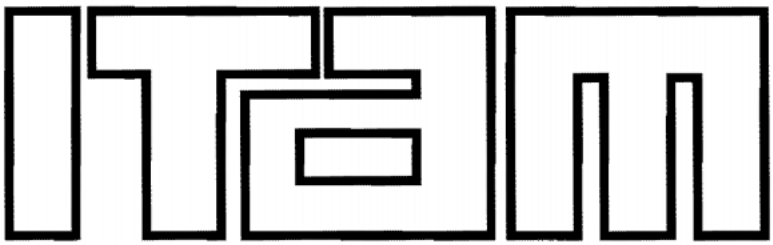
INSTITUTO TECNOLÓGICO AUTÓNOMO DE MÉXICO



Aprendizaje Reforzado para el Juego de la Distribución de Cerveza

T E S I S

QUE PARA OBTENER EL TÍTULO DE

MAESTRA EN CIENCIA DE DATOS

P R E S E N T A

MARÍA FERNANDA ALCALÁ DURAND

ASESOR: DR. ADOLFO JAVIER DE UNÁNUE TISCAREÑO

CIUDAD DE MÉXICO 2021

Con fundamento en el artículo 21 y 27 de la Ley Federal del Derecho de Autor y como titular de los derechos moral y patrimonial de la obra titulada "**Aprendizaje Reforzado para el Juego de Distribución de Cerveza**", otorgo de manera gratuita y permanente al Instituto Tecnológico Autónomo de México y a la Biblioteca Raúl Bailléres Jr. autorización para que fijen la obra en cualquier medio, incluido el electrónico y la divulguen entre sus usuarios, profesores, estudiantes o terceras personas, sin que pueda percibir por la divulgación una contraprestación

María Fernanda Alcalá Durand

_____

FECHA

_____

FIRMA

Agradecimientos

Esta no es la forma final de los agradecimientos. Pero a grandes rasgos,

Mi mamá, mi novio, mi familia y mi asesor.

Amigos que me han ayudado sustancialmente: Paris Méndez, Jan Vlachy, Felipe Gerard, Laila Wahedi.

Abstract

Este trabajo de Tesis propone representar el conocido *Juego de la Distribución de Cerveza* como un modelo multiagente de cadena de suministro, añadiendo una restricción tal que se aproxime más a un problema del mundo real: la producción de materia prima es finita y solamente ocurre en un periodo específico. El acercamiento elegido es la aplicación de dos métodos de aprendizaje reforzado: *Policy Iteration* y *Q-learning*; ambas técnicas encuentran estrategias aplicables para todos los agentes. Se demuestra que con *Policy Iteration* se encuentra un conjunto de políticas que les reportan mayores utilidades que algunas estrategias básicas, bajo un esquema de recompensas y castigos. Por otro lado, *Q-learning* demuestra ser un algoritmo pesado computacionalmente y, aunque proporciona estrategias para los agentes, no puede responder a cambios tan rápidamente como *Policy Iteration*.

Se concluye que el aprendizaje reforzado no solamente es eficiente, sino suficientemente flexible como para identificar cambios permanentes en la demanda y adaptar las consecuentes políticas de compra. Asimismo, se muestra evidencia de que para cada agente en la cadena, es preferible seguir estrategias optimizadas independientemente de las decisiones de los demás agentes. Finalmente, se explora la posibilidad de que en subsecuentes trabajos, se añadan estrategias diferentes o restricciones adicionales al *Problema de la Distribución de la Cerveza* para obtener modelos más fieles a las condiciones del mundo externo y, por lo tanto, utilizables en un contexto real.

Términos clave Juego de la Distribución de Cerveza, Aprendizaje Reforzado, Iteración de Política, Q-aprendizaje

Índice general

Capítulo 1

Introducción

Work is the curse of the drinking classes.

Oscar Wilde

1.1. Objetivo y presentación del problema

Una cadena de suministro, según Sterman [15], es el conjunto de estructuras y y procesos que una organización utiliza para proporcionarle un producto a un cliente. Este producto puede ser tangible (como un automóvil) o intangible (como un servicio). Sin embargo, esto es una sobresimplificación, dado que cualquier cadena en el mundo real tiene muchos más factores. Para ser completamente realista, un sistema de simulación de tal cadena de suministro contemplaría todas las variables presentes en su entorno: desde aquellas intrínsecas al proceso como los costos de almacenamiento y los tiempos de espera para el cumplimiento de las órdenes, hasta externas como el precio de bienes sustitutos y complementarios; o aquellas más complejas, como dinámicas entre distintos agentes, tal que los consumidores cambien de distribuidor después de cierto número de órdenes no cumplidas.

Sin embargo, es imposible replicar todas las condiciones de un sistema tan complejo como la economía en el mundo real, e incluso es difícil incorporar un alto número de variables al tratar de optimizar un problema de este calibre. También es posible que incluso después de plantear un

sistema tal con una cantidad manejable de condiciones, este sea tan complejo o falto de linearidad, que no exista una solución analítica y sea necesario utilizar herramientas más sofisticadas.

Por esta y otras razones, los problemas económicos a resolver matemática o programáticamente deben hacer ciertos supuestos acerca de las condiciones que los rigen. Para asegurar la aplicabilidad de las soluciones, es necesario asegurar un equilibrio entre apego a la realidad (más variables, menos supuestos) y control de complejidad (más supuestos, menos variables).

Tomemos como ejemplo el Juego de la Distribución de Cerveza, un famoso problema académico de cadena de suministro. En la forma estándar del problema (que será detallada más adelante) comúnmente se supone que uno de los agentes que forman parte de la cadena, la fábrica, tiene acceso a las materias primas para producir cerveza en cualquier momento del año y controla el calendario de producción (un ejemplo se puede encontrar en Sterman [15]). Sin embargo, los datos reales de producción anual de cebada, nos muestran que este es un supuesto poco razonable.

Por otro lado, en dicha forma estándar también se supone que la demanda del consumidor es constante (salvo un salto único en el cual el consumidor comienza a requerir el doble del inventario, y mantiene esta nueva demanda por el resto del juego). De nuevo, los datos en el mundo real nos muestra que las personas en EUA beben mucha más cerveza cerca y durante las festividades de fin de año. En México parece pertinente también considerar un pico de consumo el día de la Independencia, y en una versión aún más fidedigna de la tendencia de demanda, porían incluirse eventos deportivos. Esto de nuevo nos muestra que el supuesto estándar de demanda del consumidor es poco razonable.

En el presente trabajo, se pretende ayudar a responder una pregunta específica que proviene de la brecha causada por tales supuestos de oferta y demanda, que la autora juzga poco razonables o coherentes con el mundo real. De esta manera, se podría generalizar el problema a tendencias de consumo complejas y producción por temporadas.

La hipótesis con la que se trabajará se puede delinear de la siguiente manera:

'Bajo este nuevo conjunto de supuestos, es posible encontrar estrategias óptimas para todos los agentes tomadores de decisiones en el Juego de la Distribución de Cerveza, por medio de algoritmos de aprendizaje reforzado, que produzcan resultados en un tiempo suficientemente veloz como para poder accionarlas.'

1.2. Metodología y resultado esperado

Dado que el objetivo del Juego de la Distribución de Cerveza para cada agente (menudeo, mayoreo, almacén regional, fábrica) es minimizar su costo, típicamente las soluciones se plantean como aquellas para un problema de optimización (Sterman [15]). Sin embargo recientemente se ha explorado la solución por medio de distintos métodos, desde algoritmos genéticos (Strozzi et al. [16]) hasta aprendizaje de máquina reforzado (Chaharsooghi et al. [3]). En el presente trabajo se explorarán dos algoritmos del segundo tipo: *policy iteration* y *Q-learning*. Además, se utilizará una demanda de consumidor que refleje tendencias del mundo real y se removerá el supuesto relacionado a la producción constante en los campos, ya que de esta manera se obtiene un modelo más realista respecto a los ciclos naturales de cosecha de las materias primas.

1.3. Organización de la tesis

Este trabajo está organizado en ocho capítulos. El primero, del cual forma parte esta sección, proporciona una vista general y la estructura del texto.

Las secciones dos y tres establecen el contexto necesario para el problema a resolver; mientras que la cuarta muestra acercamientos previos a la solución, dentro de los cuales ya existen propuestas con los algoritmos presentes en este trabajo. La quinta sección contiene una idea nueva para mejorar tales algoritmos conviertiendo el problema en uno más cercano a la realidad: removiendo el supuesto de producción ilimitada en los campos.

La sección seis contiene la estructura del producto de datos construido para guardar los resultados y las principales diferencias necesarias dada la forma diferente de la solución dependiendo del algoritmo. La sección siete contiene los resultados de este trabajo y una comparación entre ellos.

Por último, la octava sección contiene las conclusiones y posibles preguntas para investigaciones futuras en esta línea.

Capítulo 2

El Problema: Juego de la Distribución de Cerveza

El *Juego de la Distribución de Cerveza* ejemplifica la relación causal entre la toma de decisiones de cada agente (que no tiene información global) con el comportamiento de todo el sistema, en este caso una cadena de suministro. Asimismo, a falta de una estrategia óptima, inherentemente presenta el efecto látigo: el retraso en la información entre agentes causa que, a través del tiempo, el comportamiento de cada uno se vuelva menos constante cuanto más lejos se encuentre del consumidor final. Por último, evidencia las ineficiencias inherentes a tratar de resolver el problema enfocándose en los agentes, ignorando que es un sistema completo.

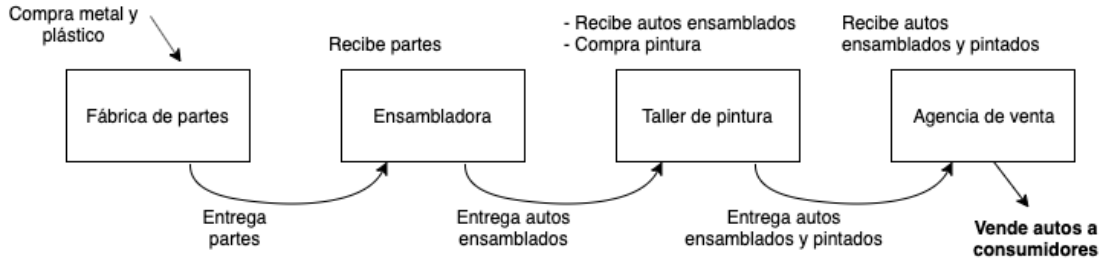
2.1. Cadenas de Suministro

Para plantear correctamente el *Problema de la Distribucińn de Cerveza*, es necesario entender el concepto de una cadena de suministro.

Para construir sobre la definición de Sterman [15] descrita en el capítulo anterior, una cadena de suministro (en inglés, *supply chain*), según Jacobs y Chase [9], es un proceso que desplaza información y material con destino y origen en los procesos de manufactura y servicio de la empresa. Una

manera común de representar el comportamiento de una cadena como un modelo es a través de existencias, flujos, ciclos y agentes[1]. Por ejemplo, en una simplificación de producción automotriz que se puede observar en la figura 2.1, se puede observar un ejemplo en el cual se representa cada eslabón como un agente (la fábrica de partes, la ensambladora, el taller de pintura y la agencia de venta), los flujos de producto como las entradas y salidas de materia en cada eslabón (por ejemplo, la fábrica de partes tiene como insumos metal y plástico, mientras que el taller de pintura tiene como insumo el auto ensamblado y litros de pintura), los flujos de información se suponen globales y sin retraso, y los ciclos pueden depender de estacionalidad, disponibilidad o simple predisposición.

Figura 2.1: *Ejemplo de simplificación de una cadena de suministro automotriz*



El estudio de las cadenas de suministro puede cubrir un campo vasto, dado que existen una gran cantidad de problemas relacionados a ellas: transporte, logística, manejo de inventario, optimización de la localización geográfica para cada uno de los eslabones, sustentabilidad, etc. Sin embargo, una vez que la cadena está en funcionamiento, uno de las principales dificultades es que los agentes encargados de optimizar las estrategias de demanda y producción de cada eslabón solamente pueden tomar decisiones "dentro" de aquel en el que se encuentran, y no tienen información más allá de los eslabones inmediatamente conectados, solamente una estimación (en inglés *forecast*) que representa su mejor predicción de tal comportamiento. Así, la información acerca de la demanda del consumidor se va diluyendo en cada nivel, además de que las decisiones tomadas tienen repercusiones más allá del futuro inmediato.

[1]Existen muchas otras maneras, como las redes de Petri, ampliamente utilizadas en la teoría de autómatas. (Shiflet y Shiflet [13])

El objetivo principal de cada agente es minimizar los costos al tiempo de maximizar los ingresos, donde los costos pueden no ser flujos de dinero sino, por ejemplo, castigos por órdenes no cumplidas o varianza alta en la producción que implique mayores costos de mantenimiento. Para llegar a este objetivo, cada uno de ellos debe tratar de inferir el patrón de demanda del consumidor, que llegará distorsionado a él, por medio de información local bastante restringida. Al tener una estimación útil de tal patrón, así como del comportamiento de sus eslabones inmediatamente conectados, el agente puede crear su estrategia de inventario y producción para maximizar su utilidad. Volviendo al ejemplo anterior de la producción automotriz, la fábrica de partes debe ordenar metal y plástico suficiente para producir y cubrir la demanda de la planta ensambladora, pero ambos eslabones deben producir manteniendo en la mente que la tendencia de demanda proviene, al final, del consumidor. Sin embargo, la planta no tiene ningún incentivo real para compartir con la fábrica, la cantidad exacta de autos ensamblados que produce o que vende cada periodo al taller de pintura. Esto obliga a cada eslabón a contar solamente con datos limitados, además de que los datos de demanda que reciben obedecen al tiempo real y los agentes no tienen la oportunidad de repetir experimentos.

Un modelo computacional que se comporte suficientemente parecido al mundo real, en el que todos los demás eslabones tomen estrategias que también maximizarían sus beneficios podría dar una opción: el experimento es replicable tantas veces como sea necesario y cada eslabón puede conocer una estrategia óptima para una gran cantidad de demandas de consumidor posibles.

2.2. El Juego de la Distribución de Cerveza

El Juego de la Distribución de Cerveza (en inglés *The Beer Distribution Game*) [14] fue planteado por primera vez en la Escuela de Administración y Dirección de Empresas Sloan del MIT para ejemplificar el *efecto látigo*, llamado así por la similaridad que tiene el comportamiento de la información en cada nivel de la cadena con el patrón ondulado que toma un látigo, en el cual existe una mayor amplitud de onda (comparable con el ruido o la varianza en la información) al alejarse del punto de origen (comparable al consumidor).

En su forma de juego de mesa, se presenta comúnmente a alumnos recién ingresados a distintas escuelas de negocios. El MIT (Dizikes [5]) y varios autores como Sterman [15] han publicado que, independientemente del rol que jueguen y de cuánta experiencia y preparación en negocios tengan, los humanos consistentemente fallan en encontrar la estrategia para maximizar la utilidad. Para nosotros los humanos, es sumamente difícil pensar de manera no lineal y tomar en cuenta efectos como retrasos (tanto en entrega de los pedidos como en la información) o la retroalimentación en los ciclos (cuando hay inventario sobrante, puede ser llevado al siguiente periodo). Estos conceptos son manejados correctamente cuando se modelan como efectos de sistemas dinámicos, tarea considerablemente más fácil para una computadora que para una persona. Asimismo, nota que es inevitable la frustración, y que incluso los equipos que obtienen los mejores resultados del juego terminan lejos del óptimo teórico.

2.3. Principales características

El juego consiste, a grandes rasgos, en asignar a cada agente un rol en una cadena de suministro de cerveza, buscando maximizar las ganancias individuales al final del juego.

Existen cuatro jugadores: minorista (*retail*), mayorista (*wholesale*), distribuidor (*regional warehouse*) y fábrica (*factory*). Dentro del juego transcurren 52 semanas (un año), durante las cuales existe una demanda del consumidor, que se revela al inicio de la semana. De esta manera, el minorista debe cubrir (restringido a su inventario) la demanda del consumidor, y decidir la orden que pedirá al mayorista para recibir la siguiente semana. Cada jugador sigue instrucciones similares, con el objetivo de maximizar sus ganancias al final del juego.

Durante este trabajo, se referirá a las relaciones entre los jugadores de la cadena de la siguiente manera: respecto a cada uno, aquel que le compra es el "inferior" y aquel que le vende es el "superior". Así, para el mayorista, el jugador inferior es el menudeo y el superior es el almacén regional. Para definir el sistema por completo, se asignará al consumidor como jugador inferior para el minorista, y a los campos como superior para la fábrica.

Todos los agentes cuentan con un inventario inicial de cajas de cerveza, y deben manejar correctamente su inventario para poder cumplir con la demanda del agente inferior, al tiempo de minimizar los costos de almacenamiento por cada caja. Todos reciben ingresos por vender cajas de cerveza, e incurren en costos por comprar inventario, almacenar inventario, y por último, una penalización por no cubrir las órdenes. La estructura del juego se puede observar en la figura 2.2.

