo la satisfacción de la demanda en cada periodo considerado. Este tipo de problemas es fundamental en la gestión de operaciones, ya que una mala planificación puede derivar en costes innecesarios o en la falta de productos para los clientes.

DESCRIPCIÓN FORMAL DEL PROBLEMA

El problema consiste en decidir, para cada periodo de un horizonte temporal dado, cuántas unidades de un producto deben pedirse o producirse, teniendo en cuenta la demanda prevista, el inventario disponible y las restricciones propias del sistema. El objetivo es encontrar una secuencia de decisiones que minimice el coste total, considerando tanto los costes directos como los indirectos.

VARIABLES, PARÁMETROS Y RESTRICCIONES

Las variables principales suelen ser:

- x_t: cantidad de producto que se tiene en inventario al final del periodo t .
- q_t : cantidad de producto que se pide o produce en el periodo t .

Entre los parámetros más habituales se encuentran:

- d_t : demanda del producto en el periodo t .
- c : coste unitario de adquisición o producción.
- h : coste unitario de almacenamiento por periodo.
- p : coste unitario de rotura de stock o penalización por no satisfacer la demanda.
- x_0 : inventario inicial.

Las restricciones típicas incluyen:

- Balance de inventario: $x_t = x_{t-1} + q_t - d_t$ para cada periodo t .
- No negatividad: $x_t \geq 0$, $q_t \geq 0$.
- Restricciones de capacidad de almacenamiento o de pedido, si las hubiera.

TIPOS DE COSTES

En la mayoría de los modelos de inventarios se consideran al menos tres tipos de costes:

- Coste de adquisición o pedido: asociado a la compra o producción de nuevas unidades.
- Coste de almacenamiento: relacionado con mantener productos en inventario, incluyendo gastos de espacio, seguros y deterioro.

- Coste de rotura de stock: penalización por no poder satisfacer la demanda, que puede traducirse en pérdida de ventas o en costes adicionales por pedidos urgentes.

EJEMPLO DE FUNCIÓN OBJETIVO

La función objetivo suele estar orientada a minimizar el coste total a lo largo del horizonte de planificación. Un ejemplo típico de función objetivo es:

Minimizar $Z = \sum_{t=1}^{T} [c \cdot q_t + h \cdot x_t + p \cdot s_t]$

donde s_t representa la cantidad de demanda insatisfecha en el periodo t.

Esta formulación permite analizar diferentes políticas de inventario y comparar su impacto en los costes totales, facilitando la toma de decisiones informada en la gestión de stocks.

MODELIZACIÓN Y RESOLUCIÓN CON PROGRAMACIÓN DINÁMICA

PLANTEAMIENTO DE ETAPAS, ESTADOS Y DECISIONES

La programación dinámica (PD) es una técnica para resolver problemas de optimización que pueden descomponerse en una secuencia de decisiones (etapas). Cada etapa representa un momento o subproblema, y en cada una se debe tomar una decisión que afecta el estado del sistema.

- Etapas: Son los momentos en los que se toma una decisión. Por ejemplo, en un problema de inventarios, cada etapa puede ser un periodo de tiempo.
- Estados: Describen la situación del sistema en una etapa determinada. Por ejemplo, el nivel de inventario disponible al inicio de un periodo.
- Decisiones: Son las acciones posibles en cada etapa y estado. Por ejemplo, cuánto pedir o producir en ese periodo.

ECUACIÓN DE RECURRENCIA

La **ecuación de recurrencia** es el corazón de la PD. Permite expresar el valor óptimo de la función objetivo en una etapa y estado, en función de las decisiones posibles y del valor óptimo en la siguiente etapa.

