UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

REPOSICIÓN AUTOMÁTICA EN EMPRESA DE RETAIL MEDIANTE ALGORITMOS DE OPTIMIZACIÓN

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

ERICK FELIPE SALOMÓN PÉREZ FLORES

PROFESOR GUÍA:
DAVID VALENZUELA URRUTIA

PROFESOR CO-GUÍA:
FRANCISCO RIVERA S.

MIEMBROS DE LA COMISIÓN:
DANIEL POLA C.

SANTIAGO DE CHILE
2022

Resumen

El presente Trabajo de Título corresponde a la memoria realizada en una compañía de *retail*, realizando modelamiento matemático y optimización. Para las compañías de *retail* es clave reponer productos en el momento y lugar preciso para tener un inventario equilibrado según los estándares de la compañía y, particularmente, no tener problemas de *stock*. Como problemas de *stock* se destacan los quiebre de *stock* (falta de productos en tiendas) y *sobre-stock* (exceso de productos en tiendas), este último puede generar que productos entren al proceso de liquidación, reduciendo el margen de utilidad. Generar un calendario de reposición eficiente es una tarea compleja debido a la gran cantidad de SKUs (*Stock Keeping Units*) y tiendas que manejan las grandes compañías, por lo que la planificación se genera mediante *softwares* especializados o incluso planificaciones manuales rudimentarias. Muchos de estos *softwares* no son capaces de satisfacer todas las necesidades particulares de cada compañía, por lo que una buena dirección de desarrollo de los *retailers* es internalizar estos procesos y generar herramientas computacionales propias.

Por lo anterior, el trabajo se enfoca en desarrollar un modelo simplificado para obtener un calendario de reposición, y herramientas computacionales que permitan poner a prueba este modelo. El modelo se desarrolla como un problema de optimización lineal de enteros, donde la función objetivo contempla un término de contribución de utilidades, tratando de maximizarlas. Adicionalmente, se agregan 3 términos a la función objetivo con el fin de moldear la solución encontrada de reposición, considerando reposiciones una vez por semana. Se simula el comportamiento del modelo mediante *scripts* programados en lenguaje Python, que permiten realizar la optimización con los *solvers* lineales existentes PuLP CBC y Gurobi. Se prueba con parámetros simulados, donde el *forecast* de la demanda es el *input* más destacable.

Como resultados más importantes se tiene que la implementación de varios términos en la función objetivo, junto con las restricciones utilizadas, influencian la optimización moldeando la solución encontrada. Lo anterior se ve como una buena herramienta para personalizar la planificación de la reposición, dirigiendo el *output* del modelo a distribuciones de *stock* acordes a la realidad de la compañía. La implementación por ventanas permite anticiparse a *peaks* de demanda, reduciendo el número de quiebres de *stock*. Se simula y evalúa el error que puede presentar el *forecast* de demanda, obteniendo perdidas de utilidades del orden del 1.8 % para un MAPE del 16 % y la disminución de utilidades es de un 6.1 % para un MAPE del 40 %.

Se obtiene un modelo robusto que entrega reposiciones con características deseables para la compañía de *retail*. Las herramientas computacionales utilizadas permiten realizar optimizaciones, y se presentan de buena manera los resultados computacionales. El trabajo realizado

permite avanzar en la dirección correcta para desarrollar herramientas propias que permitan obtener un calendario de reposición utilizando *solvers* lineales.

A mi lalita, lo logramos.

Agradecimientos

Agradezco a mis padre. No todo esfuerzo da frutos pero el de ustedes si los dio. Gracias a ambos por siempre ponerle el hombro y siempre promover una buena educación para mi.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Problema

La reposición de *stock* de productos en tiendas corresponde a una actividad necesaria, y clave, para cualquier compañía minorista (*retailer*). Esta consiste en generar una planificación que permite entregar los productos correctos, en el momento correcto, en la cantidad óptima [37], en función de las particularidades de operación y comportamiento de cada tienda del *retailer*.