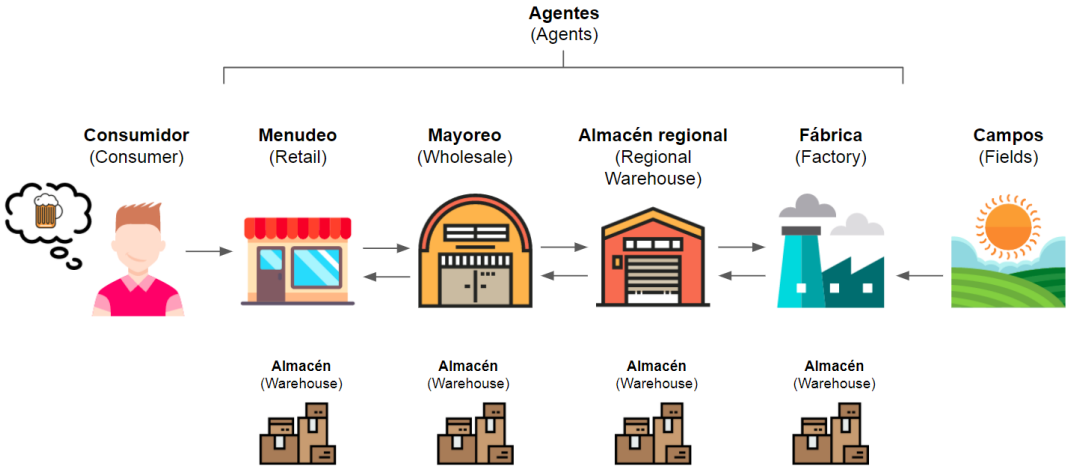


Figura 2.2: Estructura del Juego de la Distribución de Cerveza[2]

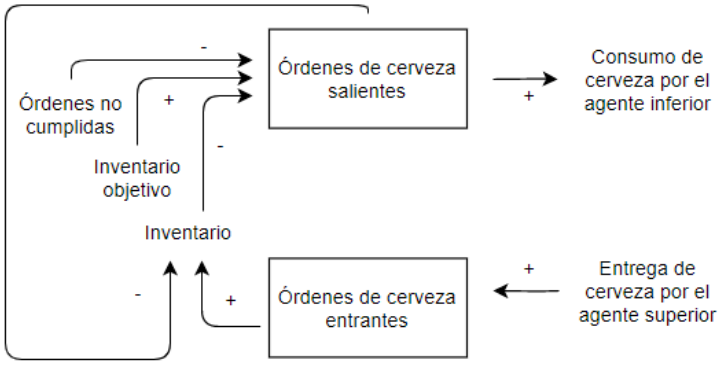
De manera general, las variables que tienen efecto en la solución de este problema son:

- Demanda del consumidor durante el año
- Producción (oferta) de los campos durante el año
- Precio de la cerveza (se supone el mismo margen nominal para cada agente)
- Costo de almacenamiento (se supone igual para todos los agentes)
- Costo de oportunidad por órdenes no cumplidas (se supone igual para todos los agentes)
- Inventario inicial (puede variar por agente)

[2]Iconos creados por Freepik, Iconnice, Roundicons en *www.flaticon.com*

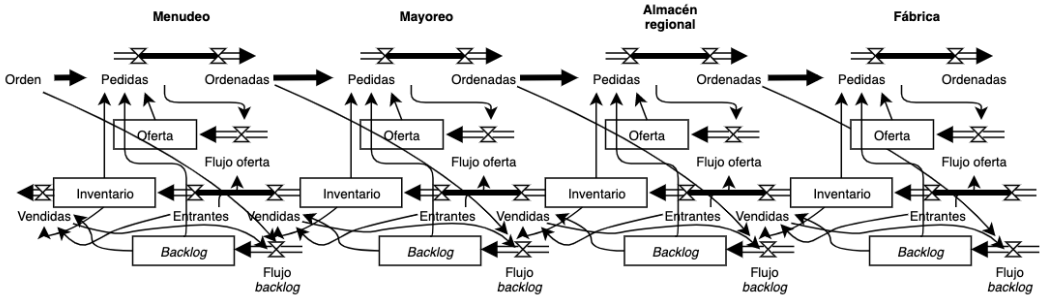
Sin embargo, en cada tiempo t, cada agente se enfrenta a factores específicos que afectan su decisión de compra para el siguiente día, cuyo diagrama se puede consultar en la figura 2.3. Este conjunto de factores ha sido mapeado anteriormente por Conneely et al. [4] y Grasl [7], desde el punto de vista de diagramas causales. Una gran parte de la complejidad de este problema se puede identificar aquí: ningún agente tienen visibilidad del proceso análogo para su agente superior, por lo tanto debe actuar bajo información incompleta y con indicaciones de comportamiento, tales como evitar el costo por órdenes no cumplidas y reaccionar a los cambios en la demanda del agente inferior.

Figura 2.3: Diagrama Causal para un agente del Juego



Este diagrama causal describe un diagrama de existencias y flujos (*stock and flow diagram*), el cual describe las ecuaciones del sistema que serán presentadas en las secciones siguientes. Se puede consultar en la figura 2.4

Figura 2.4: Diagrama de Existencias y Flujos para el Juego



Todas las variables, menos la demanda del consumidor y la producción en los campos, pueden

ser declaradas para el sistema (igual para todos los agentes) o individualmente (diferente para cada agente). Como se verá más adelante, el sistema es sumamente sensible a pequeños cambios en cualquiera de las variables; por ejemplo, si un agente comienza el juego con suficiente inventario para todo el año, el agente superior nunca podrá venderle cerveza e incurrirá solamente en costos por almacenamiento.

2.3.1. El Efecto Látigo

El *Efecto Látigo* es un fenómeno que se produce en cadenas de suministro. Se llama de esa manera porque, mientras más "arriba" en la cadena de suministro se encuentre un agente (es decir, más lejos del contacto directo con el comprador), más distorsionada es la información que tiene acerca de la verdadera demanda del comprador; tal varianza se puede visualizar como una curva que asemeja un látigo. Chaharsooghi et al. [3] nota que la distorsión se debe mayormente a tres factores: prejuicios en la información de la demanda por parte de los miembros cercanos al consumidor, retraso en el intercambio de información entre los miembros de la cadena y soporte logístico inapropiado a través de la cadena.

El caso más común de efecto látigo surge por la necesidad de un inventario de seguridad, para asegurar que un agente no incurrirá en costos por órdenes no cumplidas. Cada agente añadiría cantidad a su orden, causando una orden artificialmente inflada para el agente más lejano al consumidor. Si el comportamiento del consumidor no cambia, las órdenes se estabilizarían en un nivel inferior después de un tiempo, pero los agentes superiores podrían estar incurriendo en un mayor costo de almacenamiento en el mismo tiempo. Además, tener menos claridad acerca de la demanda real del consumidor

El efecto puede ejemplificar con el siguiente escenario:

1. El consumidor, que generalmente compra 6 cervezas, ahora quiere 10, pero la tienda minorista solamente cuenta con 7. El minorista le venderá todo su inventario, pues es la acción que maximiza su ganancia. Después, debe decidir si volverá a tener un inventario de 6 o si debe pedir un número mayor de cervezas, atendiendo la posible creciente demanda. Decide pedir

9 cervezas al siguiente nivel, la tienda de mayoreo.

2. El mayorista cuenta con 17 cervezas. Llena el pedido del minorista, pero decide que tenía guardado demasiado inventario, así que se queda con 8 cervezas en su almacén, sin hacer una orden al siguiente nivel, la tienda de distribución.

3. La tienda de distribución decide no comprar unidades a la fábrica, dado que no disminuyó su inventario.

4. La fábrica conoce la restricción de estacionalidad de la cebada, así que compra una mínima cantidad a los campos.

En este escenario, que se puede visualizar en la figura 2.5, el mayorista obtuvo información distorsionada acerca del repentino crecimiento en la demanda del comprador, mientras que la tienda de distribución podría incluso interpretar que el comprador disminuyó su consumo. Si este comportamiento se mantiene durante algunos periodos más, recibiría la noticia (por medio de un incremento en las órdenes regulares) con un retraso considerable, lo cual inevitablemente causará perturbaciones en sentido contrario cuando incrementen sus órdenes recibidas.

Figura 2.5: Ejemplo de efecto látigo

En el caso de que la demanda sea determinista (con un castigo por órdenes no cumplidas), la orden óptima en cada tiempo para cada miembro de la cadena es “una por una”, es decir, demandar al agente superior exactamente lo mismo que fue demandado por el agente inferior.

2.3.2. La dinámica de las existencias y flujos

Las existencias y flujos (*stocks and flows*) son componentes básicos en un sistema dinámico [3]. Las existencias son acumulaciones basadas en la diferencia entre los flujos entrantes y los salientes. El flujo neto a una existencia en un tiempo determinado t es la tasa de cambio de la existencia en el tiempo t:

$$Existencia(t) = \int_{t_0}^{t} [Entrada(s) - Salida(s)]ds + Existencia(t_0)$$

Es común que en un sistema con varias existencias con sus respectivos flujos ocurran retrasos. Esto sucede porque, en general, los flujos de entrada y salida son gobernados por decisiones diferentes. En el Juego de la Distribución de Cerveza, para cada agente, su flujo de salida depende de la demanda del agente inmediatamente inferior limitada por su propia existencia, y su flujo de entrada depende de su propia demanda limitada por la existencia del agente inmediatamente superior. Esto, aunado a un comportamiento ligeramente aleatorio de la demanda del consumidor, puede desembocar en insuficiencia de existencias para cualquier agente en cualquier tiempo t.

Un sistema está en equilibrio cuando ninguna de sus existencias cambia. Esta condición se obtiene cuando el cambio neto de sus flujos es cero, es decir, cuando el flujo entrante es igual al flujo saliente.

La solución trivial para este equilibrio es que todos los flujos sean nulos (*equilibrio estático*), sin embargo, en el caso de la cadena de suministro, este escenario llevaría a que ningún agente tuviera ventas, y por consiguiente, tampoco tuviera ganancias. Una situación más adecuada sería un *equilibrio dinámico*, en el cual las existencias totales se mantienen constantes, pero los flujos son diferences de cero. Para la cerveza, esto significaría que la cantidad de cerveza en los almacenes se mantiene constante, pero cada caja de cerveza se mueve de un agente a otro en algún punto en el tiempo. Sin restricciones de oferta, la situación ideal tendería a no guardar cerveza en los almacenes. Sin embargo, el efecto látigo, causado por cambios en la demanda del consumidor, com-

[3] Para mayor profundidad acerca de sistemas dinámicos y el comportamiento de las existencias y flujos, Sterman [15] profundiza y provee de las bases adecuadas para sistemas más complejos.

plica la estimación de los flujos entrantes y salientes en cada tiempo, pues para cada agente son desconocidas tanto la demanda del agente inferior como la capacidad del agente superior de cubrir la demanda propia. Existe una complicación extra al añadir estacionalidad en la producción de los campos de cebada.

2.4. Acercamiento en el Presente Trabajo

En este trabajo se modelará el Problema de Distribución de Cerveza, *The Beer Distribution Game*, ampliamente utilizado y explicado por el Profesor Sterman [15] en la Escuela de Administración y Dirección de Empresas Sloan del MIT, como una cadena de suministro, para ilustrar el concepto de *efecto látigo* (en inglés, *bullwhip effect*). Este efecto recibió tal nombre debido a que la varianza en la información acerca de la demanda real tiene el mismo comportamiento que un látigo: mientras más lejano se encuentra del origen (consumidor), mayor amplitud tiene la onda (varianza).

Existen acercamientos anteriores a este problema, en específico Chaharsooghi et al. [3] propone ya un acercamiento con *Q-learning* [4] pero sin restricción de estacionalidad en la materia prima, Strozzi et al. [16], por medio de Algoritmos Genéticos, Kimbrough et al. [10] y Zarandi et al. [18] proponen híbridos de un algoritmo genético y aprendizaje reforzado.

Por otro lado, algunas líneas de investigación como Busoniu et al. [2] se han concentrado en el aspecto de teoría de juegos que compete a este problema: si un vendedor de menudeo no tiene existencias, los consumidores podrían elegir otro. En este caso, los agentes serían construidos como adversarios. Dado que en el modelo que se resuelve en el presente trabajo, el sistema se compone de agentes representativos (i.e. se resuelve la ecuación para el agente prototipo llamado "tienda de menudeo", pero tal solución óptima podría ser adoptada por cualquier tienda de menudeo específica), es equivalente a tener un solo agente, que por sí mismo no puede ser ni adversario ni cooperativo, sino solamente maximiza su propia utilidad. Es por esto que no se implementará la metodología de teoría de juegos.

[4] Los nombres de los algoritmos serán representados con itálicas en este trabajo.

Sin embargo, todos los acercamientos anteriores suponen que los campos pueden adaptarse de inmediato a la demanda de la fábrica, de cierta manera tienen "producción infinita". Los aportes de este trabajo son:

1. Tendencia de demanda del consumidor de cerveza más flexible y cercana a condiciones reales, basada en tendencias catalogadas por una agencia de investigación de los EUA
2. Restricción estacional en la producción de cebada, basadas en tendencias catalogadas por el departamento de agricultura de los EUA

Tales supuestos deberían alterar el comportamiento de los agentes tal que respondan a la tendencia de demanda del consumidor, y mantengan una existencia positiva en sus almacenes para poder enfrentar tal demanda en periodos de baja producción. Esto debería verse reflejado en estrategias que sigan la curva de producción cuando existe una restricción en los campos, o que se asemejen más a la curva de demanda del consumidor si tal restricción no existe.

En el mundo real, los agentes tienen acceso a esta información, a sus propios datos internos de ventas de años pasados, a predicciones del año en curso, etc. El modelo que se construirá en este trabajo solamente considera los datos públicos, lo que sería el caso para un agente completamente nuevo en el juego, pero es una oportunidad de desarrollo futuro.

En este trabajo se presentarán diversas simulaciones, que reflejan distintos escenarios tanto de condiciones iniciales, como de comportamiento. Se analizará, por ejemplo, si las estrategias encontradas por el algoritmo son preferibles a otras estrategias de sentido común, qué pasa si los agentes no tienen posibilidad de almacenar, si solamente un agente sigue una estrategia óptima, entre otros. Es importante notar que, en todos los escenarios, se asegurará que los supuestos en cada escenario sean realistas, por ejemplo, el costo de almacenamiento deberá ser menor al margen por ventas, etc.

Capítulo 3

Aprendizaje Reforzado

3.1. Orígenes y Descripción

El Aprendizaje Reforzado se basa en la psicología conductista: un *agente* busca ser recompensado con un premio, el cual obtiene cuando realiza una secuencia de *acciones* que lo llevan a concluir una tarea exitosamente. Además, para maximizar la cantidad de *recompensa* que recibe - o, alternativamente, minimizar el tiempo que espera entre un premio y el siguiente - comienza a optimizar su política (π) para llegar a la meta satisfactoriamente. Así, tal agente *aprende* estrategias y va *reforzando* el aprendizaje de tales estrategias que maximicen sus recompensas. Una explicación mucho más profunda de las bases e historia del aprendizaje reforzado, así como la mayor parte de las definiciones mencionadas en este capítulo, pueden ser encontradas en Sutton y Barto [17].

3.1.1. Terminología común

Existen conceptos utilizados en prácticamente la totalidad de los algoritmos de aprendizaje reforzado, independientemente de las particularidades que presenten. A continuación se presenta una lista, basada en la recopilación no exhaustiva de ellos de Hu [8], pero que cubre los términos utilizados en este trabajo.

1. Agente: es la entidad que aprenderá durante el proceso. Por ejemplo, puede ser un ratón

robótico aprendiendo a llegar al queso en el laberinto; o un jugador virtual aprendiendo a superar todos los niveles de un videojuego.

2. Estado: describe la situación en la cual se encuentra el sistema. Por ejemplo, para el ratón robótico, el estado es la casilla en la que se encuentra. El estado puede tener tantas dimensiones como sea necesario: por ejemplo, el estado de un coche autodirigido sería descrito por su posición, velocidad, objetos en su radar e incluso cantidad de gasolina en el tanque.

3. Mundo: se refiere la totalidad de los posibles estados.

4. Acción: es lo que un agente puede hacer en cada estado. Generalmente, el conjunto de acciones que puede tomar un agente en un estado determinado es finito, aunque el conjunto de acciones que tome durante todo el tiempo pueda tener combinaciones que se aproximen al infinito. Por ejemplo, para el ratón robótico, sus acciones son realizar un movimiento hacia adelante, atrás, derecha o izquierda.