En notación general:

$$f_k(s) = \min_{a \epsilon A(s)} \{c_k(s, a) + f_{k+1}(T(s, a))\}$$

donde:

- $f_k(s)$: valor óptimo a partir de la etapa k en el estado s .
- a : decisión posible en el estado s .
- $c_k(s, a)$: coste inmediato de tomar la decisión a en el estado s en la etapa k .
- $T(s, a)$: estado resultante tras tomar la decisión a .
- $f_{k+1}T(s, a)$: valor óptimo futuro desde el nuevo estado.

Esta ecuación se resuelve de forma recursiva, normalmente desde la última etapa hacia la primera.

EXPLICACIÓN DEL ALGORITMO DE PD PASO A PASO

1. Definir etapas, estados y decisiones: Identificar claramente cada uno para el problema concreto.
2. Establecer la ecuación de recurrencia: Formular cómo se calcula el valor óptimo en cada etapa y estado.
3. Condiciones de frontera: Definir el valor óptimo en la última etapa (por ejemplo, coste cero si no hay más decisiones).
4. Resolución hacia atrás: Calcular el valor óptimo en cada estado de la última etapa, luego en la penúltima, y así sucesivamente hasta la primera etapa.

5. Reconstrucción de la política óptima: Una vez calculados los valores óptimos, se reconstruye la secuencia de decisiones óptimas siguiendo el camino que llevó a esos valores.

VENTAJAS FRENTE A OTROS MÉTODOS

- Descomposición eficiente: Permite dividir problemas complejos en subproblemas más simples y manejables.
- Evita cálculos redundantes: Almacena soluciones de subproblemas ya resueltos (memorización), evitando recalcularlos.
- Flexibilidad: Se aplica a problemas lineales y no lineales, deterministas o estocásticos, y con variables discretas o continuas.
- Principio de optimalidad: Garantiza que la solución global óptima se construye a partir de soluciones óptimas de subproblemas.

EJEMPLO NUMÉRICO: INVENTARIO CON PROGRAMACIÓN DINÁMICA

PLANTEAMIENTO DEL CASO

Una empresa debe satisfacer la demanda de un producto durante 3 periodos. Los datos son:

- Demanda por periodo: $d_1 = 3$, $d_2 = 2$, $d_3 = 4$
- Costo de producción por unidad: $c = 2$
- Costo de mantener inventario por unidad y periodo: $h = 1$
- Capacidad máxima de producción por periodo: 4 unidades
- Inventario inicial: 0 unidades
- Inventario final deseado: 0 unidades

Objetivo:
Determinar el plan de producción que minimice el costo total (producción + inventario).

RESOLUCIÓN MANUAL PASO A PASO

1. DEFINICIÓN DE ETAPAS, ESTADOS Y DECISIONES

- Etapas: Cada periodo de tiempo (1, 2, 3).
- Estados: Inventario disponible al inicio de cada periodo(x_t).
- Decisiones: Cantidad a producir en cada periodo (q_t).

2. ECUACIÓN DE RECURRENCIA

El costo mínimo desde el periodo t en adelante, partiendo del estado x_t:

$$f_t(x_t) = \min_{q_t} \quad [c \cdot q_t + h \cdot x_t + 1 + f_{t+1}(x_{t+1})]$$

donde
$x_{t+1} = x_t + q_t - d_t$ y q_t está limitado por la capacidad y la demanda.

3. RESOLUCIÓN HACIA ATRÁS (BACKWARD)

Etapa 3 (último periodo):
El inventario final debe ser 0, así que $x_4 = 0$.

$$f_3(x_3) = \vdash_{\infty}^{c \cdot q_3} \begin{matrix} si\ 0 \le q_3 \le 4\ y\ x_3 + q_3 - d_3 = 0 \\ en\ otro\ caso \end{matrix}$$

Etapa 2:
Para cada posible x_2, probamos todas las decisiones posibles (q_2), calculamos el inventario resultante x_3, y usamos el costo óptimo de la etapa 3.

Etapa 1:
Igual, pero partiendo de $x_1 = 0$.