Una planificación de reposición adecuada, tiene beneficios asociados a la buena rotación de los productos, permitiendo generar un mayor margen de utilidades [33]. Por el contrario, una planificación inadecuada puede causar problemas de *sobre-stock* o quiebres de *stock* de algunos SKUs (*Stock Keeping Units*) [36]. El *sobre-stock* ocurre cuando la cantidad de productos en tiendas supera en exceso los índices de demanda, lo que provoca un desajuste en los niveles deseados de *stock* [32]. Tener *sobre-stock* en una tienda es un gran problema de planificación, debido a que se pierde la oportunidad de vender productos en un menor lapso de tiempo en otras tiendas, evitar liquidaciones posteriores y liberar capacidad en tiendas (30-40 % de ventas estancadas) [27][45]. Otro problema de *stock* son los quiebres que se producen cuando consumidores no encuentran el producto en la talla, color y/o marca en la tienda que desean. Entre las causas que generan quiebres de *stock* se tienen, con más de un 40 %, los problemas para prever la demanda y para generar los pedidos precisos para cubrir dicha demanda [40], que es donde se enmarca este trabajo. Como respuesta, los consumidores no realizan la compra en más de un 50 % de las veces que se enfrentan a situaciones de quiebres de *stock* [16][34][40]. Dicho esto, tener problemas de *stock* provoca una insatisfacción en los consumidores, efectos negativos en la imagen y lealtad a la tienda generando riesgo de pérdida de consumidores[38]. Lo anterior se traduce en pérdida de utilidades para compañías de *retail*, como ejemplo, se estima que la Industria Europea de Retail tiene pérdidas que alcanzan los 400 mil millones de euros cada año por quiebres de *stock* [16].

La logística que alimenta a las tiendas con productos tiene sólo una dirección: desde los centros de distribución (CDs) hacia las tiendas. En contadas excepciones se hacen transferencias entre tiendas o de forma inversa a los CDs, porque es un proceso complejo y caro.

Es por esto, que una vez que se define a qué tiendas se envían los productos, estos no tienen otra opción que ser vendidos en esa tienda, al precio que sea. Si se envían demasiados productos a una tienda específica, y estos productos no son vendidos en el tiempo adecuado, no quedará otra opción que reducir su precio, afectando directamente al margen obtenido para la compañía. Como un ejemplo extremo, se puede mencionar el caso de la reposición de parkas a una tienda en Punta Arenas (extremo sur) comparado a una tienda en La Serena (centro norte) para la misma época del año. Teniendo un stock centralizado, es razonable considerar una mayor cantidad de parkas a reponer en Punta Arenas que en La Serena, para asegurar que estas se logren vender en un tiempo acotado y a precio completo, es decir, con el mayor margen posible. El caso ideal para los *retailers* es comprar una cantidad de productos adecuados, distribuirlos correctamente en las tiendas, y asegurar su venta en la temporada que corresponde, evitando llegar a situaciones de liquidaciones masivas.

Una forma de tratar con este problema es utilizar la tecnología, que permite realizar tareas de alto computo en muy poco tiempo y de manera precisa. Utilizar modelos matemáticos para la planificación de reposición de los distintos SKU a cada una de las tiendas permite atacar el problema de manera centralizada y tomando decisiones basadas en datos, evitando que algunas tiendas generen problemas de stock en otras. Al generar una reposición automática disminuyen las responsabilidades de los empleados, pasando de planificadores a supervisores, disminuyendo también las horas-persona necesarias para hacer la planificación. Por último, se destaca que este nuevo enfoque centralizado ya no se limita a las peticiones de cada tienda de forma individual, sino que se ve como un conjunto de estas para generar un mayor beneficio global para la empresa.

Se define el problema de manera matemática como una optimización, donde la función objetivo está relacionada con el beneficio económico de la empresa y las restricciones tienen relación con las capacidades de almacenamiento y transporte de productos (restricciones logísticas), además de restricciones lógicas. Como resumen de manera cualitativa:

$$\begin{aligned} \text{Max.} \quad & \text{Beneficio económico para la empresa} \\ \text{s.a.} \quad & \text{Restricciones logísticas} \\ & \text{Restricciones lógicas} \end{aligned}$$

Definir la forma en que se modelan los beneficios es un desafío importante dentro del problema, ya que no sólo depende de variables externas a la empresa, sino que también depende de la política empresarial y de cómo el directorio establece las metas a largo plazo, por lo tanto, también la formulación de los beneficios es discutible y variable en el tiempo. Usualmente, para definir los beneficios se toman en consideración factores como predicciones de ventas, caracterización de artículos similares, sumado a un modelamiento de las preferencias de los compradores [12][15]. También, se consideran los costos asociados a cada producto y el costo por tener por tiempos prolongados los productos almacenados en centro de distribución y tiendas.

En el contexto de la transformación digital o de la revolución industrial 4.0, las empresas están buscando mecanismos para acelerar los tiempos de respuesta y reducir los errores ocasionados por la manipulación de estos grandes volúmenes de datos. La automatización de

tareas repetitivas y el apoyo en los datos para la toma correcta de decisiones es una tendencia que llegó para quedarse y que se está potenciando en cada una de las áreas de las empresas. Es así como han surgido gran cantidad compañías que entregan servicios dirigidos a *retailers* para realizar una planificación de la reposición de manera automática, considerando ventas de los SKUs en las semanas anteriores, para ir corrigiendo semana a semana.