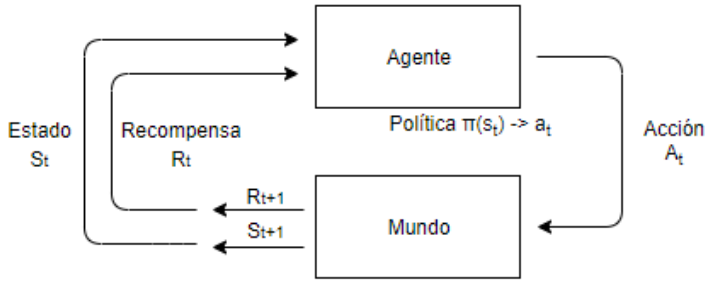
5. Recompensa: cuando un agente toma una acción en un estado, recibe una recompensa. Es muy importante notar que la recompensa puede tomar muchas formas: es posible que sea solamente la meta última, en vez de existir después de cada acción (como el queso para el ratón). La recompensa también puede ser negativa, equivalente a un castigo, lo cual causa que el agente aprenda a evitar las acciones que lo llevan a ella.

6. Política (*policy*) es una función que toma como argumento un estado s y devuelve una acción a; equivalente a la estrategia del agente. La meta del aprendizaje reforzado es encontrar la política óptima. Generalmente se le representa como $\pi(s) -> a$.

La relación entre estos conceptos se puede consultar en la figura 3.1, popularizada por Sutton y Barto [17].

Una vez que tenemos asignados estos elementos, resulta relativamente simple explicar el problema, pero no así la implementación. Por ejemplo, ratón solamente recibirá la recompensa después de un movimiento específico que lo lleve a la casilla en la que se encuentra el queso; todos los demás movimientos son solamente un medio para acercarse. El aprendizaje reforzado resuelve esta

Figura 3.1: Interacción de los conceptos en Aprendizaje Reforzado



situación asignándole valor a las recompensas a largo plazo en vez de solamente a las inmediatas. Esto quiere decir que si planteamos el objetivo del aprendizaje como llegar al queso, el ratón no podrá decidir si es mejor un camino más rápido o uno en el que dio vueltas sin sentido en un mismo lugar, siempre que en un tiempo finito haya llegado al final. Sin embargo, si descontamos el valor del queso por cada casilla extra que le toma llegar a él, entonces buscará el camino más corto, optimizando su secuencia de acciones interactuando con el mundo que lo rodea para descubrir relaciones acción-estado. La representación matemática de este problema se condensa en la *Ecuación de Bellman*, la cual será desarrollada en la siguiente sección, pero de una manera muy simple podría ser representada como:

$$\textit{Recompensa} = \textit{Valor calórico del queso} - \textit{Calorías necesarias para moverse una casilla} * \textit{Total de casillas visitadas}$$

3.1.2. La Ecuación de Bellman

La principal característica del agente es que tiene la capacidad de tomar decisiones sobre sus acciones, las cuales son su forma de interactuar con el mundo, llevándolo de un estado a otro. El agente no tiene acceso a todas las consecuencias de sus acciones; de hecho, ni siquiera conoce todo el mundo. Además, como se verá, su percepción es que no existe un retraso en la recompensa.

El agente toma una acción en el tiempo t, la cual depende del estado s_t del mundo. En $t+1$, el mundo reaccionó ya a la interacción del agente con él, así que el agente recibe una recompensa

r_{t+1} y toma una nueva acción dependiendo del estado s_{t+1} del mundo. Sin embargo, no es óptimo seleccionar acciones solamente con base en la recompensa r_{t+1}, pues la naturaleza temporal del problema lo convierte en un problema a largo plazo, y el agente estaría considerando solamente consecuencias en el corto plazo.

Así, el agente debe aprender que existe un *retraso* entre cada acción que toma y el premio. Supongamos que, en una cuadrícula, el premio se encuentra en la casilla (x,y). El agente solamente puede llegar a esa casilla meta desde las adyacentes, pero si no se encuentra en una de estas, primero debe acercarse a alguna de las casillas adyacentes, por ejemplo (x-1,y). Recursivamente, si no se encuentra en ninguna de estas casillas, debe encontrar un camino que lo lleve ahí. Este proceso se conoce como *exploración*, pues es posible que el agente no reciba ninguna recompensa por sus acciones hasta que encuentre por primera vez la casilla (x,y) en el tiempo t y asigne un valor de recompensa positivo (aunque descontado) a la casilla en la que se encontraba al tiempo *t-1*. Así, cuando el agente comienza su exploración, irá aprendiendo que, lejos de que la recompensa sea inmediata, debe tomar una secuencia de acciones para llegar a ella. Una vez que el agente tiene mejor conocimiento acerca de cuáles casillas lo acercan al premio, pasa a una fase llamada *explotación*, en la cual busca la recompensa más grande entre todas las que conoce. Un agente debe comenzar su aprendizaje puramente en modo exploratorio, y transicionar al modo de explotación conforme conoce mejor su ambiente.

Podemos entonces definir la *función de valor* asociada a la política como el valor esperado de la recompensa al tiempo t dado que el agente se encuentra en el estado s. Además, cada periodo de tiempo se le aplica un factor de descuento γ para obtener el valor presente de tal recompensa. De esta definición deriva uno de los conceptos más importantes en el aprendizaje reforzado, la *Ecuación de Bellman*. A continuación se explica el desarrollo de esta, obtenido de Sutton y Barto [17].

La idea básica es que la recompensa total en el tiempo t es que se compone de la recompensa parcial en el tiempo $t+1$ después de tomar alguna acción; más la recompensa parcial en el tiempo $t+2$, descontada porque existe un retraso y tomando en cuenta que se proviene de un estado

s posible si el agente tomó la acción que tomó en el tiempo $t+1$, y así sucesivamente. Esto se representa con la siguiente ecuación recurrente:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + ...$$
$$R_t = r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + ... \right)$$
$$R_t = r_{t+1} + \gamma R_{t+1}$$

El valor de un estado s se puede definir como la recompensa esperada dado que el agente empieza en ese estado, y depende de su política:

$$V^\pi(s) = E_\pi \left\{ R_t | s_t = s \right\} = E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right\}$$

3.1.3. El método ϵ-*greedy*

Cuando una persona se muda a una nueva ciudad, es posible que quiera encontrar su nuevo restaurante favorito. Al principio experimentará en muchos restaurantes, probará comida nueva, e irá descubriendo el mundo y acumulando aprendizaje. Después de algún tiempo, esta persona ya tendrá una idea bastante robusta de cuál(es) restaurante(s) le gusta(n), así que más bien explotará este conocimiento e irá ahí más seguido. Sin embargo, si mantiene sus opciones abiertas y de vez en cuando visita un restaurante nuevo, podría encontrar su nuevo favorito.

El método ϵ-*greedy* es una manera de que el agente ajuste su comportamiento mientras transcurre el tiempo: al principio debe *explorar* para conocer la mayor cantidad de consecuencias a sus acciones posibles, pero más adelante debe *explotar* el conocimiento de cuáles acciones le han reportado buenas recompensas y tomar esas decisiones más seguido.

Definamos $p_r t$ y $p_t t$ como las probabilidades al tiempo t de exploración y explotación, respectivamente. Entonces:

$$p_r t = 1 - \epsilon(t)$$
$$p_t t = 1 - p_r t \qquad \forall t$$

A esta estrategia de exploración se le llama $\epsilon - greedy$ porque el agente es codicioso (en inglés, *greedy*) al querer siempre buscar la máxima recompensa posible, pero solamente hasta un ϵ. Tal función ϵ suele ser implementada como decreciente de forma lineal para el aprendizaje, de tal forma que mientras pasa el tiempo, el agente escoge las acciones conocidas que le reportan mayor utilidad más seguido. De esta manera se asegura que siempre existe una probabilidad positiva de explorar.

3.1.4. Los MDP (Procesos de Decisión de Markov)

Un algoritmo de aprendizaje reforzado que cumple con la propiedad de Markov se llama Proceso de Decisión de Markov (MDP, por sus siglas en inglés). La propiedad de Markov a grandes rasgos, dice que el futuro solamente depende del estado obtenido con las acciones inmediatamente anteriores, no del pasado que haya transcurrido previamente. En términos formales ([17]), si se supone que existen un número finito de estados y recompensas [1], entonces se puede considerar cómo un mundo respondería en el tiempo $t+1$ a la acción tomada en el tiempo t. Esto se describe con la distribución conjunta de probabilidad:

$$P\{S_{t+1} = s', R_{t+1} = r \quad | \quad S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t\},$$

para todos las recompensas r, estados s' y todos los valores posibles de los eventos pasados: $_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t$. La propiedad de Markov indica que la respuesta del mundo en el tiempo $t+1$ depende solamente del estado y acciones en el tiempo t, es decir:

$$p\,(s', r \quad | \quad s, a) = P\{S_{t+1} = s', R_{t+1} = r \quad | \quad S_t = s, A_t = a\},$$

para todos las recompensas r, estados s' y acciones a. Si el mundo tiene la propiedad de Markov, entonces es posible calcular la recompensa esperada dado un estado actual y una acción.

Formalmente, un Proceso de Decisión de Markov es una tupla (X, U, f, ρ) en donde X es el conjunto finito de estados, U es el conjunto finito de acciones de un agente en un estado,

[1] Si el supuesto de finitez para los estados de recompensas es removido, el argumento debe ser extendido a integrales y densidades de probabilidades en lugar de sumas y probabilidades.

$f : X * U * X \to [0, 1]$ es la función de probabilidad de transición de estado, y $\rho : X * U * X \to \mathbb{R}$ es la función de recompensa. Es evidente que las políticas anteriores no juegan un rol, solamente el estado en el que se encuentra el agente.

Cuando el conjunto de acciones (o la política) a tomar es difícil de aprender porque no existen ejemplos, o el mundo/conjunto de acciones/conjunto de consecuencias es demasiado grande (e.g. infinito), es apropiado utilizar Aprendizaje de Máquina Reforzado en lugar de Aprendizaje de Máquina regular (i.e. modelos supervisados o no supervisados). Gracias a que el agente va descubriendo y acercándose a una política óptima, no es estrictamente necesario que explore todas las acciones posibles.

En este trabajo se aplicarán dos algoritmos sumamente conocidos y aplicados en un sinfín de ámbitos: *Policy Iteration* y *Q-learning*, y compararemos su rendimiento al aplicarlos al problema de la distribución de cerveza.

3.2. Policy Iteration

En este algoritmo, se empieza con una política aleatoria, se encuentra su valor y se realiza un paso que encuentra una política nueva (mejor) basado en la anterior. Bajo el supuesto de que los agentes tienen una cantidad finita de acciones en cada estado, existen una cantidad finita de posibles políticas y es razonable esperar convergencia en un algoritmo de búsqueda. De cierta manera, dado un tiempo suficientemente grande y cardinalidad finita de las acciones a tomar en cada estado por cada agente, *policy iteration* asegura convergencia a la política óptima. En general, el algoritmo parece terminar en tiempo exponencial o pseudopolinomial, pero el máximo tiempo posible es todavía un tema abierto de investigación.

Recordemos que en cada tiempo t, cada agente puede tomar una acción a y recibe una recompensa r. De manera general se puede suponer que debido a esto, el proceso de aprendizaje se comporta como un proceso de Markov (MDP). Así, de manera iterativa se puede asignar una política π a cada agente (es importante recordar que una política es el conjunto de acciones a_t a tomar

en cada estado e_t), evaluar su desempeño por medio de la recompensa final que representa para el agente, y avanzar a la siguiente iteración en la cual se intentará mejorar la mejor política histórica.

Esto quiere decir que, tomando una política base π_0, se puede encontrar una política π_1 que otorga una recompensa igual o mayor (i.e. $V^{\pi_0}(s) < V^{\pi_1}(s) \forall s \in S$). En algún momento, el algoritmo llega a una política π_* que no puede ser mejorada, la cual es declarada la política óptima.

Se puede representar este proceso como:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} ... \xrightarrow{I} \pi_* \xrightarrow{E} V^{\pi_*}$$

En donde $\xrightarrow{E}$ significa que una política se *evalúa*, y $\xrightarrow{I}$ significa que se mejora (por *improvement*, en inglés).

Las ecuaciones que representan el algoritmo se presentan a continuación como son descritas por Frazzoli [6]. Para encontrar el valor de cada estado bajo la política π:

$$V(s) = \sum_{s' \in S} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V(s')] \quad \forall s \in S$$

Para mejorar la política, se aplica para cada $s \in S$:

$$\pi(s) \leftarrow \max_a \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V(s')]$$

iterando ambos pasos hasta que π_* converja.

El modelo desarrollado en este trabajo será implementado como un algoritmo de *policy iteration*. Algunas partes relevantes del código se pueden consultar en la sección de Resultados.

Existe un algoritmo similar llamado *value iteration*, el cual itera sobre la función de valor e infiere la política relacionada al valor óptimo obtenido al final, en lugar de iterar sobre la política y evaluar el valor. Se suele preferir el algoritmo de *policy iteration* sobre *value iteration*, pues converge más

rápidamente (en términos de iteraciones necesarias), presumiblemente porque la función de valor cambia muy poco de una política a otra. Además, dado que se itera sobre la política anterior y se busca una mejor, el algoritmo converge rápidamente y cada paso asegura una mejora. Se puede consultar más acerca del algoritmo *value iteration* y de la programación y convergencia de ambos en Sutton y Barto [17].

3.3. Q-Learning

Uno de los algoritmos más populares de aprendizaje es *Q-learning*. Obtiene este nombre por la *función Q*, que indica el valor descontado de la recompensa asociado a una acción en un estado determinado. La principal ventaja de este tipo de aprendizaje es que no está basado en un modelo, es decir, que se conoce una cantidad mínima de detalles acerca del mundo y su funcionamiento, y el agente aprende con base en su experiencia al interactuar. Esta libertad permite, por ejemplo, relajar el supuesto del proceso comportándose como un proceso de Markov.

3.3.1. Conceptos

Recordemos la ecuación de valor de un estado, basada en la ecuación de Bellman, que fue desarrollada en la sección introductoria:

$$V^{\pi}(s) = E_{\pi}\left\{R_t | s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\}$$

De aquí sigue que el valor de tomar una acción específica a en el estado s usando la política π es:

$$Q_{\pi}(s,a) = E_{\pi}\left\{R_t | s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\}$$

$$V(s) = \max_{a} Q(s,a)$$

$$Q(s,a) = R(s,a) + \gamma * \max_{a} Q(s^{'}, a^{*})$$

Donde s' es el siguiente estado, y a^* representa todas las acciones posibles. Al estimar la función

Q para cada par de estado con acción, es posible encontrar la mejor acción para cada estado y, así, obtener una política óptima.

Al iniciar el periodo de aprendizaje del algoritmo, la función Q se establece como $Q(s, a) = 0 \; \forall \; s, a$, y en cada paso (e iteración) se actualiza su valor.

3.3.2. Algoritmo

1. Asignar $Q(s, a) = 0$ para todos los estados y acciones.
2. Posicionarse en un estado s
3. Seleccionar acción a^* y ejecutar
4. Recibir recompensa r
5. Observar estado nuevo $s^{'}$
6. Actualizar $\hat{Q}(s, a) = r(s, a) + \lambda \, \text{máx}_{a'} \, \hat{Q}(s^{'}, a^{'})$
7. Asignar nuevo estado $s \leftarrow s^{'}$
8. Volver al punto 2 hasta convergencia

La principal diferencia entre *Policy iteration* y *Q-learning* es la forma del resultado para cada agente. Mientras que el primer método proporciona una sola política óptima, que indica la acción a tomar por el agente en cada tiempo t, el segundo consta de una serie de valores de la función Q correspondientes a diferentes acciones en cada estado s, incluso si tales estados solamente son alcanzables tomando acciones previas no correspondientes a las óptimas. En términos prácticos, el resultado de *Q-learning* permitiría corregir errores y volver a acercarse al óptimo, pero con un costo de oportunidad relacionado al tiempo computacional requerido para entrenarlo.

Hasta este momento, se han establecido las bases para dos técnicas de aprendizaje reforzado para un agente. Sin embargo, el Problema de la Distribución de Cerveza es multiagente por naturaleza: cada eslabón de la cadena de suministro juega un papel diferente o puede tener márgenes y costos

diferentes. Esta generalización se conoce como Aprendizaje Reforzado Multi-Agente, el cual presenta retos nuevos, al tiempo de abrir las puertas a sistemas dinámicos más complejos.

3.4. Aprendizaje Reforzado Multi-Agente (ARM)

Existen muchos problemas que inherentemente parecen tener soluciones sencillas, pero es necesario que más de un agente aprenda de manera simultánea, especialmente si la recompensa al final es compartida entre todos ellos. Por ejemplo, imaginemos que tenemos dos robots, ladrillos y un cuadro dibujado en el piso. Cada uno de los robots está cerca de esquinas opuestas y tiene acceso a la mitad de los ladrillos. La recompensa final se les otorga cuando forman un cuadrado de ladrillos, y es mayor mientras menor sea el tiempo que les tome terminar. En este caso, la estrategia óptima es que cada uno de los robots coloque, en cada tiempo, el ladrillo que más cerca le quede. El probema se vuelve sumamente complejo, pues cambios en la estrategia de un agente puedem afectar la estrategia de otros agentes. Por ejemplo, si uno de los robots tiene un comportamiento errático porque recibe un castigo si le sobra energía al terminar la tarea, el otro deberá adaptarse a esta nueva forma de actuar.