4. TABLA DE RESULTADOS

Etapa	Estado x_t	Decisión q_t	Inventario siguiente (x_{t+1})	Costo inmediato	Costo futuro	Costo total
3	0	4	0	8	0	8
3	1	3	0	6	0	6
3	2	2	0	4	0	4
3	3	1	0	2	0	2
3	4	0	0	0	0	0

Ahora, hacia atrás:

Etapa 2:
Para cada x_2, probamos $q_2 = 0, 1, 2, 3, 4$, calculamos x_3, y sumamos el costo inmediato y el costo óptimo de la etapa 3.

Por ejemplo, para $x_2 = 0$:

- $q_2 = 2 : x_3 = 0 + 2 - 2 = 0$, costo inmediato $2 \cdot 2 + 1 \cdot 0 = 4$, costo futuro $f_3(0) = 8$, total $4 + 8 = 12$.
- $q_2 = 3 : x_3 = 0 + 3 - 2 = 1$, costo inmediato $2 \cdot 3 + 1 \cdot 1 = 7$, costo futuro $f_3(1) = 6$, total $7 + 6 = 13$.
- $q_2 = 4 : x_3 = 0 + 3 - 2 = 1$, costo inmediato $2 \cdot 4 + 1 \cdot 2 = 10$, costo futuro $f_3(2) = 4$, total $10 + 4 = 14$.

El mínimo es 12, con $q_2 = 2$.

5. ANÁLISIS

- El método permite encontrar el plan de producción óptimo, minimizando el costo total.
- Se observa que producir exactamente lo necesario en cada periodo puede no ser óptimo si los costos de inventario y producción varían.
- La tabla muestra cómo la decisión en cada etapa depende del estado y de los costos futuros, no solo del costo inmediato.

