Inventory Simulation Project Report

Leismael Sosa Hernández, Alejandro Yero Valdéz

March 28, 2024

1 Introducción

1.1 Descripción del proyecto

Nuestro proyecto consiste en desarrollar una simulación basada en eventos discretos para analizar el comportamiento de un sistema de inventario. El sistema de inventario analizado presenta las siguientes entidades:

1. Tienda: Esta entidad representa una tienda que vende productos de un solo tipo.
2. Proveedor: Esta entidad le vende productos a la tienda cuando esta lo pide.
3. Cliente: Esta entidad representa a los compradores del producto ofrecido por la tienda.

Las entidades anteriores se relacionan de la siguiente forma. La tienda tiene en cada momento de tiempo una cierta cantidad del producto disponible para vender y siempre que esta cantidad baje de cierto umbral le ordena unidades adicionales al proveedor. La política de pedido que usa el cliente, llamada (s, S), plantea que siempre que la cantidad de unidades del producto en el almacén (inventario), denotada por x, sea menor que s ($x < s$) se debe pedir al proveedor la cantidad $S - x$.
Se tiene que el costo de pedir y unidades del producto es una función especificada, denotada por $c(y)$, y toma L unidades de tiempo el envío, con el pago realizado en el momento de la entrega.
También se tiene que la tienda paga un costo por mantener el inventario. Este costo es de h (un valor fijo) por unidad de artículo por unidad de tiempo. Además, se tiene que siempre que un cliente demanda más del producto de lo que está disponible actualmente, entonces se vende la cantidad a mano y el resto del pedido se pierde para la tienda.

1.2 Objetivos y metas

El objetivo de la simulación es poder recoger un conjunto de estadísticas sobre como se comporta el sistema al variar ciertos parámetros y también optimizar

los mismos para mejorar algunas de estas estadísticas.
Las estadísticas de interés se sacan en un plazo de tiempo fijo (puede ser 1 día de trabajo, 1 semana, etc.) y algunas de estas son las siguientes:

- Distribución en la linea de tiempo de la cantidad de clientes que llegaron a la tienda por hora.
- Distribución en la linea de tiempo de la cantidad de pedidos de productos por hora vs la cantidad de productos que se pudieron vender (útil para detectar horas picos y ver gráficamente cuantos pedidos se perdieron por hora).
- Razón entre la cantidad total de productos pedidos por clientes en un plazo de tiempo y la cantidad total de estos que se vendieron (recordar que puede que no se tengan los suficientes productos para satisfacer la demanda).
- Distribución en la linea de tiempo (por horas) de cuanto se pagó por el mantenimiento del inventario.
- Costo total (en el plazo de tiempo dado) del mantenimiento del inventario
- Distribución en la linea de tiempo de como fue variando la cantidad de productos en el inventario
- Costo total (en el plazo de tiempo dado) de los pedidos al proveedor.
- Balance de la cuenta de la tienda al finalizar la simulación.

1.3 Variables que describen el problema

Las variables con las que inicializamos la simulación son las siguientes:

- $s \rightarrow$ Entero que representa la cantidad mínima del producto que se puede tener en inventario de la tienda antes de solicitar un pedido al proveedor.
- $S \rightarrow$ Entero que representa la cantidad máxima del producto que se puede tener almacenado en el inventario de la tienda.
- $initial\_inventory \rightarrow$ Entero que representa la cantidad de producto que se tiene inicialmente en el inventario.
- $ordering\_cost\_function \rightarrow$ Función que recibe un entero x y representa el costo de pedirle al proveedor x unidades del producto.
- $lead\_time \rightarrow$ Entero que representa el tiempo que se demora en llegar el producto por parte del proveedor.
- $holding\_cost\_rate \rightarrow$ Entero que representa la constante de pago por mantener el inventario.

- *product_value* → Entero que representa cuanto debe pagar un cliente por obtener el producto.
- *client_arrival_dist* → Función que devuelve el tiempo que se demora en aparecer el próximo cliente. Por defecto esta función representa la distribución de Poisson.
- *client_demand_dist* → Función que devuelve la cantidad de producto que pedirá el próximo cliente. Por defecto esta función representa una distribución uniforme.