A grandes rasgos el aprendizaje reforzado multi-agente (ARM), es una generalización de los conceptos estudiados anteriormente, al permitir más de un agente aprender y tomar sus propias decisiones, en un mundo en el cual puede interactuar con los demás. Muchas de las definiciones específicas del ARM, la explicación a fondo de cómo comportamientos erráticos en un agente pueden afectar las estrategias de los demás, así como de su intersección con teoría de juegos, pueden encontrarse en Bloembergen y Hennes [1].

Existen pros y contras relacionados al ARM, como los referidos por Busoniu et al. [2]. Algunos beneficios son que es posible que algunos agentes asuman tareas ajenas si esto ayuda a la optimización del premio final, es inherentemente rubusto, y generalmente están diseñados de tal manera que añadir nuevos agentes es fácil. Por otro lado, algunos retos son la escalabilidad debido al crecimiento exponencial de la dimensión del espacio estado-acción, o a la complejidad que se genera al añadir cada nuevo agente, si este tiene una definición, estrategias o caracterśticas diferentes.

En este trabajo, se considerará a cada eslabón de la cadena de suministro de cerveza como un agente. Cada agente solamente puede comunicarse con los niveles inmediatamente vecinos; es decir, las únicas interacciones que puede tener con el mundo son el número de órdenes que recibe del nivel inferior y el inventario que pide al nivel superior. De esta manera, nuestros agentes no son adversarios, cooperativos ni independientes; esto es un *input* de la dirección en la cual fluye la cadena de suministro. El objetivo de cada agente es maximizar sus recompensas individuales.

Hemos definido una penalización por mantener cerveza en el inventario (el costo del almacén), así que la decisión concerniente a la petición del nivel inferior queda determinada: venderá todo lo que pueda, pues cada venta le reporta una ganancia, y no llenar la orden completa cuando tiene suficiente inventario lo haría incurrir en un costo innecesario.

Para cada agente, el conjunto de **acciones** que puede tomar es solamente el número de cervezas que pedirá al nivel inmediatamente superior en cada tiempo t. Lo que tendrá guardado en la bodega en el tiempo t estará constituido por el número de cervezas que tenía en el tiempo anterior $t-1$, menos el número de cervezas vendidas al agente inferior, más el número de cervezas que recibe del nivel inmediatamente superior por el pedido de reaprovisionamiento, restringido a que cada agente solamente cubrirá la orden del nivel inferior si tiene suficiente inventario para hacerlo.

El objetivo de cada agente es maximizar su recompensa. Sin embargo, este es un problema ligeramente diferente a los comunes de *Q-learning*, en los cuales el valor de la recompensa es conocido y, una vez encontrado, se buscan las acciones óptimas "de atrás hacia adelante" (como el ejemplo antes mencionado de un ratón buscando un queso en un laberinto).

Su política está definida con base en la función Q, una vez que el proceso de aprendizaje fue finalizado, de esta manera, puede realizar una búsqueda sobre todas las posibles acciones en los estados y sencillamente escoger la mejor, lo cual converge a la política (cuasi)óptima.

Es importante destacar que este sistema toma solamente una de las ramas que existen en la industria de cualquier producto (existe más de un minorista, etc.), e incluso, toma solamente un producto. Es entonces una generalización en la que se propone un "agente representativo" en cada nivel de la cadena. Aún así, es un sistema complejo bastante robusto y sensible a cambios pequeños como un aumento en el costo de almacenamiento o un cambio agresivo en el margen de un eslabón.

Capítulo 4

Acercamientos Previos a la Solución

4.1. Estática: máxima ganancia posible

El sistema tiene una solución analítica si no existe una restricción en la oferta (es decir, el campo siempre puede cubrir la demanda de la fábrica) y cada agente tiene en el día n información acerca de la demanda del día $n+1$ de su agente inmediatamente inferior. En este caso, los agentes ni siquiera necesitan usar sus almacenes: sencillamente compran cada día lo necesario para el día siguiente. Podemos observar el comportamiento de esta solución en la figura 4.1.

Sin embargo, incluso cuando todos los agentes tienen información perfecta, podemos observar una versión muy ligera del efecto látigo. Si el minorista tiene suficiente cerveza para cubrir algunos días de demanda del consumidor, entonces no tiene incentivo para comprarle al mayorista hasta que su inventario se agote. Esta situación puede replicarse en los dos niveles superiores, de tal manera que la fábrica no recibe nada de información acerca de la demanda del consumidor durante una cantidad considerable de tiempo. Este efecto se puede observar en la figura 4.2.

Si la fábrica quisiera estimar su demanda para el siguiente año solamente con esta información, tendría que crear un modelo (aunque fuera muy sencillo) para interpolar aquellos días en los cuales no tuvo información. En el caso optimista, usará el mismo patrón observado retroactivamente, y

Figura 4.1: Estrategias aprendidas con información perfecta

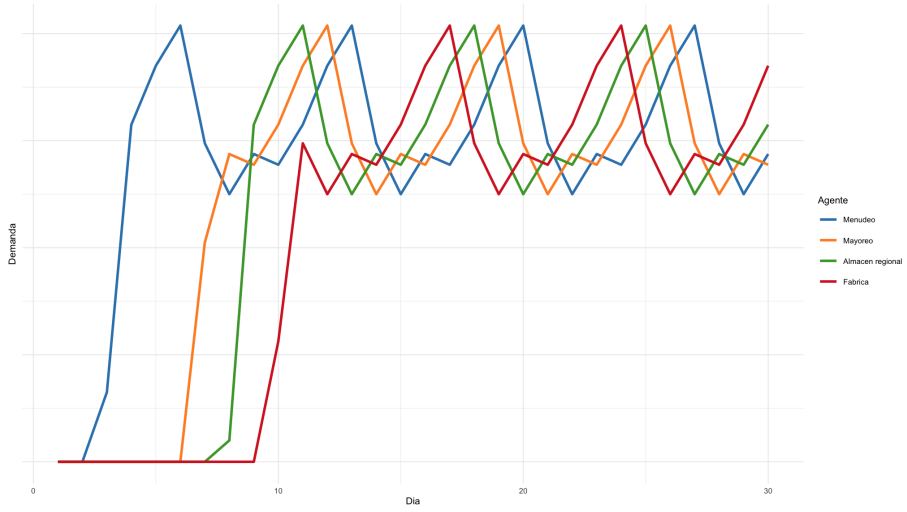
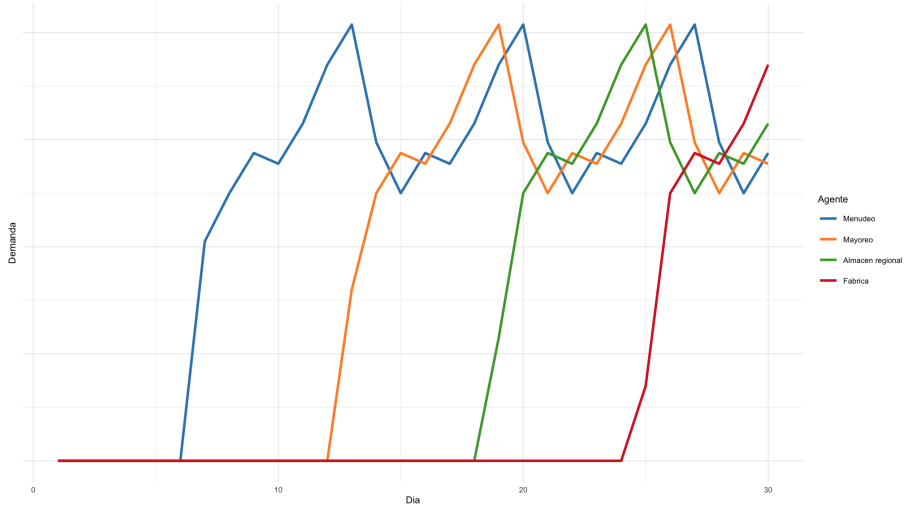


Figura 4.2: Retraso en visibilidad de la información para agentes en la cadena



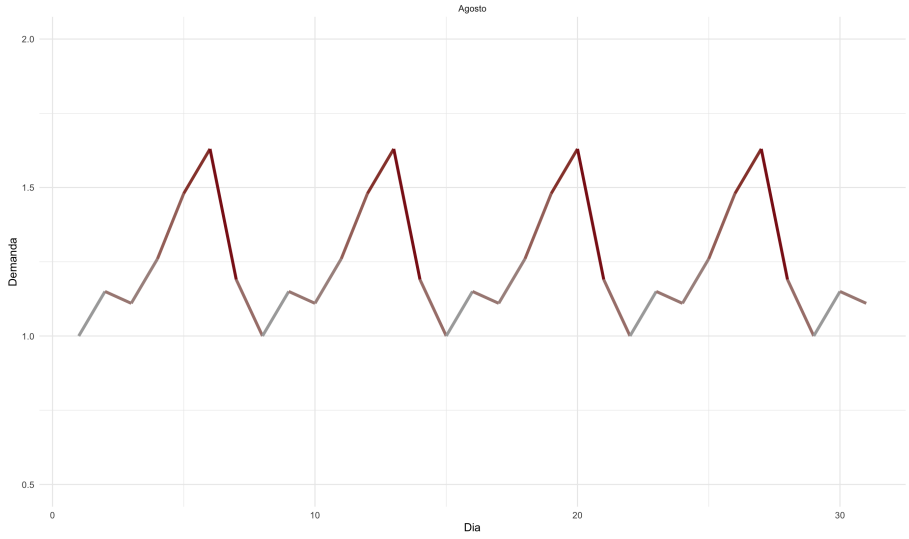
tendrá una aproximación relativamente buena. En el caso pesimista, supondrá que la demanda del consumidor durante esos días fue efectivamente cero, y sin duda causará una burbuja de falta de inventario que se propagará hacia abajo en la cadena de suministro. El efecto látigo habrá hecho de las suyas.

4.2. Estacional y aleatoria: más parecido al mundo real

En el mundo real, ningún producto tiene una demanda perfectamente constante. Más aún, existen una infinidad de productos cuya demanda varía con un patrón estacional: por ejemplo, las medicinas antigripales se venden más en invierno. La cerveza (y, en general, las bebidas alcohólicas) también siguen este tipo de patrones.

En primer lugar, existe un patrón semanal bastante esperado: según Saad [12], se consume más cerveza los fines de semana (ver la figura 4.3). Además, existe un patrón relacionado a las festividades comunes: por ejemplo, en EUA el consumo de bebidas alcohólicas se duplica en las festividades navideñas.

Figura 4.3: Ejemplo de un Mes Típico



Para el presente trabajo, se combinarán los dos patrones descritos anteriormente, y se añadirá un pequeño componente aleatorio que cambiará la demanda cada año. El comportamiento "base" de la demanda se puede consultar en la figura 4.4. En ella, se puede vislumbrar el patrón de aumento cada fin de semana, así como un incremento en las festividades navideñas y un pico creado por la fiesta de Independencia de México. Un ejemplo del comportamiento base, pero perturbado por un poco de aleatoriedad se puede consultar en la figura 4.5. Sin embargo, cabe mencionar que este es

un modelo simple, pues en algunas regiones, podría también existir un patrón relacionado al clima, o incluso un alza en la demanda siguiendo temporadas deportivas.

Figura 4.4: Demanda Anual de Cerveza

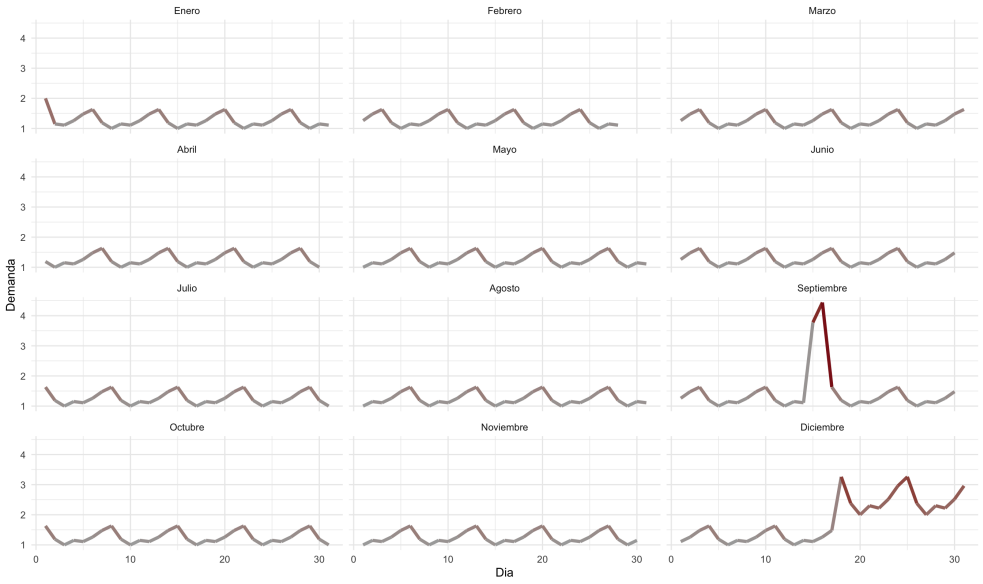
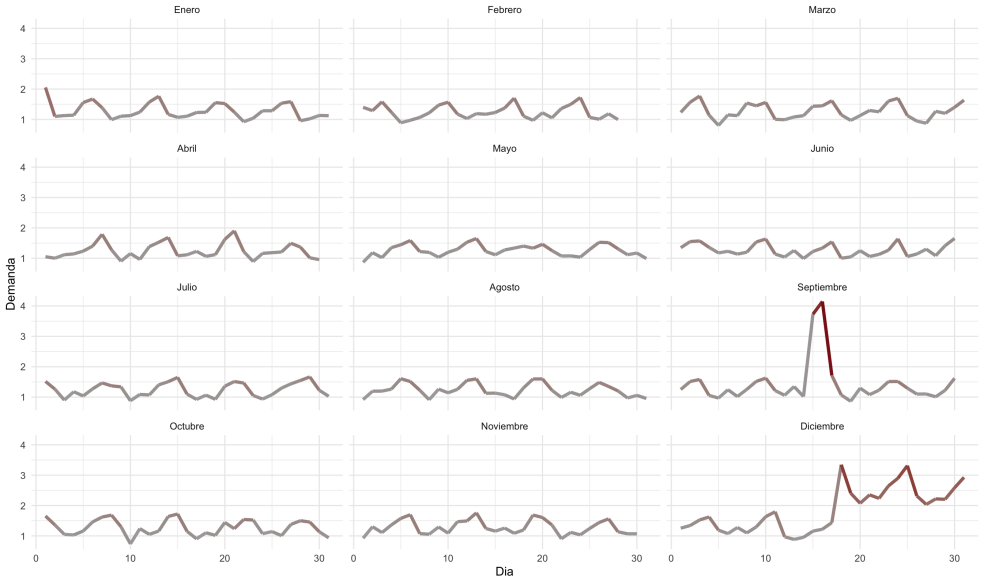


Figura 4.5: Realización de una Demanda Anual de Cerveza con ruido



Como el comportamiento varía a lo largo del año y existe un costo por almacenar inventario, cada agente debe prepararse con adecuada anticipación para los picos de demanda. Además, el componente aleatorio resulta en la necesidad de conservar inventario de seguridad (en inglés, *safety stock*) para asegurar que la demanda será cubierta la mayor parte del tiempo.

4.3. Q-Learning

Chaharsooghi et al. [3] publicó ya un acercamiento a este problema usando *Q-learning*, sin embargo la solución que encuentra implica que los inventarios (al final del día, después de las transacciones del día) se aproximan a cero, consecuencia directa de que los agentes minimizan su costo de inventario. Además, la demanda del consumidor obedece a un patrón aleatorio.

Sin embargo, ninguna de estas soluciones considera dos factores importantes del mundo real: la temporalidad de la demanda (por ejemplo, para la cerveza la cantidad demandada es mayor durante el fin de semana) y la restricción de temporalidad de producción de materias primas. En el mundo agrícola, los campos solamente producen ciertas cosechas en ciertos periodos, y es necesario tener suficiente inventario para cubrir la demanda durante todo el año. En el siguiente capítulo, se presentará este escenario como un nivel extra de complejidad al modelo, limitando la oferta de cebada por parte de los campos.