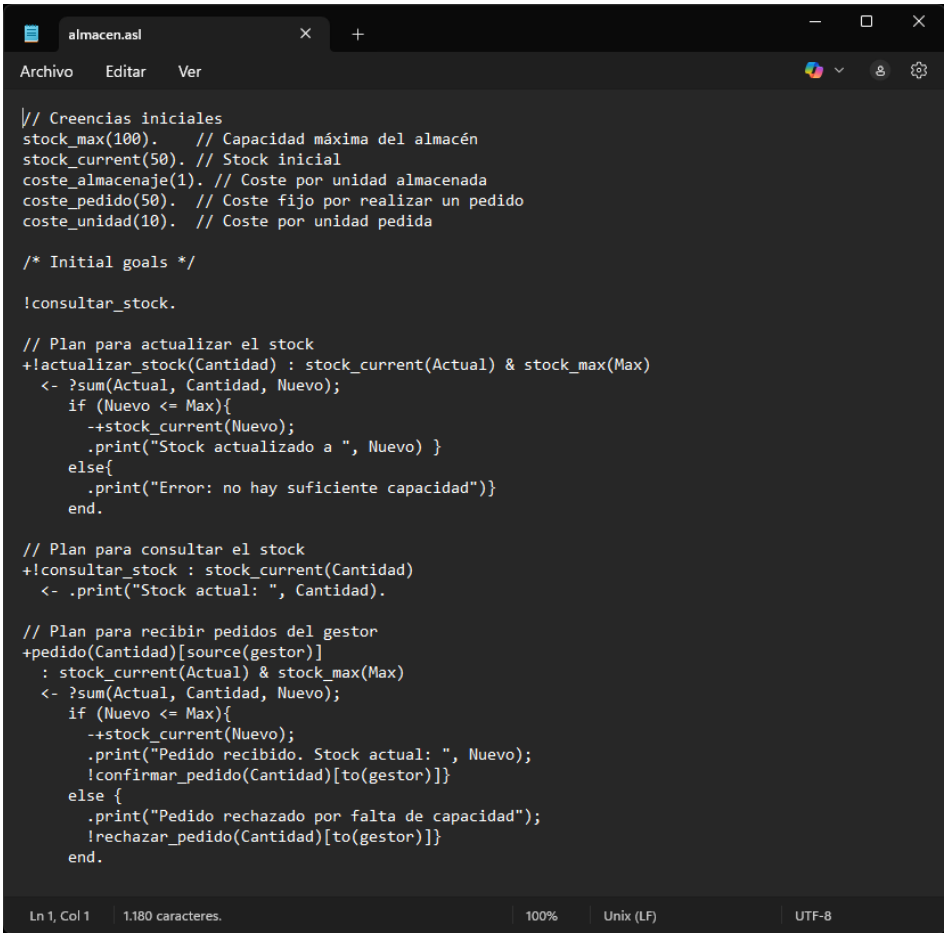
EXPLICACIÓN DE LA LÓGICA DEL AGENTE

El agente de gestión de inventarios está diseñado para tomar decisiones automáticas sobre cuándo y cuánto pedir, basándose en información como el stock actual, la demanda esperada y los costos asociados. Su lógica incluye:

- Monitoreo del inventario: El agente observa continuamente el nivel de stock.
- Predicción de demanda: Utiliza datos históricos o reglas para estimar la demanda futura.
- Cálculo de pedidos óptimos: Aplica un modelo para decidir la cantidad a pedir y el momento adecuado, minimizando costos de almacenamiento y faltantes.
- Comunicación y acción: El agente ejecuta pedidos y actualiza el sistema según las respuestas del entorno (por ejemplo, confirmación de recepción de productos).

CÓDIGO AGENTSPEAK

almacen.asl



gestor.asl

gestor.asl

Archivo Editar Ver

```
//gestor.asl
// Creencias iniciales
demanda_esperada(20). // Demanda esperada para el próximo período
stock_seguridad(10).  // Stock de seguridad mínimo

// Tabla de costes (para programación dinámica)
// Estructura: costo(Q, S) donde Q es cantidad a pedir, S es stock actual
costo(0, S, Coste) :-
    stock_current(S) & demanda_esperada(D) & max(0, D - S, Faltante) & max(0, S - D, Sobrante) &
    coste_almacenaje(Ca) & coste_unidad(Cu) &
    coste_pedido(Cp) &
    Coste = (Faltante * 100) + (Sobrante * Ca). // Coste por faltante alto

costo(Q, S, Coste) :-
    Q > 0 & stock_current(S) & demanda_esperada(D) &
    sum(S, Q, NuevoS) &
    max(0, D - NuevoS, Faltante) &
    max(0, NuevoS - D, Sobrante) &
    coste_almacenaje(Ca) & coste_unidad(Cu) &
    coste_pedido(Cp) &
    Coste = Cp + (Q * Cu) + (Faltante * 100) + (Sobrante * Ca).

//Inicio
!start.

// Plan de inicio
+!start
  <- .print("Iniciando gestión de inventario...");
     !gestionar_inventario.

// Plan principal para gestionar inventario
+!gestionar_inventario
  <- .print("Calculando mejor pedido...");
     !calcular_mejor_pedido;  // Corregido el nombre (añadida "d")
     !realizar_pedido_optimo.

// Plan para calcular el pedido óptimo (corregido el nombre)
+!calcular_mejor_pedido  // Corregido el nombre (añadida "d")
  : stock_current(S) & stock_seguridad(SS) & stock_max(Max)
  <- .print("Stock actual: ", S);
     .findall(C, costo(Q, S, C), ListaCostes);
     .min(ListaCostes, MinCoste);
     .findall(Q, costo(Q, S, MinCoste), [MejorQ|_]);
     +pedido_optimo(MejorQ);
     .print("Pedido óptimo calculado: ", MejorQ, " (coste: ", MinCoste, ")").
```