2 Detalles de implementación

Para implementar esta simulación seleccionamos el lenguaje Python debido al ecosistema de librerías que proporciona y los pasos que seguimos fueron los siguientes:

- Desarrollamos 3 clases que modelan cada uno de los eventos de nuestra simulación. Estos son:
  - **Sell** → Este evento representa el arribo de un cliente a la tienda pidiendo un producto. Tiene 2 propiedades:
    - *time* → El tiempo en el que el cliente llega a la tienda y realiza el pedido.
    - *amount* → La cantidad de unidades que quiere comprar el cliente.
  - **SupplyArrival** → Este evento representa la llegada de productos a la tienda por parte del proveedor. Tiene 2 propiedades:
    - *time* → El tiempo en el que el envío llega a la tienda.
    - *amount* → La cantidad de unidades recibidas.
  - **SimulationEnd** → Este evento representa el fin de la simulación y su único objetivo es detener esta al procesar el evento. Tiene como propiedad el tiempo (*time*) en el que se debe detener la simulación.
- Desarrollamos 4 clases para representar registros de datos. Estos son necesarios a la hora extraer información estadística de la simulación. Estas son:
  - **SellRegistry** → Este registro guarda información sobre las ventas a los clientes. Sus propiedades son:
    - *time* → El tiempo en el que ocurrió la venta.
    - *amount_asked* → Representa la cantidad de unidades del producto que el cliente quiere comprar.
    - *amount_seeled* → Representa la cantidad de unidades del producto que la tienda es capaz de venderle al cliente (depende del inventario en ese momento).

- **StockRegistry** Este registro guarda las variaciones de la cantidad de productos en el almacén. Cuando se realiza una venta de productos, o cuando se reciben productos del proveedor, es cuando se crea una entrada en este registro. Sus propiedades son:
  - *time* → Representa el tiempo en el que el almacén varió la cantidad de productos disponibles.
  - *amount* → Representa la cantidad de productos en el almacén.
- **BuyRegistry** → Este registro guarda los momentos en los que la tienda realiza un pedido de productos al proveedor. Tiene las siguientes propiedades:
  - *time* → Representa el tiempo en el que la tienda realiza el pedido.
  - *amount* → Representa la cantidad de productos que pide la tienda.
- **BalanceRegistry** → Este registro guarda la información referente al balance de la cuenta de la tienda a lo largo del tiempo. En nuestra simulación se asume que la tienda siempre tiene dinero para pagar, por lo que este registro realmente representa cuanto ha ganado o perdido la tienda desde el inicio de la simulación. Si el balance es bajo 0 la tienda está perdiendo dinero y si es mayor que 0 la tienda está ganando. Las propiedades de esta clase son:
  - *time* → Representa el tiempo en el que ocurrió un cambio de balance en la cuenta de la tienda.
  - *balance* → Representa el valor del balance actual.

- Desarrollamos la clase **InventorySimulation** que se encarga de correr la simulación de inventario. Esta clase genera los eventos mencionados anteriormente en el documento y los procesa. Cada vez que procesa los eventos se encarga de actualizar los registros pertinentes.

- Desarrollamos la clase **Statistics** que se encarga de tomar los resultados guardados en los registros de una corrida de la simulación y los procesa para sacar estadísticas que muestran de forma general informaciones sobre las ventas de productos, compras al proveedor, variaciones del inventario y también información sobre el balance de la cuenta de la tienda.

Todo lo anterior es lo que creamos para analizar el comportamiento de nuestro modelo del sistema del Inventario y sacar informaciones estadísticas del mismo. Pero además de esto nosotros también creamos una clase que se encarga de optimizar los resultados de algunas de las estadísticas mas importantes de la simulación. Esto lo hace modificando los parámetros de la política (s,S) de la simulación. Esta clase se llama **Optimizer** y usa el algoritmo de Hill Climbing para llevar a cabo la optimización.

3 Modelo Matemático

Una tienda tiene un almacén donde guarda un tipo de producto y los vende a un precio p por unidad. La llegada de los clientes a la tienda se simula con una distribución *Poisson* y la cantidad de unidades pedida es simulada con una función que distribuye G. La tienda usa una política de pedido (s, S), donde $s < S$, tal que si la cantidad x de unidades del producto en el almacén cumple que $x < s$ se tiene que la tienda pide al proveedor una cantidad $S - x$ de unidades del producto. Se tiene que el costo de pedir y unidades del producto al proveedor es una función $c(y)$ y que la entrega del producto se demora L unidades de tiempo, donde L es constante. También hay un costo por el almacenamiento de los productos, este es de h (constante) por unidad de producto por unidad de tiempo, donde la unidad de tiempo en este caso es 1 hora.