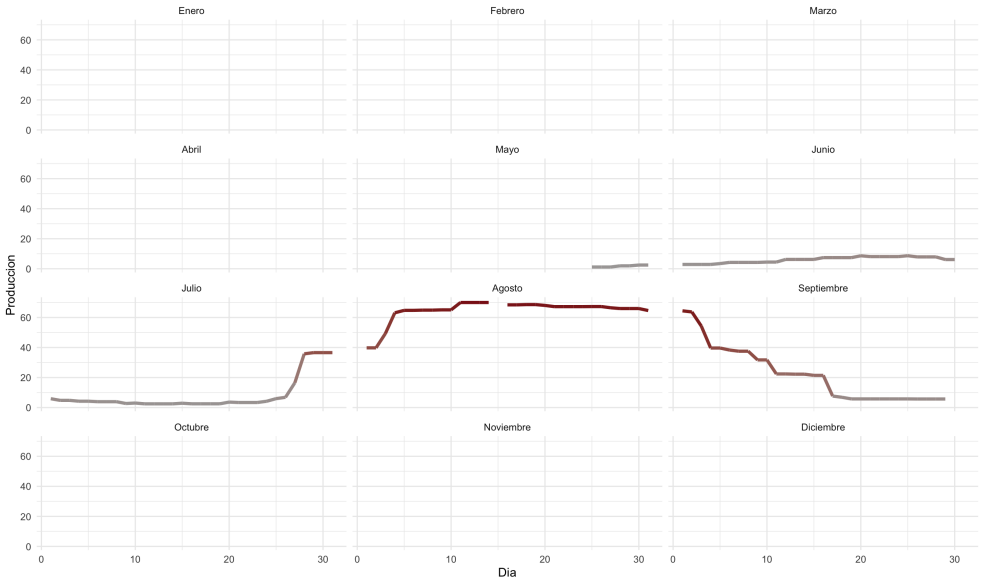
Capítulo 5

La Complicación: Restricción de Estacionalidad en los Campos

En su mayor parte, las soluciones al juego de distribución de cerveza que se han propuesto anteriormente suponen que el agente al final de la cadena (el proveedor de materias primas) tiene inventario infinito. Sin embargo, en el mundo real esta situación no se presenta: especialmente en materias primas que provienen del campo, existen ciclos de siembra y cosecha. Para construir el modelo de la manera más aplicable a la realidad posible, se puede agregar la restricción correspondiente: el productor de materias primas solamente lo hace en ciertos momentos del año.

Utilizando los datos más recientes disponibles del departamento de agricultura de los Estados Unidos de América (ver of Agriculture [11]) se han obtenido los ciclos naturales de uno de los principales componentes de la cerveza, la cebada, los cuales se pueden consultar en la figura 5.1. Es posible observar que la producción es nula entre octubre y mayo, con la mayor parte concentrada en agosto y comienzos de septiembre. Dependiendo de la magnitud de la demanda comparada con la oferta, esto podría significar que la cadena de suministro tiene que surtirse de cebada durante este periodo para poder cubrir la demanda del resto del año, haciendo uso de la capacidad de almacenamiento en sus respectivos almacenes.

Figura 5.1:



En la figura 5.2 se puede ver el comportamiento de la demanda del consumidor(el cual fue descrito en capítulos anteriores) en rosa; el de la producción en los campos, ahora con restricción, en verde; y la demanda de cada agente, en negro.

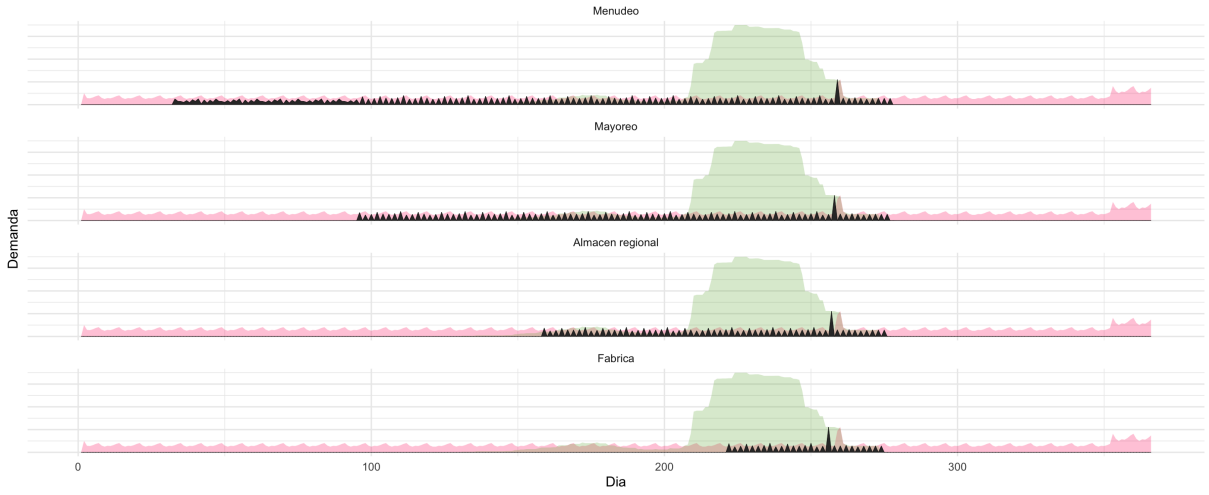
Si todos los agentes tomaran las decisiones relacionadas a la simple máxima "pide hoy lo que esperas vender mañana" sin tener almacenamiento, entonces el resultado estaría lejos de ser el óptimo. En este escenario se presentan varios efectos:

- Cada agente cuenta con inventario inicial, así que tarda en comenzar a hacer pedidos al agente inmediatamente superior
- Cada agente deja de pedirle al agente inmediatamente superior cuando el segundo ya no tiene inventario. Esto es especialmente notorio para la fábrica: comienza a tener demanda positiva cuando ya hay producción en los campos, cerca del día 150
- Si la oferta es más baja que la demanda, los agentes piden todo lo que haya disponible, pues es preferible cubrir la demanda parcialmente que no cubrirla en lo absoluto
- Los agentes se preparan adecuadamente para el pico de septiembre, con un poco más de

anticipación a medida que se encuentran más lejos del consumidor

- Sin embargo, ninguno de los agentes puede aprender que debería comprar mucho más en el periodo de producción de los campos (en verde) para poder cubrir la demanda de todo el año (en rosa) y maximizar su ganancia

Figura 5.2: Demanda de cada agente durante el año, suponiendo falta de almacenes



Como se puede observar en la figura 5.2, los agentes no se prepararon adecuadamente para cubrir la demanda posterior al tiempo en que los campos tuvieron producción positiva. Esto sucede porque, en este modelo, como no existen almacenes, los agentes ven solamente un periodo hacia el futuro, así que solo se preparan para ese día. Esta complicación vuelve imperante que todos los agentes usen sus almacenes para poder afrontar la demanda, incluso cuando no hay producción. Tal solución se vuelve compleja porque hay muchos parámetros en juego, por mencionar algunos:

- El costo diario de almacenamiento influye directamente en qué tanto tiempo un agente puede mantener la cerveza en inventario para cubrir una venta futura, tal que aún obtenga un margen positivo cuando esta suceda. Si es relevante asegurar que nunca tendrán cerveza guardada durante más de un año, o durante cualquier periodo relacionado, por ejemplo, con la caducidad del producto, es necesario que se cumpla la restricción:

$$margen_{agente} <= 365 * almacenamientoagente$$

- El castigo por orden no cumplida (*metapolicy*) también juega un papel importante en la interacción anterior: si es suficientemente grande, incluso podría resultar en que un margen negativo causado por el costo de almacenamiento es aún preferible a las consecuencias de perder la venta. Si se busca asegurar que siempre sea preferible buscar una venta, entonces es necesario que se cumpla la restricción:

$$castigo_{agente} >= margen_{agente} + 365 * almacenamientoagente$$

- Si la producción total en el año es mayor que la demanda total durante el mismo periodo, no es necesario comprarle toda la cebada a los campos. Paralelamente, si la producción es menor a la demanda, entonces es necesario comprar toda la cebada disponible, pues completar algunas ventas siempre es preferible a no completar ninguna

Sin embargo, algunas de estas condiciones son teóricas y no forzosamente deben cumplirse: es posible que mantener una cerveza en el inventario desde finales de septiembre hasta principios de mayo (cuando vuelve a haber producción) sencillamente no sea redituable.

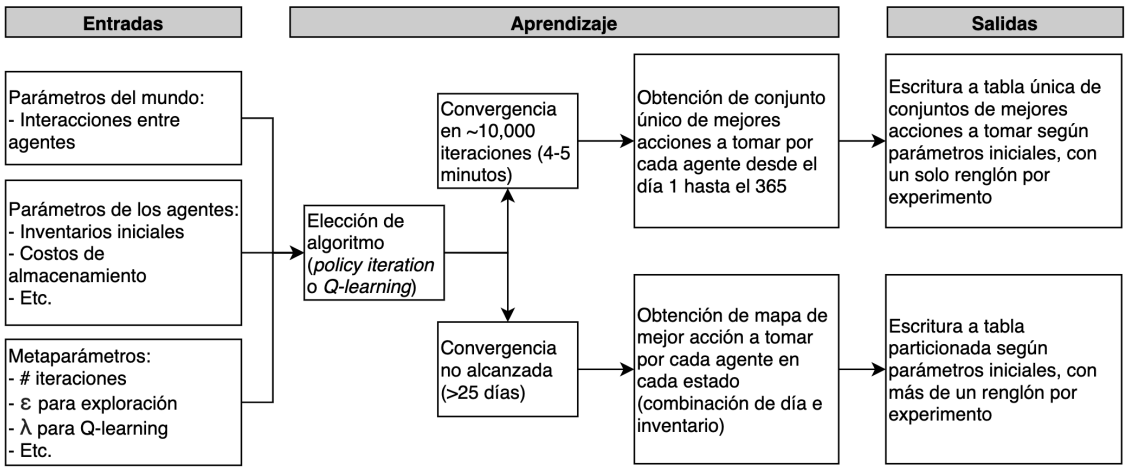
En este trabajo presentaremos un modelo con esta restricción, y lo resolveremos tanto con *policy iteration* como con *Q-learning*. Ambos modelos de aprendizaje reforzado presentados en esta tesis son capaces de capturar todas estas interacciones e incorporarlas al aprendizaje de cada agente.

Capítulo 6

El Producto de Datos

Se creó un *pipeline* completo que toma los parámetros iniciales, los cuales consisten tanto en condiciones del mundo como en los datos históricos de demanda y oferta; ejecuta el modelo de aprendizaje elegido (ya sea *policy iteration* o *Q-learning*; y por último inserta los resultados en una base de datos para hacer posible consulta y análisis posteriores.

Figura 6.1: *Pipeline* del proceso



Al construir la base en la cual se albergan los resultados, se adoptaron las mejores prácticas para creación de bases de datos, esquemas y tablas: siguen una estructura específica comúnmente

llamada *tidy* (limpia), en la cual cada variable es una columna, cada observación una fila y cada tipo de unidad observacional es una tabla.

Dado que cada uno de los algoritmos de aprendizaje, *policy iteration* y *q-learning*, tiene requerimentos diferentes y la forma en que presenta resultados es distinta, en la base de datos se construyeron dos esquemas respectivamente, cada uno portando el nombre del algoritmo respectivo.

6.1. Policy Iteration

Es necesario crear tablas que contengan los parámetros necesarios: una para el mundo (tales como la demanda y oferta), otra para los agentes (tales como precios de venta, inventarios iniciales) y una última para los experimentos (los hiperparámetros del algoritmo, tales como ϵ y λ). Por otro lado, dado que este método produce una política en forma de vector de tipo numérico, la manera óptima de almacenar los resultados es en una tabla que contenga como llave el identificador (id) del experimento y el agente, y un renglón por resultado.

Como llave de las tablas, se asigna el identificador (id) del experimento como el *hashlib*[1] de la concantenación de ciertos atributos de este: el sello de tiempo en el cual fue ejecutado y las *policies* óptimas de cada uno de los agentes. Esto crea un identificador único y también permite revisar que ningún dato haya sido modificado manualmente, en caso de que tal revisión sea pertinente.

Se obtienen un total de 4 tablas (una por agente) que se relacionan por medio del identificador del experimento.

[1] Las funciones de *hashing* toman datos de tamaño arbitrario y los mapean a un *hash* de longitud fija.

6.2. Q-Learning

El resultado de este algoritmo es una tabla que relaciona a cada par (s, a) (estado, acción), con un valor de la función Q, así que no es práctico utilizar el mismo formato de tabla de resultados que con *policy iteration*, en el cual un renglón contiene un resultado. Se crea entonces una tabla por resultado, y una tabla extra que relaciona el identificador (id) del experimento con el nombre respectivo de la tabla. El resto de la estructura del esquema es parecido al de *policy iteration*, pues también existe necesidad de relacionar los parámetros de cada experimento con sus resultados. Debido a la estructura de la base, el número de tablas es variable.

6.3. Especificaciones técnicas

Para el proceso se utilizaron PostgreSQL 10 y Python 3.6.5 en la distribución contenida en Anaconda, así como varios paquetes de Python descritos en el archivo de requerimentos del repositorio de Github. Todos los procesos fueron ejecutados en una computadora Macbook Air con 8 GB de RAM y 1 procesador tipo intel i5 con 2 cores.

Capítulo 7

Resultados

En esta sección se presentará el código utilizado para el modelo de aprendizaje reforzado, así como los resultados en cada experimento realizado. El código completo puede consultarse en el repositorio público $https://github.com/fleoren/Tesis\_Maestria$. Para maximizar la reusabilidad y la posible exposición a otras audiencias, tal código está escrito y documentado en inglés.

Durante el aprendizaje se realizaron numerosos experimentos con diferentes metaparámetros (e.g. ϵ para el algoritmo $\epsilon - greedy$), parámetros del mundo (e.g. la penalización por órdenes no cumplidas) y parámetros de los agentes (e.g. los márgenes de ganancia de cada uno). En esta sección se presentarán algunos análisis y resultados relevantes.

En general, el método *policy iteration* convergió rápidamente, siendo capaz de encontrar estrategias óptimas para todos los agentes en menos de 10 minutos. Asimismo, se encontró que en la gran mayoría de los casos (obtenidos con simulaciones), se obtiene un mejor desempeño para todo el sistema de cadena de suministro favoreciendo la estrategia inteligente sobre métodos comunes, incluso si solamente uno de los agentes lo hace.

7.1. Código: el Modelo, el Mundo y los Agentes

A pesar de que nuestro sistema tiene seis roles, en realidad solamente cuatro de ellos toman decisiones:

- El consumidor **obedece** a su necesidad interna de cerveza, mayor en fines de semana y festividades
- La tienda minorista **decide** cuánta cerveza va a comprar a la tienda de mayoreo
- La tienda de mayoreo **decide** cuánta cerveza va a comprar al almacén regional
- El almacén regional **decide** cuánta cerveza va a comprar a la fábrica
- La fábrica **decide** cuánta cebada va a comprar a los campos
- Los campos **obedecen** a sus ciclos de siembra y cosecha

A pesar de que, estrictamente, en nuestro sistema multiagente todos los roles son agentes, por claridad en la descripción y el flujo del código nos referiremos como agentes solamente a aquellos roles que toman decisiones. Las ecuaciones de los agentes en los extremos se pueden conectar fácilmente al sistema manteniendo la misma lógica si suponemos que, para el consumidor, su *policy* óptimo es su demanda normal durante el año, y para los campos, es su producción.

Para construir todo el sistema, son necesarios varios elementos: los metaparámetros del modelo, los parámetros globales del mundo, los parámetros individuales de los agentes, los agentes como entidades funcionales, y las relaciones entre los agentes.

7.1.1. El modelo: metaparámetros

Existen cinco parámetros que son necesarios para asegurar el buen funcionamiento del modelo:

- Semilla para realizaciones aleatorias, para permitir replicabilidad de experimentos, con valor permanente 20170130

- Total de épocas (*epochs*) de entrenamiento, es decir, cuántas iteraciones hará el modelo antes de llegar a un resultado final, con valor base $10,000$ pero flexible para diferentes experimentos
- Tiempo de comienzo, para que cada agente observe las cantidades demandadas por el agente inferior y calibre correctamente la magnitud de su propia demanda, con valor permanente $0{,}05\,\%$
- Epsilon (ϵ), que controla la proporción de explotación contra exploración y cuyo uso fue detallado en el capítulo Aprendizaje Reforzado, con valor permanente $0{,}05\,\%$
- Lambda (λ), que controla el descuento de recompensas obtenidas en tiempos t más adelantados que el actual y cuyo uso fue detallado en el capítulo Aprendizaje Reforzado, con valor permanente $90\,\%$
- Demanda máxima en un día para cualquier agente, para mantener las realizaciones dentro de un rango razonable, con valor permanente 75, el cual es mucho mayor que la demanda diaria del consumidor (cuyo valor está siempre por debajo de 4 unidades)

7.1.2. El mundo: parámetros globales

El mundo contiene a los agentes, por lo tanto la mayor parte de los parámetros le pertenecen a estos. Sin embargo, el mundo necesita dos controles importantes para asegurar que el algoritmo aprende correctamente, los cuales podrían definirse a nivel agente pero por simplicidad se mantendrán globales:

- Costo por órdenes no cumplidas: valor base 2, idealmente debe mantenerse debajo del margen más pequeño de los agentes para asegurar que es preferible vender cerveza que no hacer nada
- Costo de almacenamiento: valor base $10/365 = 0{,}027$; este debe ser mayor a cero para asegurar que los agentes no almacenan todo en el día 1, suficientemente alto como para impactar en el margen si se almacena demasiado, pero suficientemente bajo como para que no sea redituable mantener cerveza almacenada más de un año

Por último, dado que solamente se referirá como *agentes* a aquellos que aprenden, se declaran las tendencias de demanda de consumidor y de oferta de los campos dentro de esta sección. Estas

tendencias se encuentran en dos archivos formato *.csv* para permitir flexibilidad a cambios.