Ln 1, Col 1 | 2.242 caracteres. | 90% | Unix (LF) | UTF-8

gestor.asl

Archivo Editar Ver

```
// Plan para realizar el pedido al almacén
+!realizar_pedido_optimo
  : pedido_optimo(Q) & Q > 0
  <- .print("Realizando pedido de ", Q, " unidades");
     .send(almacen, tell, pedido(Q));
     .print("Esperando confirmación...").

+!realizar_pedido_optimo
  : pedido_optimo(0)
  <- .print("No se necesita realizar pedido este período").

// Manejo de confirmaciones/rechazos
+confirmar_pedido(Cantidad)[source(almacen)]
  <- .print("Pedido confirmado por almacén: ", Cantidad).

+rechazar_pedido(Cantidad)[source(almacen)]
  <- .print("Pedido rechazado por almacén: ", Cantidad);
     !recalcular_pedido.
```

Ln 49, Col 29 | 2.242 caracteres. | 90% | Unix (LF) | UTF-8

EJEMPLO DE EJECUCIÓN Y RESULTADOS

Supón que el agente detecta que el stock actual es bajo y la demanda esperada para el próximo periodo es alta. El agente:

1. Consulta el stock actual (por ejemplo, 10 unidades).
2. Predice que la demanda será de 20 unidades.
3. Calcula que debe pedir 15 unidades para mantener un stock de seguridad y minimizar costos.
4. Envía la orden de pedido.
5. Cuando el pedido se recibe, actualiza el stock y repite el proceso en el siguiente ciclo.

VENTAJAS DE USAR AGENTES PARA ESTE TIPO DE PROBLEMAS

- Automatización: Los agentes pueden operar sin intervención humana, reduciendo errores y tiempos de respuesta.
- Adaptabilidad: Pueden ajustar sus decisiones en tiempo real ante cambios en la demanda o en el entorno.
- Escalabilidad: Es posible gestionar múltiples productos o almacenes simultáneamente, cada uno con su propio agente.
- Optimización continua: Los agentes pueden incorporar técnicas avanzadas para mejorar sus decisiones a medida que reciben más datos.

DIAGRAMA DE INTERACCIÓN

El diagrama presentado ilustra la arquitectura y el flujo de información entre dos agentes principales en un sistema de gestión de inventarios: Almacén y Gestor. Cada agente tiene funciones, creencias y acciones específicas, y se comunican entre sí para optimizar la gestión de pedidos y el control de stock. A continuación, se explica cada componente y la dinámica general del sistema:

AGENTE ALMACÉN

El Almacén es responsable de mantener y actualizar la información sobre el inventario disponible. Sus principales creencias y acciones son:

- stock_max(0): Define la capacidad máxima del almacén.
- stock_current(Q): Indica la cantidad actual de inventario disponible.
- coste_almacenaje(costo): Representa el costo asociado al almacenamiento de productos.
- coste_pedido(costo): Costo fijo por realizar un pedido.
- coste_unidad(costo): Costo por cada unidad almacenada o pedida.

Acciones del Almacén:

- consultar_stock: Permite consultar el nivel de inventario.
- actualizar_stock(cantidad): Actualiza la cantidad de inventario tras recibir o despachar productos.

Comunicación:

- El almacén puede recibir solicitudes de pedido del gestor y responde confirmando o rechazando el pedido según la disponibilidad.

AGENTE GESTOR

El Gestor es el encargado de tomar decisiones estratégicas sobre cuándo y cuánto pedir, basándose en la información recibida del almacén y en parámetros de optimización.

- costo(q,s,coste): Calcula el costo total de una política de inventario dada.
- stock_seguridad(Q): Define el nivel de stock de seguridad para evitar rupturas.
- demanda_esperada(Q): Estima la demanda futura.

Acciones del Gestor:

- realizar_pedido_optimo(q): Ejecuta el pedido óptimo calculado.
- calcular_mejor_pedido(s,ss,max): Determina la mejor política de pedido considerando el stock actual, el stock de seguridad y la capacidad máxima.