Las variables con las que contamos en la simulación se pueden dividir en las siguientes categorías:

- **Variables de tiempo**
  - **t** → El tiempo global de la simulación.
- **Variables de Registros**: Estas variables guardan información útil para análisis estadístico.
  - **SellRegistry** → Este registro guarda información sobre las ventas a los clientes.
  - **StockRegistry** → Este registro guarda información sobre las variaciones de la cantidad de productos en el almacén.
  - **BuyRegistry** → Guarda información sobre los momentos en los que la tienda paga un pedido de productos al proveedor.
  - **BalanceRegistry** → Guarda la información referente al balance de la cuenta, es decir, cuanto a ganado o perdido en cada momento.
  - **PayHoldingRegistry** → Guarda la información de cada pago realizado por la tienda para mantener el inventario.
- **Variables del estado del sistema**: Variables que muestran el estado del sistema en el tiempo t
  - **actual_balance** → Representa el balance de la cuenta de la tienda.
  - **actual_inventory** → Representa la cantidad de unidades del producto que se tiene en el almacén.
  - **pending_order** → Dice si hay un pedido pendiente o no.
- **Lista de eventos**
  - **Sell** → Este evento representa el arribo de un cliente a la tienda pidiendo un producto. Tiene 2 propiedades:

    - $time \rightarrow$ El tiempo en el que el cliente llega a la tienda y realiza el pedido.
    - $amount \rightarrow$ La cantidad de unidades que quiere comprar el cliente.
  - **SupplyArrival** $\rightarrow$ Este evento representa la llegada de productos a la tienda por parte del proveedor. Tiene 2 propiedades:
    - $time \rightarrow$ El tiempo en el que el envío llega a la tienda.
    - $amount \rightarrow$ La cantidad de unidades recibidas.
  - **PayHolding** $\rightarrow$ Este evento representa la orden de pagar por el servicio de mantener el inventario. Tiene solamente la propiedad **time** para representar el momento en el que se debe efectuar el pago.
  - **SimulationEnd** $\rightarrow$ Este evento representa el fin de la simulación y su único objetivo es detener esta al procesar el evento. Tiene como propiedad el tiempo ($time$) en el que se debe detener la simulación.

- **Inicialización**
  - **t** $= 0$
  - **actual_balance** $= 0$
  - **actual_inventory** = cantidad inicial en el almacén
  - **actual_inventory** = false
  - Se verifica la política (s, S) y si se debe pedir al proveedor se agrega a la cola de eventos el evento **SupplyArrival** en el tiempo $t + L$ y la cantidad $S - x$, donde x es la cantidad actual en inventario.
  - Se genera el tiempo de arribo del próximo cliente, t_c, por la distribución Poisson. Se genera el la cantidad de unidades que pedirá, **amount**, por la distribución G. Se crea un evento **Sell** asignándole las anteriores variables y se agrega este evento en la cola de eventos.
  - Si el espacio de tiempo en el que se tiene que realizar el pago por mantener productos en el almacén está dado por la variable t_a, entonces se debe crear un evento **PayHolding** para el tiempo dado por $t + t_a$.
  - Se crea el evento **SimulationEnd** guardando el tiempo en el que debe terminar la simulación y este se agrega a la cola de eventos.

- **Ciclo de la simulación** Se ejecuta mientras haya un evento en la cola. Aquí se extrae el evento con menor prioridad.
  - Se guarda en la variable **E** el evento extraído de la cola de eventos.
  - Se actualiza el tiempo t de la simulación al tiempo del evento **E**.
  - En dependencia del tipo de evento que represente **E** se cae en uno de 3 posibles casos. Estos se analizan más adelante en el documento.