7.1.3. Los agentes: funcionamiento e interrelaciones

Cada uno de los agentes es creado como entidad funcional con el siguiente código, ejemplificado para el agente de menudeo (*retail*). La única información específica que necesita cada agente es un nombre, su inventario inicial, sus precios de venta al agente inferior y compra al agente superior, y el costo de almacenamiento (el cual está definido de manera global). Todos los demás parámetros necesarios son establecidos al crear el mundo.

```
retail_agent = Agent(name = "Retail",
                     inventory = retail_ininv,
                     selling_price = retail_price,
                     buying_price = wholesale_price,
                     warehouse_price = warehouse_price)
```

Cada agente es creado como una clase *Agent* con varios métodos que gobiernan la manera en que interactúan con el mundo y los otros agentes, como el proceso de compra/venta de cerveza a los agentes superiores e inferiores, y el pago recurrente por almacenamiento. Estos métodos se verán más adelante, cuando se expongan las ecuaciones que los definen.

Una vez que los metaparámetros, los parámetros globales del mundo y los agentes han sido creados, el mundo se inicializa de la siguiente manera:

```
# Creating Supply Chain Agents
customer_agent = Customer(customer_demand)
retail_agent = Agent(name = "Retail",
                     inventory = retail_ininv,
                     selling_price = retail_price,
                     buying_price = wholesale_price,
                     warehouse_price = warehouse_price)
wholesale_agent = Agent("Wholesale",
```

```
                    wholesale_ininv,
                    wholesale_price,
                    regional_warehouse_price,
                    warehouse_price)
regional_warehouse_agent = Agent("Regional_Warehouse",
                    regional_warehouse_ininv,
                    regional_warehouse_price,
                    factory_price,
                    warehouse_price)
factory_agent = Agent("Factory",
                    factory_ininv,
                    factory_price,
                    field_price,
                    warehouse_price)
fields_agent = Fields(fields_supply)

# Assigning interactions
retail_agent.downstream_agent = customer_agent
retail_agent.upstream_agent = wholesale_agent
wholesale_agent.downstream_agent = retail_agent
wholesale_agent.upstream_agent = regional_warehouse_agent
regional_warehouse_agent.downstream_agent = wholesale_agent
regional_warehouse_agent.upstream_agent = factory_agent
factory_agent.downstream_agent = regional_warehouse_agent
factory_agent.upstream_agent = fields_agent
```

Como se puede notar, los "agentes" a los extremos de la cadena se inicializan de manera diferente. Dado que ambos obedecen estrategias fijas, en lugar de decidir y tener que aprender, el único parámetro necesario es el comportamiento que deben obedecer.

Cada uno de los agentes que **deciden**, y por lo tanto pueden aprender, son inicializados con la misma clase *Agent*, pero con sus parámetros de inventarios y precios propios. Por último, se establecen las relaciones entre los agentes para crear la cadena. Esta manera de construir y definir explícitamente los agentes y sus relaciones permite fácilmente añadir o remover agentes de la cadena, si fuera necesario hacerla más corta o larga.

Durante el aprendizaje, todos los agentes siguen el mismo conjunto de reglas (aprender cuánto tienen que comprarle al agente superior en cada día, maximizando la ganancia que obtienen de venderle al agente inferior, al tiempo de minimizar los costos de almacenamiento y la penalización por órdenes no cumplidas), así que las ecuaciones de ganancia y aprendizaje son las mismas. En este trabajo, se utilizará la siguiente construcción del mundo, con cierta notación para los distintos parámetros del mundo:

1. Para cada agente a, el agente superior en la cadena de suministro es $a+1$; el inferior es $a-1$ y se representan con subíndices para cada atributo
2. Cada agente tiene un precio de venta p_a, un costo de almacenamiento c_a y un costo por orden no cumplida (*backlog*) b_a
3. En cada día d, cada agente a recibirá del agente superior $a+1$ la cantidad demandada por el primero en el día $d-1$, sujeto a que tal cantidad sea menor o igual a la cantidad que el agente $a+1$ tenía en el almacén
4. En cada una de estas transacciones, cada agente recibe el dinero equivalente, basado en su respectivo precio de venta, e incurre en costos de almacenamiento y por órdenes no cumplidas

Esto quiere decir que, independientemente del algoritmo de aprendizaje que usemos, cada agente a calcula su ganancia en el día d de la siguiente manera:

$$\begin{aligned} ganancia_{a,d} = &(p_a * ventas_{a,d}) \\ &-(b_a * (demanda_{a-1,d} - ventas_{a,d})) \\ &-(p_{a+1} * ventas_{a+1,d-1}) \end{aligned}$$

$$-(c_a * inventario_{a,d-1})$$

Es decir, cada agente recibe el dinero correspondiente a las ventas que hace al agente inferior, paga el costo castigo relacionado a las órdenes no cumplidas, paga las órdenes hechas al agente superior y paga el costo de almacenamiento.

Dentro del método existen tres funciones para lograr esta ecuación:

```
def give_downstream(self, orders):
    # Checks if he has availability to fulfill order,
    # fulfills as much as he can
    if self.inventory >= orders:
        self.total_money = self.total_money + \
            (orders * self.selling_price)
        self.inventory = self.inventory - orders
        return orders
    else:
        orders_that_could_be_fulfilled = self.inventory
        # Sells all its inventory
        self.total_money = self.total_money + \
            (orders_that_could_be_fulfilled * self.selling_price)
        # If there were non fulfilled orders, those cause a penalty
        self.backlog = (orders - self.inventory) * backlog_cost
        self.total_money = self.total_money - self.backlog
        self.inventory = 0
        return orders_that_could_be_fulfilled

def receive_upstream(self, orders):
    # Receives orders from upstream agent first thing in the morning
    self.inventory = self.inventory + orders
    self.total_money = self.total_money - \
```

```
            (orders * self.buying_price)

def pay_for_warehousing(self):
        # Pays for warehousing of inventory: must be done either
        # "first thing in the morning" or "last time in the night"
        self.total_money = self.total_money - \
                (self.inventory * warehouse_price)
        self.total_warehousing_costs = self.total_warehousing_costs + \
                (self.inventory * warehouse_price)
```

Cada uno de estos métodos mapea limpiamente a los renglones: los primeros dos se refieren al precio propio y las ventas al agente inferior; el tercero se refiere al precio del agente superior y las ventas del agente superior (por lo tanto, las compras del agente de interés); y por último el cuarto es el costo de inventario y las cervezas guardadas.

Es importante notar que las reglas de funcionamiento del mundo lo convierten en un ambiente que, si bien no es estocástico en el sentido estricto de la definición, tampoco es puramente determinista: un agente que tome la misma acción puede tener diferente recompensa de una realización a otra debido a un manejo de inventario diferente del agente superior. El sistema presenta un comportamiento más estocástico durante la fase de exploración, y tiende a uno completamente determinista durante la fase de explotación. Es por esto que en este trabajo, no se han incluido probabilidades de transición (indicadas como $T(s, \pi(s), s'$ en las ecuaciones de secciones anteriores) en el modelo empleado, cada acción tomada en un estado por un agente solo tiene una posible consecuencia, dada por las restricciones de inventario y los parámetros del mundo tales como costos y precios.

Si bien el código del modelo fue construido para tener flexibilidad en los parámetros del mundo (como los precios de compra y venta de los agentes, o los costos de almacenamiento), la mayor parte de los resultados presentados en esta sección fueron obtenidos con la misma combinación prudente de tales parámetros. Existen casos extremos, por ejemplo que todos los agentes comenzaran el año con más inventario del que necesitarán, y por lo tanto nunca necesitarán presentar una cantidad

demandada ante sus respectivos agentes superiores. Otro caso extremo podría surgir si el costo de almacenamiento fuera mayor al margen de ganancia y que la penalización por órdenes no cumplidas no compesara tal diferencia. En este trabajo no se explorarán estos casos extremos.

Una vez establecidas todas las relaciones entre todos los jugadores así como los parámetros del mundo, se han obtenido dos formas diferentes de solucionar el juego: la primera con *policy iteration* y la segunda con *Q-learning*. En este trabajo no se ahondará en los resultados de *Q-learning* dado que el algoritmo no convergió en un tiempo razonable, y también dado que *policy iteration*, a pesar de ser un algoritmo más simple, dio buenos resultados en todas las simulaciones.

7.2. Policy Iteration

Las ecuaciones que describen el aprendizaje de un agente son:

1. La ecuación de utilidad (recompensa) para cada agente a en el día d:

$$R(a,d) = r_{a,d} + \gamma * r_{a,d+1} + ... + \gamma^{364} * r_{a,d+364}$$

2. El vector de política para cada agente tiene la siguiente forma, guardando en cada elemento la cantidad demandada al agente superior en cada tiempo t, que en este caso es cada día del año:

$$\pi(s) = \{\pi(s)_{t=1}, \pi(s)_{t=22}, ..., \pi(s)_{t=365}\}$$

Se utilizan las ecuaciones planteadas en secciones anteriores para actualizar el vector de política óptima cuando se encuentra una nueva política con un valor mayor.

El código utilizado para el aprendizaje con *policy iteration* se encuentra a continuación. Los comentarios se han omitido, aunque los originales en inglés pueden encontrarse en el repositorio de *Github*, pues se ha procurado tener explicaciones en español en el documento. También se omitieron

las líneas de código que crean *logs* o archivos utilizados para visualizaciones.

Antes de inicializar el aprendizaje, el parámetro ϵ juega su papel para crear la cantidad demandada cada día. Puede ser aleatoria (exploración) o la cantidad aprendida (explotación); dependiento de la realización aleatoria del parámetro.

```
def create_demand(day):
    x = np.random.uniform(0, 1)
    if x < p_exploration:
        return np.random.randint(0,max_demand)
    else:
        return agent.best_policy[day-1]
```

Una vez que se ha definido esa función, se utiliza un iterador para el total de épocas a entrenar, los agentes y los días del año.

```
for j in range(total_epochs):
    p_exploration = max(epsilon_greedy_converges_to ,(total_epochs - j) /
↪   total_epochs)
    day = 0
    for agent in agents:
        agent.inventory = agent.initial_inventory
        agent.total_warehousing_costs = 0
        agent.total_money = 0
        agent.backlog = 0


    while day < 365:
        day+=1
        # Factory
        fulfilled_to_factory = min(factory_agent.current_policy[day-1],
                                   max(fields_agent.current_policy[day-1] -
↪   factory_agent.current_policy[day-1],0))
```

```
        factory_agent.receive_upstream(fulfilled_to_factory)
        factory_agent.current_policy[day-1] = fulfilled_to_factory
        # Regional Warehouse
        fulfilled_to_regional_warehouse =
↪ factory_agent.give_downstream(regional_warehouse_agent.current_policy[day-1])

↪ regional_warehouse_agent.receive_upstream(fulfilled_to_regional_warehouse)
        factory_agent.policy_inventory[day-1] = factory_agent.inventory
        regional_warehouse_agent.current_policy[day-1] =
↪ fulfilled_to_regional_warehouse
        # Wholesale
        fulfilled_to_wholesale =
↪ regional_warehouse_agent.give_downstream(wholesale_agent.current_policy[day-1])
        wholesale_agent.receive_upstream(fulfilled_to_wholesale)
        regional_warehouse_agent.policy_inventory[day-1] =
↪ regional_warehouse_agent.inventory
        wholesale_agent.current_policy[day-1] = fulfilled_to_wholesale
        # Retail
        fulfilled_to_retail =
↪ wholesale_agent.give_downstream(retail_agent.current_policy[day-1])
        retail_agent.receive_upstream(fulfilled_to_retail)
        wholesale_agent.policy_inventory[day-1] = wholesale_agent.inventory
        retail_agent.current_policy[day-1] = fulfilled_to_retail
        # Customer
        fulfilled_to_customer =
↪ retail_agent.give_downstream(customer_agent.current_policy[day-1])
        retail_agent.policy_inventory[day-1] = retail_agent.inventory

        for agent in agents:
```

```
            agent.current_payout[day-1] = agent.total_money
            if j == np.floor(total_epochs*agent.time_for_zero_policy):
                agent_demand = 0
            if j < warmstart_proportion * total_epochs:
                agent.average_downstream_demand =
↪ np.mean([agent.average_downstream_demand,

↪ agent.downstream_agent.current_policy[day-1]])
                agent_demand = np.round(agent.average_downstream_demand)
            else:
                agent_demand = create_demand(day)
            agent.current_policy[day-1] = agent_demand
            agent.pay_for_warehousing()
            if agent.current_payout[day-1] > agent.best_payout[day-1]:
                agent.best_policy[day-1] = agent.current_policy[:][day-1]
                agent.best_payout[day-1] = agent.current_payout[:][day-1]
```

El iterador principal controla la cantidad de veces que se recorrerán los 365 días para cada agente, con el metaparámetro de épocas totales. En cada nuevo año (cada nueva iteración) se reinicializan tanto el metaparámetro ϵ como los inventarios iniciales y el dinero de cada agente, mas no así sus políticas aprendidas.

Cada año (que tiene 365 días), cada agente irá tomando decisiones diarias y monitoreando cómo afecta eso a su recompensa, la cual se estableció como el dinero al final del experimento. Así, existen 4 bloques de código sumamente similares, solamente diferenciados por el nombre del agente que aprende. Por último, hay dos líneas adicionales para el "agente" consumidor: aunque él no aprende, es importante contabilizar correctamente las órdenes que le fueron cumplidas por el agente de menudeo.

El último iterador en el código, para cada agente, es donde sucede el aprendizaje. Primero, si

todavía se encuentra en el periodo de observación (controlado por un metaparámetro) asignará como política actual el promedio de la demanda observada de su agente inferior. En otro caso, monitorea si la acción tomada el día anterior resultó en una mejor recompensa en el día actual; si es el caso, reemplaza en el vector de política óptima; si no, conserva el vector óptimo que tuviera anteriormente. De esta manera, acciones que resulten en mejores recompensas irán construyendo la política a seguir al final.

Es muy importante notar que todos los agentes son creados con el mismo método *Agent* y utilizan exactamente el mismo algoritmo para aprender, pero podrían aprender *policies* diferentes o ser impactados en menor o mayor medida por el efecto látigo.

Cabe destacar que el vector de política óptima puede ser inicializado de cualquier manera. De manera razonable podría ser un vector de ceros, lo cual representa una estrategia de completa inacción, un vector cuyos elementos son realizaciones de un generador de números aleatorios para no imponer ninguna estrategia, etc. En este caso, se decidió tener un periodo corto de observación al agente inferior, para tener una política de línea de base que podría juzgarse de sentido común, y como presión hacia una estrategia conservadora en contraste con una aleatoria. Así, todos los agentes inicializan su entrenamiento en un nivel relativamente estable alrededor del promedio de la demanda del agente inferior.

7.2.1. Tiempo de entrenamiento

Se hicieron pruebas con distintas iterationes totales y el tiempo que tomó realizarlas:

Iteraciones	Tiempo
1,000	25 seg
10,000	4 min
100,000	40 min
1,000,000	6.5 h

Se puede notar que el tiempo necesario para iteraciones crece de forma lineal, pues los agentes no

deben aumentar la cantidad de información que aprenden a medida que el tiempo pasa. Incluso si se decidiera tomar la opción de $1,000,000$ de iteraciones, el modelo entrena suficientemente rápido como para reentrenarlo diario de ser necesario (ante cambios abruptos en demanda u oferta).

7.2.2. Mundo ideal: agentes inteligentes

Los agentes aprenden estrategias que siguen el comportamiento de la oferta, lo cual es el comportamiento esperado dado que en las especificaciones iniciales, se definió manualmente que la demanda anual no sobrepasara la oferta anual. Este resultado puede observarse en las figuras 7.1 a 7.4, en donde se presentan las políticas encontradas con $1k$, $10k$, $100k$ y $1M$ iteraciones.