Esta memoria se enmarca en el modelado del problema de optimización, orientando la reposición a tiendas como un problema lineal de enteros (ILP). Se busca poner a prueba el modelo matemático desarrollado realizando simulaciones de escenarios para una empresa de *retail* utilizando solvers lineales, como los que ofrecen Gurobi y PuLP, para resolver el problema de optimización. Esto corresponde a un primer paso en generar una herramienta computacional más personalizada para obtener un calendario de reposición. Se presenta como una alternativa de desarrollo para que la empresa de *retail* en cuestión, al desarrollar tecnologías propias de este tipo se pueden remplazar los servicios contratados a compañías que ofrecen software de planificación de inventario.

1.2. Situación actual

En esta sección se presenta el panorama actual de la empresa en esta materia. Hoy en día, la empresa de *retail* en la cual se enmarca este trabajo cuenta con empleados encargados de la planificación de la reposición de productos (*planners*). Estos trabajan de manera centralizada para generar reposiciones a cada una de las, aproximadamente, 50 tiendas distribuidas a lo largo de Chile. Se utiliza un software que apoya en la planificación de la reposición. Este software, genera una reposición en base a las ventas del SKU en la semana anterior, pero no utiliza un *forecast* de demanda para las semanas posteriores, por lo que no puede surtir, por si solo, los productos de manera anticipada. Esto genera problemas en semanas donde la demanda de productos en las tiendas sube considerablemente, por ejemplo, en navidad, donde los *planners* deben solicitar con meses de anticipación una mayor cantidad de productos para almacenar en tiendas (empujes manuales) para tener los productos a tiempo y evitar quiebres de *stock*.

Se hace necesario, dado el contexto descrito anteriormente, un software que libere la carga de trabajo que por ahora la realizan personas. En consecuencia, es necesario tener herramientas computacionales que permitan generar un calendario de reposición sin depender de la pericia de *planners* experimentados. Para una persona es una tarea compleja encontrar una solución eficiente (calendario de reposición) cuando se consideran muchos SKUs y tiendas. Un algoritmo puede acercarse más a una solución eficiente utilizando gran cantidad de datos.

Actualmente la responsabilidad de generar un calendario de reposición recáe en los *planners*, por lo que la generación de una buena planificación depende del experiencia y habilidad de cada encargado. No se tiene una metodología generalizada y centralizada que considere todas las variables descritas anteriormente, por lo que el proceso actual no es completamente automático.

1.3. Objetivos

1.3.1. Objetivo general

Construir un modelo y una metodología que, mediante herramientas computacionales, sea capaz de entregar las cantidades de artículos a reponer, en cada una de las tiendas del *retail* seleccionado, de manera centralizada y con un horizonte de semanas determinadas por el *forecast* de la demanda.

1.3.2. Objetivos específicos

- Definir parámetros y datos a utilizar.
- Modelar función objetivo, definiendo claramente el beneficio.
- Definir costos, modelando correctamente el costo de oportunidad de tener indefinidamente un producto en bodega y en tienda.
- Definir restricciones del problema.
- Desarrollar herramientas, que permitan implementar *solvers* lineales existentes para resolver un problema de optimización lineal de enteros (ILP).
- Definir métricas de evaluación del rendimiento de cada algoritmo de optimización.
- Implementar herramientas que permitan analizar los resultados de optimización.
- Evaluar la validez de las soluciones encontradas, dada la realidad de la empresa.

1.4. Alcances

En primer lugar, se señala que este trabajo, por si solo, no busca generar una herramienta productiva para la empresa. Se busca desarrollar un primer modelo que permita generar un calendario de reposición, considerando factores logísticos de la empresa. El trabajo busca también realizar un estudio de las componentes del problema de optimización lineal, tales como función objetivo y restricciones, para que sea flexible y se adapte a las necesidades de la empresa.

Se contempla también el desarrollo computacional de un *script* en el cual se escriba el modelo matemático de optimización desarrollado, y que permita que *solvers* lineales existentes puedan realizar la optimización de la reposición de productos. Busca también realizar simulaciones de distintos escenarios donde opera la optimización. Lo anterior sin datos reales, los parámetros en las simulaciones son fijados en este trabajo. No se desarrolla un *forecast* de la demanda de los productos, se considera un *input* para el modelo.

El trabajo concluye con la implementación en Python de visualizaciones y métricas que permitan obtener nociones del desempeño del modelo desarrollado, y de las simulaciones realizadas.