- gestionar_inventario(almacen): Supervisa y controla el inventario del almacén.
- start(self): Inicializa el gestor.

Comunicación:

- El gestor envía solicitudes de pedido al almacén y espera confirmación o rechazo.
- Utiliza la información de costos y demanda para optimizar las decisiones de inventario.

INTERACCIÓN ENTRE ALMACÉN Y GESTOR

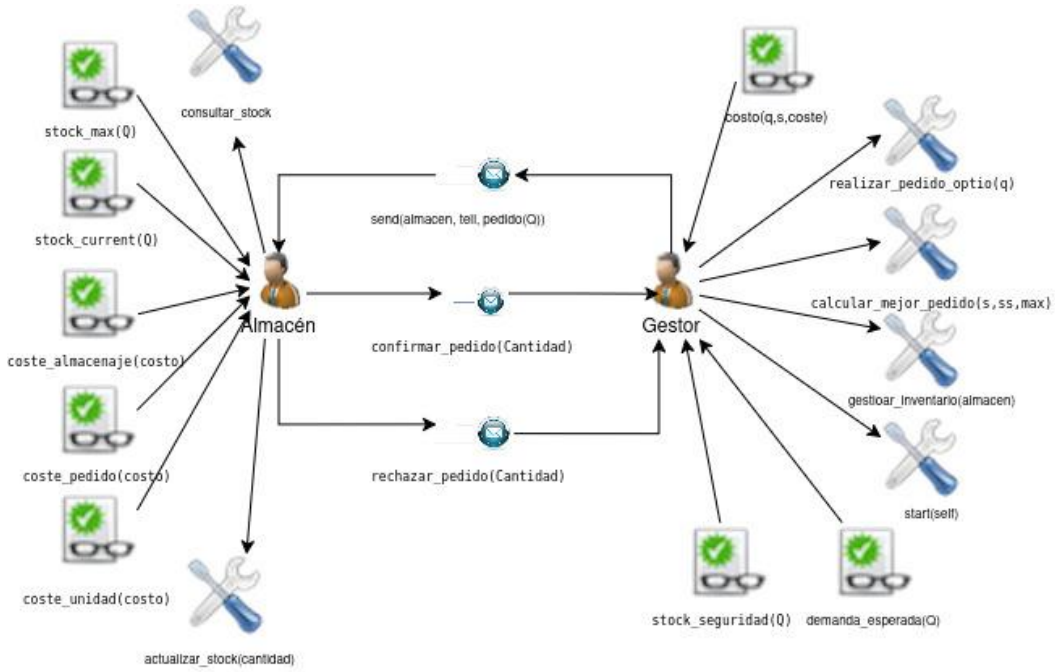
El flujo de información y acciones entre ambos agentes es el siguiente:

1. El Gestor consulta el stock al almacén para conocer la cantidad disponible.
2. El Gestor calcula el pedido óptimo usando la información de stock, demanda esperada y costos.
3. El Gestor envía un pedido al almacén (send(almacen, tell, pedido(Q))).
4. El Almacén evalúa el pedido y responde:
    - confirmar_pedido(Cantidad): Si hay suficiente stock, confirma el pedido.
    - rechazar_pedido(Cantidad): Si no hay suficiente stock, rechaza el pedido.
5. El Almacén actualiza el stock tras la confirmación del pedido.

PROPÓSITO DEL SISTEMA

Este sistema permite automatizar y optimizar la gestión de inventarios mediante la colaboración de dos agentes inteligentes. El **Gestor** se encarga de la toma de decisiones estratégicas, mientras que el **Almacén** ejecuta las operaciones físicas y mantiene la información actualizada. La comunicación entre ambos asegura que las decisiones se basen en datos actualizados y que se minimicen los costos asociados al inventario.