- Al finalizar de procesar el evento se verifica el nivel del inventario y si cumple con la política (s, S) y además se tiene que no hay un pedido pendiente, es decir, que la variable **pending_order** es **false**, se genera un evento **SupplyArrival** con tiempo $t_a = t + L$, (donde L es el tiempo que se demora en llegar el envío del proveedor) y con la cantidad S - **actual_inventory**. Luego se agrega este evento a la cola de eventos y se actualiza la variable **pending_order** a **true**.
- Luego se genera con la distribución Poisson un nuevo t_c que es el tiempo que se demorará en aparecer el próximo cliente. Se genera con la distribución G un nuevo valor para **amount**, es decir, cuantas unidades ordenará este próximo cliente. Se crea un nuevo evento **Sell** con los valores $t + t_c$ para el tiempo y **amount** para la cantidad de unidades que pedirá. Luego se agrega a la cola de eventos.
- Se actualiza el registro **BalanceRegistry** guardando los valores t en la variable **time** y el **actual_balance** en la variable **balance**.
- Se actualiza el registro **StockRegistry** guardando el tiempo actual de la simulación, dado por la variable t y la cantidad de unidades del producto en el almacén, dado por la variable **actual_inventory**.

- **Caso 1 - El evento E es de tipo Sell**
  - Se extrae la cantidad de unidades que desea comprar el cliente en la variable **amount**.
  - Se calcula cuanto producto se puede vender al cliente con la fórmula **min(actual_inventory, amount)** y se guarda en **sell_amount**.
  - Se actualiza la variable **actual_balance** a **actual_balance**+= **p** · **sell_amount** · **amount**, donde **p** es el precio del producto.
  - Se actualiza la variable **actual_inventory** a **actual_inventory - sell_amount**.
  - Se actualiza el registro **SellRegistry** guardando el tiempo actual de la simulación, dado por la variable t, la cantidad pedida, dada por **amount** y la cantidad que el almacén pudo vender, dado por **sell_amount**.

- **Caso 2 - El evento E es de tipo SupplyArrival**
  - Se extrae la cantidad de unidades que se recibió por el proveedor en la variable **amount**.
  - Se actualiza la variable **actual_inventory** a **actual_inventory** + **amount**.
  - Si se recibieron y unidades, entonces se calcula el costo por los productos recibidos con la función de costo de envío $c(y)$ y se almacena en la variable **cost**. Luego se actualiza la variable **actual_balance** a **actual_balance** - **cost**.

- Se actualiza la variable **pending_order** a **false**.
- Se actualiza el registro **BuyRegistry** guardando el tiempo actual de la simulación, dado por la variable t, y se guarda el número de unidades recibido, dado por la variable **amount**.

- **Caso 3 - El evento E es de tipo PayHolding**
  - Se calcula el costo por mantener el inventario y se guarda en **cost**.
  - Se actualiza la variable **actual_balance** a **actual_balance - cost**.
  - Se actualiza el registro **PayHoldingRegistry** guardando el tiempo en el que se realizó el pago con el tiempo actual de la simulación, dado por la variable t, y guardando el costo de este pago, almacenado en **cost**.
  - Se genera un nuevo evento **PayHolding** para el tiempo $t + t_a$, donde t_a es el tiempo que hay entre pagos por el mantenimiento del almacén. Luego se agrega este evento a la cola de eventos.

- **Caso 4 - El evento E es de tipo SimulationEnd** $\rightarrow$ En este caso simplemente se detiene la simulación, es decir, se sale del ciclo.

Lo descrito anteriormente es la formalización del modelo de la simulación, pero hay algunos puntos a aclarar.

- En nuestro modelo se asume que la tienda siempre tiene dinero para pagar, de donde que el **balance** de la tienda sea **negativo** no significa que la tienda esté sin dinero, solamente significa que la tienda está perdiendo dinero.

- Se asume que el tiempo de atención al cliente es instantáneo. Por esto en nuestra simulación ningún cliente debe esperar en una cola a ser atendido. Además, si hay mas de 1 cliente en la tienda al mismo tiempo (algo posible), se asume que la tienda atiende a todos los clientes de forma instantánea (en el mundo esta es la tienda mas eficaz en este aspecto, de hecho, la única) y se agrega al registro de ventas una lista de ventas en ese mismo tiempo.

- Los eventos generados se agregan en una cola de prioridad (donde el menor elemento sale primero). La prioridad está dada por el tiempo en el que se debe procesar el evento. En caso de que más de 1 evento tenga lugar en el mismo espacio de tiempo, se desambigua por la prioridad que se tiene entre eventos. Esta se muestra a continuación.

  1. **SimulationEnd**
  2. **SupplyArrival**
  3. **Sell**
  4. **PayHolding**

4 Resultados y experimentos

Los resultados estadísticos y los experimentos realizados al aplicarle una optimización local a la simulación, se encuentran en el notebook del repositorio.