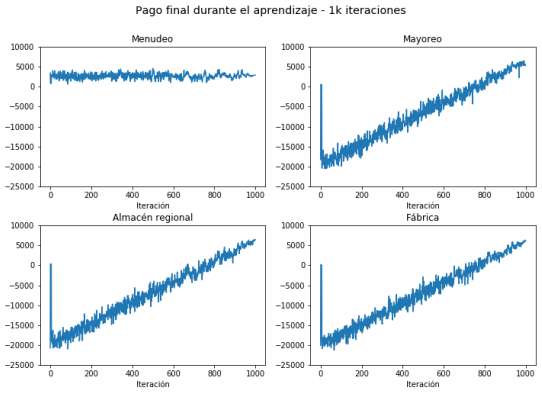


Figura 7.1: Evolución de recompensas con 1k iteraciones

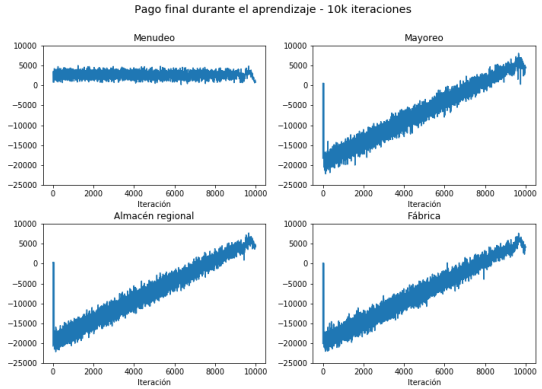


Figura 7.2: 10k iteraciones

Es esperado observar que todos los agentes presentan una tendencia positiva al encontrar mejores *policies* en cada iteración del algoritmo. Sin embargo, es interesante notar que la función de valor del agente Menudeo empieza (después de un periodo de pre-calentamiento de 0,5 % del total de iteraciones) en un punto mucho más cercano al óptimo que los otros agentes. Esto puede explicarse porque, dada la configuración del mundo, el agente Menudeo tiene información directa de la demanda del consumidor, y por lo tanto puede ajustar su política mucho más rápidamente.

El algoritmo converge de manera relativamente rápida: en las figuras 7.1 a 7.4 se puede observar que, con diferente número de iteraciones, el comportamiento del crecimiento de la función de valor

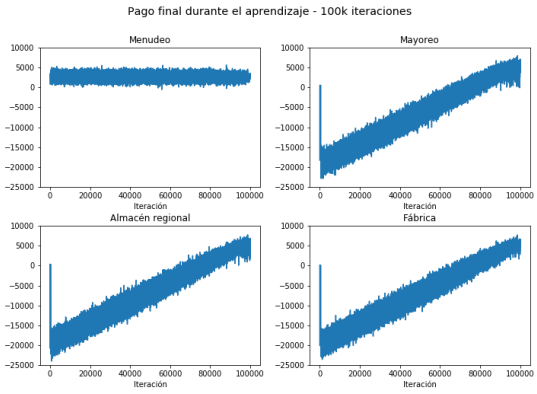


Figura 7.3: 100k iteraciones

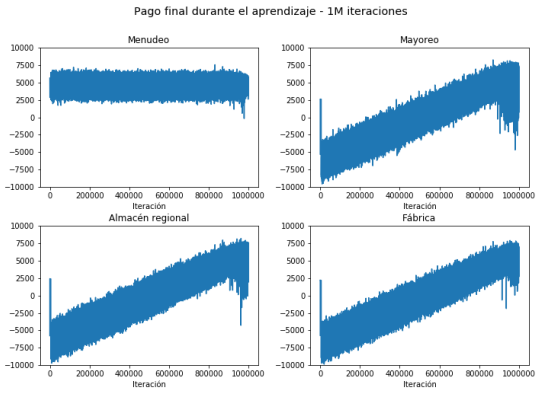
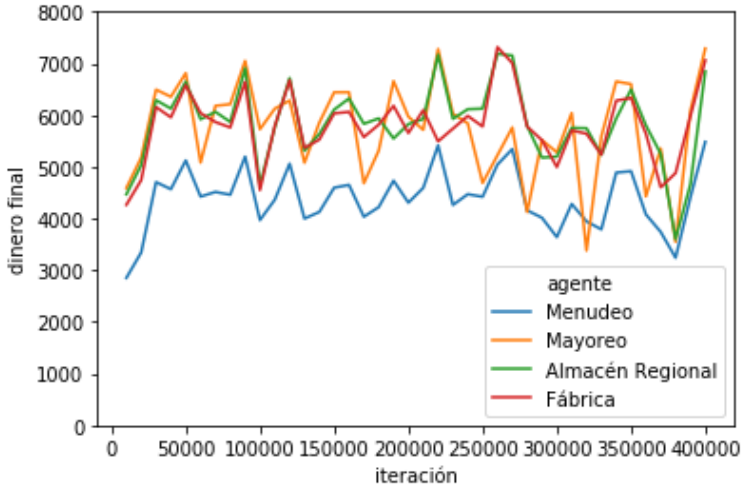


Figura 7.4: 1M iteraciones

(inferida de la política óptima en la iteración i) es similar. Asimismo, en las figuras 7.6 a 7.9 se puede observar una comparación de las políticas obtenidas con los mismos umbrales de iteraciones: la forma es extremadamente parecida sin importar la cantidad de iteraciones. Este fenómeno nos indica que no es necesario un número alto de iteraciones, pues se encuentra el valor máximo y la forma correcta de la política óptima con solamente $10,000$ iteraciones. Con las condiciones de *hardware* y *software* especificadas en secciones anteriores, este proceso tarda un poco menos de 5 minutos.

Otra manera de analizar la rápida convergencia al óptimo del algoritmo es comparando las recompensas finales (asociadas a la estrategia óptima encontrada) para cada agente, con diferente número de iteraciones. En la figura 7.5 se puede observar que no es necesario un número alto de iteraciones, ya que el resultado final es sumamente estable.

Figura 7.5: Comparación de recompensa final con diferente total de iteraciones



Las políticas aprendidas tienen el comportamiento esperado: todos los agentes compran cantidades altas durante ambos picos de producción en verano, con la finalidad de abastecerse lo más posible para las ventas del resto del año. Es importante recordar que estas políticas óptimas dependen de la selección de parámetros del mundo (precios, costos, etc.) en gran medida, así como los metaparámetros (tiempo de calentamiento, ϵ, etc.), lo que implica que es necesario realizar un entrenamiento nuevo en caso de que estos cambien, o si de un año a otro los agentes tienen, por ejemplo, diferentes inventarios iniciales. Este problema puede ser resuelto entrenando un modelo como *Q-learning*, opción que se explorará de manera superficial más adelante en esta sección.

Durante el análisis de las políticas y sus resultados, salió a la luz un fenómeno no esperado que afecta de manera distinta a los agentes, tal que en algunas iteraciones resultan con recompensas más bajas de lo esperado. Para profundizar en esto, se exploró la posibilidad de que no todos los agentes tomen decisiones óptimas, cuyos resultados son presentados en el siguiente apartado.

7.2.3. Agentes con diferentes niveles de aprendizaje

Se ha hablado ya de las estrategias óptimas obtenidas por el algoritmo de *policy iteration*, sin embargo es de suma importancia comparar el potencial beneficio que podrían tener los agentes

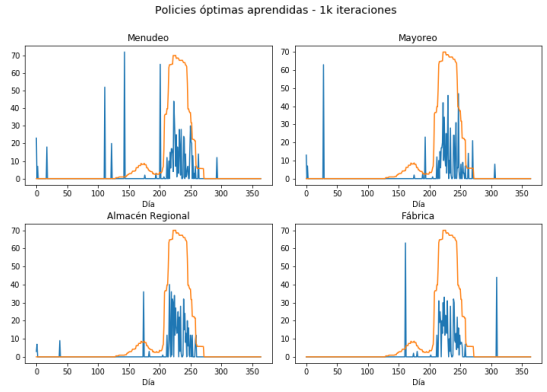


Figura 7.6: Evolución de recompensas con 1k iteraciones

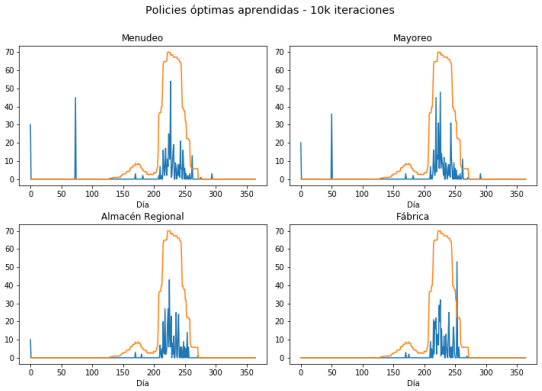


Figura 7.7: 10k iteraciones

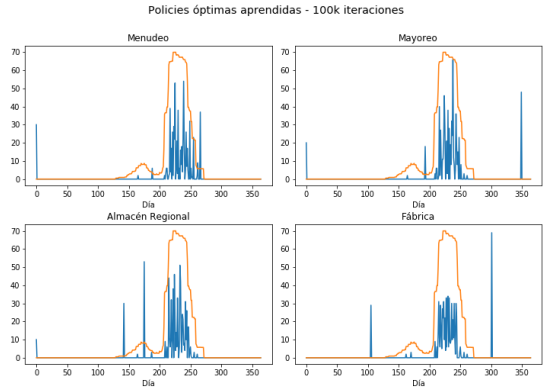


Figura 7.8: 100k iteraciones

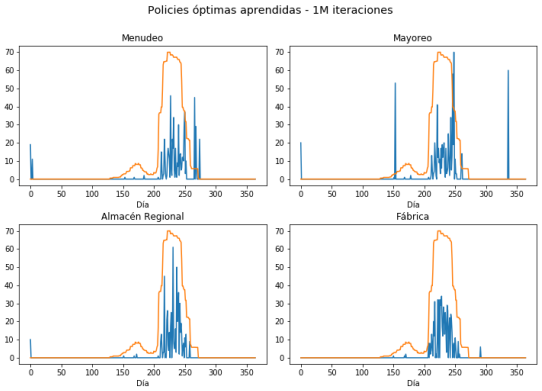


Figura 7.9: 1M iteraciones

tomando como línea base una estrategia creíble pero poco optimizada. Se busca responder a preguntas como: ¿vale la pena construir, entrenar y seguir el modelo, comparado con una estrategia de sentido común?, o ¿es preferible usar la estrategia encontrada por el modelo, incluso si los demás agentes en la cadena toman otro acercamiento?

Para explorar estos escenarios, se definen dos tipos de estrategias y varias combinaciones de ellas. La primera estrategia, relativamente básica, es una que dictaría el sentido común: cada agente calcula el promedio de peticiones del agente inferior durante los últimos 5 días, y fija ese número como su demanda al agente superior. Por otro lado, la segunda estrategia, óptima según el re-

sultado del algoritmo *policy iteration*, responde tanto a un periodo más largo, como a factores adicionales tales como el costo de oportunidad de perder una orden comparado con el costo de almacenamiento. En el mundo ideal, que se exploró en la sección anterior, todos los agentes siguen la estrategia encontrada por el modelo. Sin embargo, es posible que una situación más realista contenga una mezcla de estrategias para cada agente, dado que cada uno es independiente y puede tomar sus propias decisiones. Existen varios escenarios intermedios entre el mundo ideal y el mundo en el que todos los agentes siguen la estrategia básica; en esta sección se explorarán los resultados si:

1. Todos los agentes siguen la estrategia básica (el cual se tomará como escenario base)
2. Un agente, que no es el analizado, sigue la estrategia aprendida, y todos los demás (incluyendo al de interés), siguen la básica
3. Un agente, que es el analizado, sigue la estrategia aprendida, y todos los demás siguen la básica
4. Todos los agentes siguen la estrategia aprendida

Para este análisis, se creó un índice que se puede interpretar como la ganancia en porcentaje con respecto a la estrategia base, es decir, un valor del índice de 0,5 % significa que esa estrategia reportó 50 % más dinero al final de ese año para ese agente que si hubiera tomado la estrategia básica. Cualquier valor positivo del índice, entonces, implica un desempeño mejor que la línea base, un valor negativo significa un desempeño peor.

También es relevante explorar si la estrategia óptima es preferible en cualquier escenario respecto a los parámetros del mundo (inventarios iniciales, precios de compra/venta, costo de almacenamiento, entre otros). Para analizar este aspecto simultáneamente, en cada nueva iteración se reinicializan todos los valores de tales parámetros de manera casi aleatoria; con restricciones de valores no-negativos para todos, y adicionalmente para los márgenes de los agentes. Así, el análisis abarca una gran cantidad de casos y demuestra flexibilidad ante condiciones iniciales.

En la figura 7.10 se puede ver una comparación del desempeño de todos los agentes con respecto a la línea base elegida. Se muestra la línea base, o la estrategia *nadie óptima* como una línea azul punteada. Las dos curvas de comparación son: una combinación de estrategias en las que solamente el agente indicado toma y actualiza sus decisiones con el algoritmo de aprendizaje *un agente óptimo* y una combinación en la que todos los agentes aprenden con el algoritmo. Para esta figura se hicieron 250 realizaciones del algoritmo con 1000 etapas de aprendizaje, inicializando todos los parámetros del mundo de manera aleatoria cada vez, con restricciones mínimas relacionadas a los rangos de cada variable (por ejemplo, asegurar márgenes e inventarios no negativos)

En general, las distribuciones de índices relacionados a todos los agentes usando las estrategias óptimas (obtenidas del algoritmo de aprendizaje), se encuentran a la derecha de las distribuciones de un solo agente aprendiendo. Esto es el comportamiento esperado, pues uno de los principales castigos para todos los agentes es el costo por orden no cumplida, y si no aprenden que deben tener suficiente inventario para cubrir esas órdenes, siempre cargarán con ese tipo de costos. Queda claro que es preferible para todos los agentes en conjunto trabajar de manera óptima (con aprendizaje) al mismo tiempo, sin importar los parámetros iniciales de costos, inventarios o márgenes. Esta comparación puede consultarse de manera más puntual en el siguiente par de tablas, para la cual es importante recordar que el cero es la línea base, así que un comportamiento positivo significa un resultado por encima de ella.

Valor esperado de los índices de cada estrategia

Agente	Nadie	Solo otro	Solo él mismo	Todos
Menudeo	0	1.60	1.18	2.54
Mayoreo	0	-0.22	0.27	2.47
Almacén regional	0	-0.23	0.13	2.76
Fábrica	0	1.68	1.01	2.54

Desviación estándar de los índices de cada estrategia

Agente	Nadie	Solo otro	Solo él mismo	Todos
Menudeo	0	0.84	0.85	0.97
Mayoreo	0	1.43	1.38	1.01
Almacén regional	0	1.38	1.60	1.13
Fábrica	0	0.94	0.68	0.97

Es importante notar que las distribuciones de índices cuando solamente un agente utiliza la estrategia de aprendizaje se encuentran constantemente a la derecha de la línea base. Esto quiere decir que siempre es preferible utilizar la estrategia de aprendizaje, sin importar la estrategia que el resto de los agentes escojan.

Una situación muy interesante se presenta en el escenario complementario al anterior: se ha establecido que es preferible usar la estrategia aprendida a pesar de que los otros agentes no hagan lo mismo, pero ¿qué pasa con los agentes que no escogen la estrategia aprendida, cuando otro agente sí lo hace? El comportamiento esperado sería que la utilidad fuera mejor que si nadie estuviera aprendiendo, dado que al menos uno de los agentes estaría optimizando su inventario. Sin embargo, esto no es cierto para todos los agentes: tanto *mayoreo* como *almacén regional* tienen valor esperado negativo si solamente otro agente toma acciones inteligentes y cada uno de ellos sigue la estrategia base. Además, las desviaciones estándar asociada a estas distribuciones - y, en realidad, a las de todos los escenarios - son considerablemente mayores a las correspondientes a los otros dos agentes. Esto quiere decir que los agentes hacia el centro de la cadena tienen mayor presión de seguir estrategias inteligentes, independientemente de si los demás agentes en la cadena lo hacen. Este comportamiento se puede ver también en la figura 7.10.

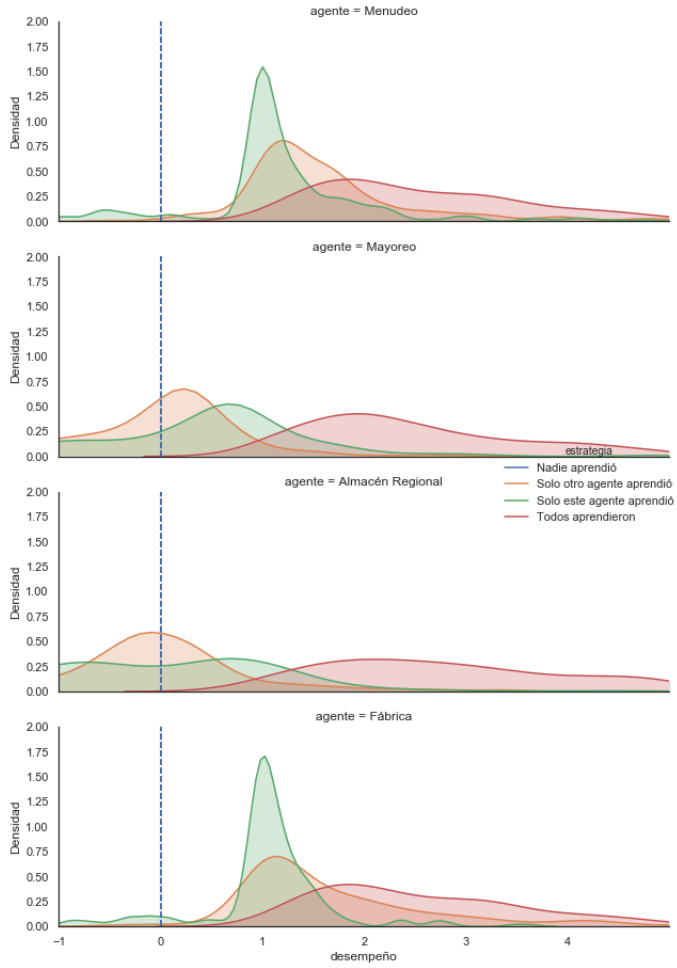
Por último, podemos observar que las distribuciones de un solo agente tomando decisiones óptimas son diferentes a pares: las de menudeo y fábrica son similares entre sí, al igual que las de mayoreo y almacén regional. Es notable que este comportamiento se presenta en los dos agentes que tienen una realidad diferente, ligada a agentes con estrategias estables y de donde provienen la demanda y oferta reales. El agente de menudeo siempre podrá vender todo su inventario lo más pronto posible,

dado que el comprador siempre tiene demanda positiva. Por otro lado, el agente de fábrica también carga con presión de los agentes inferiores de reducir sus costos de órdenes no cumplidas, así que también tiene asegurado vender su inventario pronto, mientras no compre cantidades innecesariamente altas. Los agentes de mayoreo y almacén regional no tienen ninguna de estas consideraciones.