1.5. Estructura del trabajo

El trabajo contiene seis capítulos que se comentan a continuación:

- **Capítulo 1 - Introducción.** Se presenta la problemática a abordar y su motivación, junto con los objetivos y alcances del trabajo.
- **Capítulo 2 - Marco teórico y estado del arte** Se presentan las bases teóricas del trabajo, destacando los problemas de optimización lineales de enteros y los algoritmos existentes para resolverlos. En el marco teórico se hace una revisión bibliográfica de problemas de optimización para la reposición en el *retail*, destacando modelos utilizados y métodos de resolución.
- **Capítulo 3 - Metodología** Se detallan herramienta computacionales a utilizar, también se explicitan nociones que son útiles a lo largo del trabajo, terminando con la descripción de las actividades presentes en el Capítulo 4 (Desarrollo y resultados).
- **Capítulo 4 - Desarrollo y resultados.** Se presenta el modelo desarrollado y se detallan las simulaciones realizadas. Se presentan en este Capítulo los resultados de las simulaciones, en forma de figuras y tablas. Se describen los resultados.
- **Capítulo 5 - Análisis y discusión** Se profundiza en el análisis de los resultados, y se discuten las consecuencias de estos.
- **Capítulo 6 - Conclusiones.** Explica las conclusiones más importantes del trabajo.

Capítulo 2

Marco teórico y estado del arte

2.1. Programación Lineal

La programación lineal es una de las grandes historias de éxito en la programación. Desde su formulación en la década de 1930 y el desarrollo del algoritmo simplex a mediados de 1940, generaciones de trabajadores en economía, finanzas, e ingeniería han sido entrenados para formular y resolver problemas de programación lineal (LP). Incluso cuando las situaciones modeladas son no lineales, se busca realizar una formulación lineal porque permiten la utilización de un software más sofisticado, los algoritmos garantizan la convergencia, y porque incertezas en los modelos y datos comúnmente hacen impracticable construir un modelo no lineal más elaborado [49].

Las propiedades fundamentales de un problema de programación lineal son:

1. Un vector de variables, el cual es obtenido al resolver el problema de optimización.
2. Una función objetivo lineal.
3. Restricciones lineales, tanto igualdades como desigualdades.

Se puede escribir de manera genérica de la siguiente forma:

$$\begin{aligned} \text{Encontrar vector} \quad & \mathbf{x} \\ \text{que maximiza} \quad & \mathbf{c}^\intercal \mathbf{x} \\ \text{sujeto a} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned}$$

donde $\mathbf{c}$ y $\mathbf{x}$ son vectores en $\mathbb{R}^n$, $\mathbf{b}$ es un vector en $\mathbb{R}^m$, $\mathbf{A}$ en una matriz de $m \times n$. Al conjunto de todos los puntos que safisfacen $\mathbf{A}\mathbf{x} \leq \mathbf{b}$, y $\mathbf{x} \geq 0$, se le conoce como la región factible, donde se busca la solución [4][48].

2.1.1. Programación Lineal Entera

La programación lineal entera (ILP) se enfoca en encontrar la combinación de variables enteras que presentan el mejor valor de la función objetivo de un problema lineal. Para encontrar este vector de enteros que optimiza la función objetivo primero se realiza una optimización considerando variables de decisión continuas, para luego, mediante otros algoritmos, encontrar la solución entera [18][48].

El algoritmo más conocido utilizado para este propósito es el *Branch and Cut* [14][28], que combina el algoritmo *Branch and Bound*, basado en la poda de regiones factibles sub-óptimas y el algoritmo *Cutting Planes* asociado a la relajación de restricciones de LP.

2.1.2. Algoritmo simplex

Simplex es el algoritmo más conocido, y más utilizado, para la resolución de problemas de programación lineal. Desarrollado en sus inicios por George Dantzig en 1947, el algoritmo simplex se basa en identificar un vértice de la región factible e iterar repetidamente avanzando de vértice en vértice hacia el óptimo. Cuando el algoritmo no detecta una dirección donde se encuentre una mejora de la función objetivo, la optimización acaba y de asume que se encuentra el óptimo de la función. Corresponde a un método exacto de solución para problemas de programación lineal [11][43].