DIAGRAMA


consultar_stock
stock_max(Q)
stock_current(Q)
coste_almacenaje(costo)
coste_pedido(costo)
coste_unidad(costo)
actualizar_stock(cantidad)
Almacén
send(almacen, tell, pedido(Q))
confirmar_pedido(Cantidad)
rechazar_pedido(Cantidad)
Gestor
costo(q,s,coste)
realizar_pedido_optio(q)
calcular_mejor_pedido(s,ss,max)
gestioar_Inventario(almacen)
start(self)
stock_seguridad(Q)
demanda_esperada(Q)

CONCLUSIONES

La programación dinámica es una herramienta fundamental para la gestión óptima de inventarios, ya que permite tomar decisiones secuenciales considerando la evolución de la demanda, los costos y las restricciones a lo largo del tiempo. Su principal ventaja radica en la capacidad de descomponer problemas complejos en etapas más simples, facilitando la obtención de políticas de inventario que minimizan los costos totales y mejoran la eficiencia operativa. Además, la programación dinámica ofrece grandes ahorros computacionales frente a métodos de enumeración exhaustiva, lo que la hace especialmente útil en problemas de gran escala o con múltiples variables. Sin embargo, su aplicación requiere una formulación cuidadosa de las relaciones recursivas y una adecuada definición de los estados y decisiones en cada etapa. En resumen, los modelos de inventarios basados en programación dinámica contribuyen significativamente a la toma de decisiones estratégicas en las organizaciones, permitiendo adaptarse a la incertidumbre y a la variabilidad del entorno empresarial.

BIBLIOGRAFÍA

[1] Valencia-Cárdenas, M., Díaz-Serna, F. J., & Correa-Morales, J. C. (2015). Planeación de inventarios con demanda dinámica. Una revisión del estado del arte. *Dyna, 82*(190), 137-146.http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0012-73532015000200023

[2] Cano Bedoya, J., & otros. (2022). Programación dinámica aplicada al análisis de inventario para atender órdenes de pedido. *ResearchGate*.https://www.researchgate.net/publication/367348525_Programacion_dinamica_aplicada_al_analisis_de_inventario_para_atender_ordenes_de_pedido

[3] Universidad Nacional de Colombia. (2016). Modelación dinámica para la optimización de inventarios multiproducto con demanda multivariada. *CLADEA*.https://cladea.org/wp-content/uploads/1634/55/2016_Paper.pdf

[4] Rodríguez, M., Salazar, F., & González, J. (2018). Un modelo para el control de inventarios utilizando dinámica de sistemas. *Estudios de la Gestión: Revista Internacional de Administración*, 6, 121-135.https://www.academia.edu/42605919/Un_modelo_para_el_control_de_inventarios_utilizando_din%C3%A1mica_de_sistemas

[5] García-Pacheco, M. C., & San Andrés-Laz, E. M. (2021). Diseño de un sistema de gestión por procesos para el manejo de inventarios. Caso: Ferretería Quiroz. *Revista Científica Multidisciplinaria Arbitrada "Yachasun"*, 5(9), 180–204.http://dx.doi.org/10.46296/yc.v5i9edespsoct.0118

[6] Vélez Vélez, S. M., & Pazmiño Linares, S. A. (2022). Importancia de los sistemas de inventarios en las organizaciones a través de una revisión bibliográfica. *AlfaPublicaciones*, 4(1.1), 342–357.http://dx.doi.org/10.33262/ap.v4i1.1.163

[7] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

[8] Winston, W. L. (2004). *Operations Research: Applications and Algorithms* (4th ed.). Thomson Brooks/Cole.

[9] Colaboradores de Wikipedia. (2024, 24 diciembre). Programación dinámica. Wikipedia, la Enciclopedia Libre. https://es.wikipedia.org/wiki/Programaci%C3%B3n_din%C3%A1mica

[10] Kariuki, C. (2024, 17 mayo). Programación dinámica: Qué es, cómo funciona y recursos de aprendizaje. Geekflare Spain. https://geekflare.com/es/dynamic-programming/