Una posible hipótesis para este comportamiento se remonta al efecto látigo: aquellos agentes más cercanos a la información pueden reaccionar más rápidamente a cambios y tomar decisiones más egoístas que aquellos en medio de la cadena de suministro. Así, el agente menudeo tiene cercanía al consumidor y el agente fábrica tiene cercanía a los campos y conocen directamente ciertas partes de la información del sistema, lo cual les permite tener estrategias más sólidas y menos dependientes de lo que decidan los demás agentes.

En conclusión, para cualquier agente es preferible usar la estrategia aprendida, sin importar lo que hagan los demás agentes. Además, los agentes en los extremos de la cadena sufren menor presión por varianza en la información recibida directamente del consumidor y de los campos.

Figura 7.10: Comparación de desempeño entre combinaciones de estrategias



7.3. Q-Learning

Este método encuentra, para cada estado, la acción que acercará al agente lo más posible al óptimo. Idealmente, la política óptima es equivalente con cualquiera de ambos métodos. *Q-learning* permite más libertad respecto a empezar a jugar el juego a mitad de año, arreglar malas decisiones tomadas por los agentes en periodos anteriores gracias a su adaptabilidad ante cambios estocásticos, etc. Sin embargo, al tener que explorar estados a los cuales los agentes llegan después de haber tomado estrategias que no son óptimas, es un algoritmo mucho más pesado computacionalmente.

Las ecuaciones que describen el aprendizaje de un agente son:

1. La ecuación de utilidad (o recompensa) para cada agente a en el día d:

$$R(a,d) = r(a,d) + \gamma * r_{a,d+1} + ... + \gamma^{364} * r_{a,d+364}$$

Donde la recompensa del agente a en el día d es la ganancia relacionada a sus respectivas transacciones, como se definió al principio de este capítulo. Es importante notar la importancia de tomar un periodo de un año después del día d sin importar en qué día específico se encuentre el agente: de esta manera, el agente aprenderá si tiene que llegar al día 365 con inventario en almacenes.

El código que describe esta ecuación es:

```
lambdas = [(lambda_q_learning**n) for n in range(365)]
agent.q_function_reward_for_action[day-1] = agent.current_payout[day-1] -
↪  agent.current_payout[day-2]
rewards = agent.q_function_reward_for_action[day-1:] +
↪  agent.q_function_reward_for_action[0:day-1]
discounted_rewards = np.multiply(rewards, lambdas)
r_s_a = np.sum(discounted_rewards)
```

2. La función Q para cada agente a dado su estado, el día d con cierto inventario inv:

$$Q_a(inv_d, compra_d) = r_{a,(d,inv,compra)} + \gamma * \max_{compras} Q_a(inv_d + compra_d, compras_{d+1}$$

El código que describe la actualización de la función Q en cada paso es el siguiente. De manera similar que para *policy iteration*, se itera por agente. Para cada uno de ellos, primero se identifica cuál es el siguiente estado (día +1, inventario al final del día). Después se busca en la tabla que guarda todos los aprendizajes pasados; si se encuentra el escenario, se maximizan todas las posibles recompensas encontradas y se asigna el valor a Q con aquella que optimice; si no, se asigna el valor 0 a la función Q.

```
subset_s_prime_a_all = q_learning_df.loc[(q_learning_df['agent'] ==
↪  (agent.name)) & (q_learning_df['day'] == (next_day)) &
↪  (q_learning_df['inventory'] == (agent_inventory + agent_demand))]

if subset_s_prime_a_all.shape[0] > 0:
    max_q_s_prime_a_all = subset_s_prime_a_all['q_s_a'].max()
else:
    max_q_s_prime_a_all = 0

q_s_a = r_s_a + (lambda_q_learning * max_q_s_prime_a_all)
```

Para cada agente, su estado es un vector de longitud 2: en el día d es el inventario que tiene en el almacén, y las acciones que puede tomar en cada estado se representan como un vector de longitud 1: son las diferentes cantidades que puede comprarle al agente superior.

Además, similarmente a *policy iteration*, se implementó este algoritmo en la modalidad *greedy*, con ϵ empezando en 1,00 y terminando en 0,05.

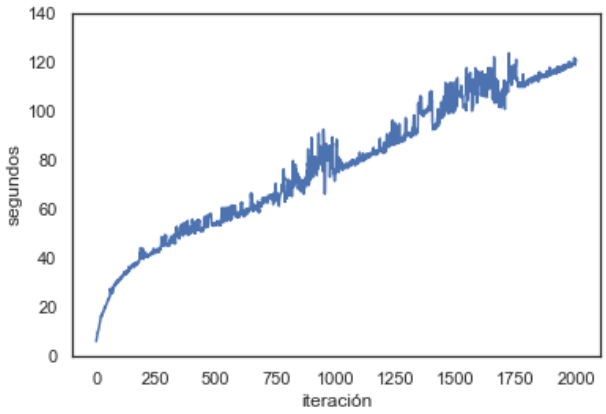
7.3.1. Tiempo de entrenamiento

El resultado de *Q-learning* es fundamentalmente diferente que el de *policy iteration*. El último solamente itera en cuatro vectores de longitud 365 (dado que hay 4 agentes, y 365 días en un año), manteniendo para cada agente solamente uno en la memoria de largo plazo: aquel que se ha determinado como el óptimo histórico. Por otro lado, el primero explora estados del mundo, incluso si no provienen de acciones "buenas" y mantiene en la memoria de largo plazo la mejor acción a tomar si se llega a ese estado, además del valor de la función Q para cada una de las posibles acciones exploradas. Esto significa que en cada iteración, el conocimiento de cada agente aumenta, pues recuerda la mejor acción a tomar para cada estado, el cual es una combinación del día del año y el inventario en tal día.

A diferencia de las iteraciones de *policy iteration*, las iteraciones de *Q-learning* toman más

memoria, y por lo tanto más tiempo, a medida que el modelo va entrenando. En la figura 7.11 puede observarse este comportamiento, el cual se ajusta a un modelo polinomial $y = 3{,}59x^{0{,}452}$ con una $R^2 = 0{,}975$. Esto quiere decir que crece más lento que un modelo lineal, sin embargo la tasa es suficientemente grande como para convertirlo en un modelo prohibitivo desde el punto de vista de tiempo necesario para entrenamiento.

Figura 7.11: Tiempo (segundos) por iteración en Q-learning



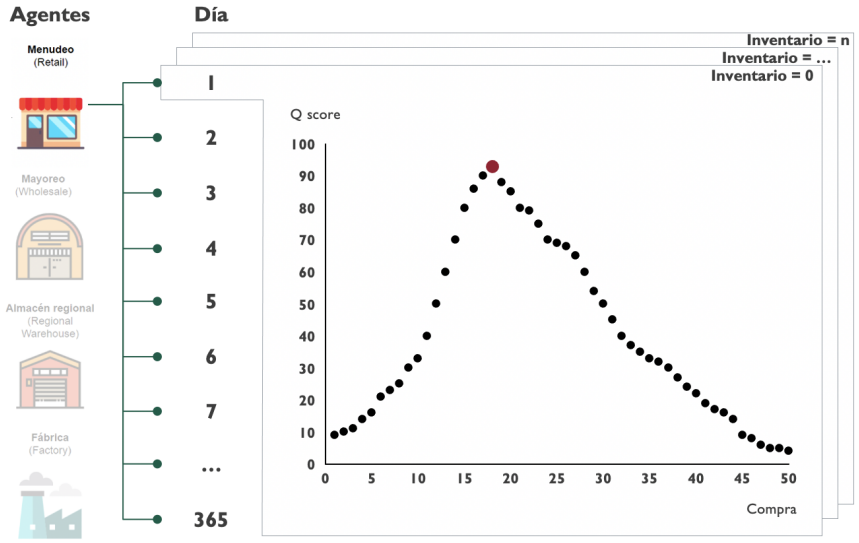
Al notar el comportamiento creciente del tiempo por iteración, se hicieron varias pruebas para documentar el tiempo necesario para correr el algoritmo de aprendizaje. En secciones anteriores se mostró que $10,000$ iteraciones eran suficientes para que *policy iteration* encontrara *policies* óptimas en menos de 10 minutos; sin embargo, para obtener $10,000$ iteraciones en *Q-learning*, parecen necesarias más de 3 semanas de entrenamiento continuo. Además, dado que por definición, este último explora y aprende a reaccionar a estados aún si no son óptimos, es probable que $10,000$ iteraciones no sean suficientes para tener un resultado utilizable.

Iteraciones	Tiempo
1,000	15 horas
2,000	47 horas
10,000	25 días (est)

7.3.2. Estrategias

Como se ha descrito anteriormente, los resultados de *Q-learning* tienen una forma muy extensa: para cada agente, existe un conjunto de posibles acciones en cada estado (combinación de día e inventario) y sus respectivos valores de la *función Q*. La manera de obtener la acción óptima para cada agente en cada tiempo t es por medio de la búsqueda de la acción a que maximiza la *función* Q dado el estado s, es decir dado el inventario. En la figura 7.12 se puede visualizar esta estructura.

Figura 7.12: Estructura de resultados de Q-learning



En sentido estricto, el resultado completo de un experimento con ciertos metaparámetros del modelo (e.g. total de iteraciones) y parámetros del mundo (e.g. inventarios iniciales o márgenes de ganancia) es una sola tabla con 6 columnas: agente, día, inventario, compra, valor de la función R y valor de la función Q. La columna de compra representa la acción a tomar por el agente en el estado - la combinación de día e inventario.

El modelo creado encuentra soluciones, pero al ser tan grande el espacio a explorar y tardar tanto las iteraciones, tales soluciones encontradas no convergen a óptimos en un tiempo razonable.

Lamentablemente, se concluye que *Q-learning* no es una opción viable, pues el poder de cómputo

y el tiempo necesarios para obtener resultados son excesivos. El método de *policy iteration* es considerablemente más flexible y más rápido, así que puede reentrenarse en caso de cualquier cambio en los datos anteriores de demanda del consumidor u oferta de los campos.

Capítulo 8

Conclusiones

Vale la pena revisitar la hipótesis con la que se comenzó este trabajo:

'Bajo este nuevo conjunto de supuestos, es posible encontrar estrategias óptimas para todos los agentes tomadores de decisiones en el Juego de la Distribución de Cerveza, por medio de algoritmos de aprendizaje reforzado, que produzcan resultados en un tiempo suficientemente veloz como para poder accionarlas.'

En primer lugar, se confirmó que, para cualquier agente, la estrategia aprendida con *policy iteration* es preferible a estrategias de sentido común, como aquella en la que se considera un promedio móvil de días recientes; sin embargo, ningún agente está obligado a seguir estas estrategias. Al analizar distintos escenarios con diferente adopción de estrategias inteligentes, se encontró que, para cualquier agente, en promedio es preferible tomar la estrategia inteligente, sin importar lo que los demás agentes hagan.

También se observó que las políticas (*policies*) óptimas son encontradas por el algoritmo de manera relativamente rápida, necesitando solamente $4-5$ minutos para completar $10,000$ iteraciones, las cuales son suficientes para llegar a estrategias estables que no cambian considerablemente al aumentar a $100,000$ o incluso $1,000,000$ de iteraciones. Esto dota al algoritmo de flexibilidad

para ajustarse a cambios en la demanda del consumidor o la oferta de los campos, pues puede reentrenarse diario o incluso más de una vez al día.

Adicionalmente, se encontró un comportamiento de efecto látigo doble: el primero por parte del consumidor y el segundo por parte de los campos. Esto quiere decir que aquellos agentes que estén más al centro de una cadena, es decir, más lejos de ambas fuentes de información simultáneamente, recibirán más varianza en la información total. Por lo tanto, mientras que en promedio todos los agentes tienen un mejor resultado si al menos uno de ellos toma la estrategia inteligente, aquellos en el centro de la cadena deberían ser los más interesados en asegurar que todos toman decisiones óptimas, e incluso en invertir en modelos más poderosos.

Por último, se concluyó que la implementación de *Q-learning* construida no es viable para una situación del mundo real, dado el tiempo necesario por iteración y la tendencia creciente de este. Tal comportamiento se debe a que cada agente debe conservar conocimiento de todos los estados y valores de la *función Q* para todas la acciones que ha probado. Al llegar a $2,000$ iteraciones, el tiempo supera los dos minutos para cada una de ellas. En contraste, el tiempo total para encontrar estrategias óptimas con *policy iteration*, para $10,000$ iteraciones, es de $4-5$ minutos.

8.1. Trabajo futuro

Se identificaron algunas oportunidades notables de mejora para este modelo, pero que implican un trabajo considerable y podrían ser temas para ahondar en un futuro:

- Comparación de desempeño de soluciones existentes con otros métodos (dinámica de sistemas, algoritmos genéticos) incluyendo la restricción de estacionalidad en la oferta
- Relajación del supuesto de agentes representativos a individuales: por ejemplo, en vez de existir una sola tienda minorista, permitir que existan un número n de ellas cuya penalización por órdenes no cumplidas no sea fija, sino la pérdida de un consumidor que se va a otra tienda
- Planteamiento diferente del algoritmo aprendizaje con *Q-learning* de tal manera que el obstáculo de tiempo de entrenamiento no sea un bloqueo

Bibliografía

[1] D. Bloembergen y D. Hennes. Fundamentals of multi-agent reinforcement learning. *Tutorial 2: MARL (multi-agent reinforcement learning*, 2013.

[2] L. Busoniu, R. Babuska, y B. De Schutter. Multi-agent reinforcement learning: an overview. *Studies in Computational Intelligence*, 310:183–221, 2010.

[3] S. K. Chaharsooghi, J. Heydari, y S. H. Zegordi. A reinforcement learning model for supply chain ordering management: An application to the beer game. *Decision Support Systems*, 45:949–959, 2008.

[4] Bryan Conneely, Jim Duggan, y Gerard Lyons. A distributed system dynamics-based framework for modelling virtual organisations. 2019.

[5] Peter Dizikes. The secrets of the system. *MIT News*, 2012.

[6] Emilio Frazzoli. *16.410 / 16.413 Principles of Autonomy and Decision Making. Fall 2010.* Massachusetts Institute of Technology: MIT OpenCouseWare, https://ocw.mit.edu/. License: Creative Commons BY-NC-SA.

[7] Oliver Grasl. Understanding the beer game using system thinking to improve game results. 2015.

[8] Junling Hu. Reinforcement learning explained. 2016.

[9] F. Robert Jacobs y Richard B. Chase. *Administración de operaciones. Producción y cadena de suministros.* McGraw Hill-Education, 2011.

[10] S. Kimbrough, D.J. Wu, y F. Zhong. Computers play the beer game: can artificial agents manage supply chains? *Decision Support Systems*, 33:323–333, 2002.

[11] United States Department of Agriculture. Field crops usual planting and harvesting dates. *Agricultural Handbook*, 628:6–7, 2010.

[12] Lydia Saad. Beer is americans adult beverage of choice this year. 2014.

[13] A.B. Shiflet y G.W. Shiflet. *Introduction to Computational Science*. Princeton University Press, 2006.

[14] John D. Sterman. Modeling managerial behavior: misperceptions of feedback in a dynamic decision making experiment. *Management Science*, 35:321–339, 1989.

[15] John D. Sterman. *Business Dynamics*. Irwin/McGraw-Hill, 2000.

[16] F. Strozzi, J. Bosch, y J.M. Zaldivar. Beer game order policy optimization under changing customer demand. *Decision Support Systems*, 42:2153–2163, 2007.

[17] R Sutton y A. Barto. *Reinforcement learning: an introduction*. The MIT Press, 1998.

[18] Mohammad Zarandi, Mohammad Hassan Anssari, Milad Avazbeigi, y Ali Mohaghar. A multi-agent model for supply chain ordering management: An application to the beer game. *Supply Chain Management*, págs. 433–442, 2011.