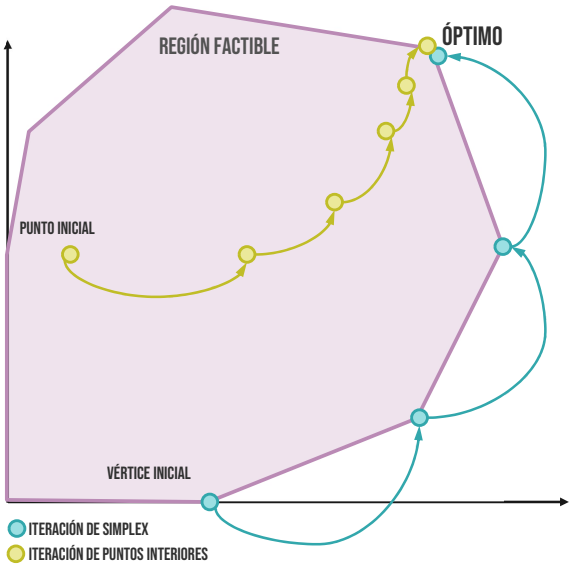

Figura 2.1: Ilustración cualitativa de iteraciones con algoritmo simplex y método de puntos interiores.

2.1.3. Método de puntos interiores

La era moderna del método de puntos interiores data 1984 cuando Karmarkar propone su algoritmo para problemas de programación lineal. El algoritmo es iterativo y tiene una complejidad distinta dependiendo de la implementación y el problema tratado, siendo en algunos casos competitivo con el algoritmo simplex [35][43]. A diferencia del algoritmo simplex, como el nombre lo señala, el algoritmo de puntos interiores se acerca en cada iteración a la solución exacta por el interior de la región factible del problema. Este algoritmo presenta ventajas sobre simplex en las situaciones en que simplex se estanca y no converge al óptimo por efectos numéricos de tratar con bordes de manera computacional [44][49].

2.1.4. Branch and Bound

Corresponde a un algoritmo implementado para encontrar la solución entera de un problema de optimización. Mediante iteraciones, se divide el espacio factible, realizando ramificaciones de las soluciones (*Branch*) y luego se acota (*Bound*) la búsqueda sólo a las ramas que aún pueden presentar el mejor valor de la función objetivo. Para ramas en las cuales ya se alcanza el óptimo, y este no supera el óptimo, con variables enteras, de otras ramas, la rama sub-óptima es podada [5].

2.1.5. Planos cortantes

Introduciendo relajaciones en el problema de optimización, el método de planos cortantes añade las restricciones del problema lineal paso a paso, buscando variables enteras que estén lo más cerca posible del vértice donde se encuentra el óptimo del caso continuo. Con esto, cuando un plano cortante intersecta variables enteras, se encuentra el óptimo entero. Este método es conocido por reducir enormemente el la capacidad de procesamiento necesaria para llegar a la solución entera [18].

2.1.6. Formulación Big-M

La formulación Big-M es un método diseñado para resolver problemas lineales que tengan restricciones de la forma "mayor o igual". Se recuerda que para implementar el algoritmo simplex se requiere de una zona factible que esté acotada, para que el algoritmo converja, ya que se mueve por los bordes de la zona. Sin embargo, este no es la única utilidad que presenta, ya que permite implementar de manera simple las llamadas *indicator constrants* [20] utilizando variables vinarias en la implementación. Esto es especialmente útil para modelar variables que discontinuas. Entre sus desventajas se encuentra el hecho de que se debe definir manualmente el valor de M que se utiliza. Este valor debe escogerse con cuidado ya que tiene que ser lo suficientemente grande para que la implementación sea correcta [7].

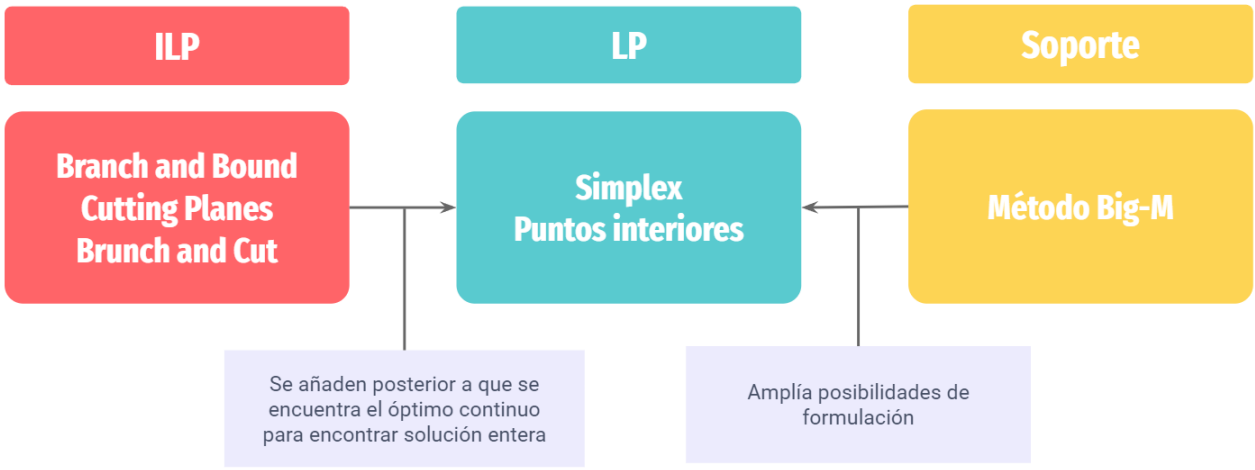

Figura 2.2: Esquema de métodos en problemas lineales.

2.2. Estado del arte: optimización en el *retail*

La optimización en el *retil* ha sido tratada con anterioridad en la bibliografía, lo que ayuda para tener referencias de modelamiento matemático del problema, aportando en gran medida a definir la función objetivo y las restricciones a utilizar [1][3][26][29]. En la mayoría de los casos se trata como un problema de optimización lineal de enteros (ILP), en donde se utilizan *solvers* lineales [12][13][15][42] y algoritmos heurísticos por igual [2][17][24][25][37]. En el trabajo de Kun Li y Huixin Tian [25] se utiliza la metaheurística PSO (Particle Swarm Optimization) [6], la cual es una buena herramienta para tener soluciones rápidas y aproximadas de LP. Cuando el problema de optimización contempla más variables, se tienen estudios de modificaciones del algoritmo PSO para no tener problemas de llegar a soluciones cercanas al óptimo [39][51].

A pesar de tener buenas referencias, se presenta como desafío poder ajustar lo que ese ha hecho en trabajos anteriores a la realidad de la empresa y los recursos con los que se cuenta (humanos, computacionales y calidad de los datos).

2.2.1. *Solvers* lineales

Son variados los *solvers* lineales disponibles para utilizar en problemas lineales de optimización. Estos corresponden a potentes herramientas para resolver gran cantidad de problemas, desarrollados para ser utilizados en una gran variedad de lenguajes de programación.

Varían en rendimiento y costo económico. Un ejemplo de *solver* lineal gratuito es PuLP CBC, incluido por defecto en la biblioteca `PuLP` [10][46], que se puede importar en Python. PuLP es un modelador de problemas lineales escrito en Python desarrollado por COIN-OR (Computational Infrastructure for Operations Research). PuLP puede generar archivos MPS o LP y llamar a los *solvers* GLPK, COIN-OR CLP/CBC, CPLEX, GUROBI, MOSEK, XPRESS, CHOCO, MIPCL, SCIP para resolver problemas lineales [9], lo que incluye soporte

para problemas de maximización y minimización con restricciones. Para utilizar un *solver* no gratuito mediante PuLP se debe tener activa la licencia correspondiente. Otro ejemplo de un solver gratuito, de código abierto, es COIN-OR CBC, siglas que provienen del inglés *Open-source Software for the Operations Community Branch and Cut* [8].

Figura 2.3: Logos de Gurobi [19] y COIN-OR [9].

Cplex (de IBM) [21] y Gurobi [19] corresponde a los *solvers* que más destacan, los cuales tienen asociados un costo en dinero. Sin embargo, ambos entregan licencias académicas gratuitas, para fines no comerciales. Gurobi se destaca por su buen rendimiento, debido a que maneja de mejor manera el álgebra lineal necesario para la solución de problemas, combina tanto simplex como puntos interiores, y evalúa ramas en paralelo, para encontrar el óptimo.

Capítulo 3

Metodología

Esta sección presenta la metodología que se lleva a cabo en el desarrollo del trabajo. Cada uno de los pasos definidos está estrechamente relacionado con los objetivos particulares del trabajo, y en consecuencia, con el objetivo general. A continuación se hace mención de las herramientas computacionales que se utilizan a lo largo del trabajo. Luego se muestra un punteo de cada una de las actividades consideradas para el desarrollo. Finalmente en la Sección 3.3 se presenta la nomenclatura que se utiliza en el siguiente Capítulo, tanto en el desarrollo de modelos como en la muestra de los resultados de la implementación de los mismos.

3.1. Herramientas computacionales

Python es un lenguaje de programación interpretado y de alto nivel. Se caracteriza por ser un lenguaje donde prima la legibilidad de código y de fácil aprendizaje. Es un lenguaje de programación de código abierto que cuenta con bibliotecas que permiten desde manipular valores hasta utilizar algoritmos de inteligencia artificial. En este trabajo se utiliza `Python 3.8.8` instalado mediante Anaconda 3, que es un *toolkit* de ciencia de datos, el cuál instala muchas de las bibliotecas más comunes utilizadas en esta disciplina. Para la manipulación de variables y arreglos se utiliza `Numpy 1.20.1`. Para el cálculo de tiempo de ejecución de código se utiliza la biblioteca `time`. Para la realización de visualizaciones y gráficos se utiliza `matplotlib 3.3.4`.

Como se menciona en secciones anteriores, se utilizan *solvers* lineales en la implementación para realizar la optimización de la reposición. Se utiliza la biblioteca `PuLP 2.5.1` para utilizar el *solver* PuLP CBC. El *solver* Gurobi, famoso por su buen rendimiento, se utiliza importando la biblioteca `gurobipy` utilizando una licencia académica.

Las simulaciones son ejecutadas en un computador con procesador Ryzen 5 4650 PRO con gráficos integrados vega 7, 16GB RAM y sistema operativo Windows 10 (llámese de ahora en adelante R5). Para gran parte de las simulaciones se utilizan valores aleatorios entre valores máximos y mínimos fijados con anterioridad (Sección 4.2.1). Estos valores se

obtienen al utilizar `random.randint()` de la biblioteca `Numpy` [30]. Para obtener siempre la misma muestra aleatoria, y en consecuencia los mismos resultados, se utiliza la semilla aleatoria `random.seed(32)` también de `Numpy` [31].

3.2. Actividades

La metodología se divide en 3 partes que se pueden nombrar como: modelamiento, implementación y evaluación. A continuación se describe el enfoque que tiene cada uno y sus principales actividades. El detalle de las actividades se encuentra de manera más amplia en la Sección 4.

Modelamiento contempla la definición del modelo de optimización. Se define que es un modelo lineal y por ende se busca una función objetivo lineal y restricciones lineales. Es importante también definir correctamente las variables y parámetros del problema, para obtener la solución deseada. Una mala elección de restricciones, o variables de decisión, puede llevar a un implementación ineficiente, significando un mayor nivel de costo computacional, lo cual genera mayores tiempos de ejecución.

Implementación hace alusión a los algoritmos a utilizar para resolver, de manera computacional, el problema modelado. En este caso el modelo es implementado en Python, introduciendo correctamente todos los elementos del problema de optimización para que *solvers* lineales ya implementados (PuLP CBC y Gurobi) resuelvan el problema de manera exacta y entera. También dentro de la implementación están las herramientas para visualizar los resultados y obtener métricas que permitan validar y evaluar el modelo de manera adecuada.

Evaluación corresponde a la parte final del proceso en donde se da una valoración a cada implementación del modelo. La metodología para lograr lo anterior es probar con varias configuraciones de parámetros, variables, implementaciones, *solvers*, entre otros, obteniendo distintos valores de las métricas. Mediante esto, se puede concluir acerca del desempeño del trabajo realizado.

Las actividades de las cuales se entra en más detalle en la Sección 4 son:

1. **Función objetivo:** Se define la función objetivo con los términos que la componen. Cada uno de estos debe representar un sentido intuitivo, de la solución que se busca. Para moldear la solución y descartar casos indeseados, se utiliza la función objetivo para empujar el óptimo a un calendario de reposición realista.
2. **Restricciones:** Se busca definir restricciones que sean lo más generales posibles, con el fin de que sean rígidas. Para realizar la evaluación, se utilizan distintas configuraciones de la función objetivo, sin embargo, las restricciones se mantienen fijas durante todo el trabajo. Por lo tanto, las restricciones están asociadas principalmente a dar sentido al problema, utilizando restricciones estándar como las que se encuentran en trabajos anteriores [12][13][37][42].
3. **Variables:** Se definen las variables del problema, entre las cuales debe estar la reposición semanal.

4. **Parámetros:** Corresponde a las entradas que recibe el modelo. Se diferencian en dos tipos: el primero corresponde a entradas modificables por el usuario, y los segundos son parámetros que están fijos y no los controlan los usuarios.

5. **Implementación por ventanas:** Para realizar la optimización se toman en cuenta una cantidad determinada de semanas que el modelo conoce por adelantado, con el fin de anticiparse a futuros *peaks* en la demanda. Con esto se puede distribuir de mejor manera el inventario en tiendas, reponiendo de forma anticipada para que no se produzcan quiebres de *stock* en las semanas de mayor demanda, debido a que existe un límite de reposición semanal asociado a restricciones logísticas (por ejemplo, una limitada capacidad de camiones para reponer a tiendas).

6. **Implementación Naive:** Se implementa un algoritmo que no conoce el la demanda futura, lo que puede provocar mayor cantidad de quiebres de *stock*.

7. **Implementación función objetivo:** Se implementa el mismo modelo por ventanas mencionado anteriormente pero esta vez con cada una de las combinaciones de funciones objetivo posibles. Esto con la finalidad de evaluar el tiempo de ejecución, demanda satisfecha y utilidades dada las distintas configuraciones.

8. **Pruebas de estrés:** Se implementa el mismo modelo por ventanas mencionado anteriormente, variando la cantidad de SKU y tiendas en el calendario de reposición, para obtener el desempeño en tiempo de ejecución del algoritmo. Adicionalmente, esto se realiza para distintos *solvers*, para comparar el desempeño de estos.

9. **Pruebas de ruido en *forecast* de demanda:** Se agrega ruido al *forecast* de demanda para evaluar la sensibilidad al incorporar la incertidumbre de la predicción. Se evalúa mediante cantidad de quiebres que se obtienen en función del nivel de ruido que se introduce.

10. **Prueba de implementación en distintas situaciones:** Dependiendo de los parámetros utilizados, el calendario de reposición se comporta de manera distinta, por lo que se realizan combinaciones de parámetros que entregan situaciones intuitivas e ilustrativas para sacar conclusiones de la implementación en estas situaciones.

3.3. Nomenclatura

Se presentan en las Tablas 3.1 y 3.2 un resumen de las siglas presentes en el trabajo, junto con los parámetros y variables usadas en el modelo y las simulaciones.

Tabla 3.1: Descripción de cada sigla utilizada en el trabajo.

Sigla	**Significado**
SKU	Stock Keeping unit
CD	Centro de distribución
LP	Programación lineal
ILP	Programación lineal de enteros
R5	Ryzen 5 4650 PRO, 16GB RAM y sistema operativo Win10

Tabla 3.2: Descripción y unidades de cada uno de los parámetros y variables utilizadas en el trabajo. Describe también los índices de dependencia de cada una: i (depende del SKU), j (depende de la tienda) y t (depende de la semana). La columna Tipo indica si corresponde a una variable de decisión del modelo (v) o corresponde a un parámetro predefinido (p) para las simulaciones.

	Descripción	**Índices**	**Tipo**	**Unidad**
R	Reposición	i, j, t	v	
δ	Variable binaria lógica	i, j, t	v	
S_{CD}	*Stock* en centro de distribución	i, t	v	
$\tilde{Q}$	Demanda satisfecha	i, j, t	v	
Inv	Inventario	i, j, t	v	
F	*Forecast* de demanda	i, j, t	p	
P	Precio	i, j, t	p	CLP
C	Costo	i, j, t	p	CLP
S_{CD0}	*Stock* inicial en centro de distribución	i	p	
F_{vol}	Volumen de cada SKU	i, j, t	p	m^3
T	Tamaño de ventana		p	semanas
I	Cantidad de SKUs		p	
J	Cantidad de semanas		p	
Inv_0	Inventario inicial	i, j	p	
B	Capacidad en tienda	j, t	p	m^3
Me	Mínimo de exhibición	i, j, t	p	
Tr	Límite máximo de transporte	t	p	m^3
a_1	Constante dimensional Z_1		p	
a_2	Constante dimensional Z_2		p	CLP
a_3	Constante dimensional Z_3		p	CLP
a_4	Constante dimensional Z_4		p	CLP/m^3
σ	Parámetro de control de ruido en *forecast*		p	

Capítulo 4

Desarrollo y resultados

Para hacer alusión a los objetivos del trabajo, se busca realizar simulaciones, implementando algoritmos de optimización, para conseguir un calendario de reposición de una cantidad determinada de SKUs a una cantidad determinada de tiendas. En este sentido, se presenta primero el modelo realizado para optimizar, que tiene como salida el calendario de reposición semanal. Posteriormente se implementa un *script* en Python con distintos *solvers* lineales que permiten realizar la optimización del modelo señalado anteriormente. Por último se implementan una serie de algoritmos que permiten variar los parámetros para evaluar el desempeño del modelo desarrollado.

4.1. Modelo

Esta Sección presenta el modelo de optimización desarrollado en este trabajo, esto incluye las variables y parámetros involucradas, como así también los términos de la función objetivo y el significado de las restricciones. Toda la notación utilizada se encuentra de manera resumida en la Tabla 3.2.

4.1.1. Función objetivo

Se presenta en la siguiente ecuación la función objetivo que se utiliza para modelar el problema de optimización.

$$\text{Max.} \quad \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{t=1}^{T} \Bigg[a_1 \overbrace{\tilde{Q}_{i,j,t}\left(P_{i,j,t} - C_{i,j,t}\right)\cdot(T-t)^2}^{Z_1} + a_2 \overbrace{\tilde{Q}_{i,j,t}\cdot(T-t)^2}^{Z_2} + a_3 \underbrace{\delta_{i,j,t}\cdot F_{i,j,t}\cdot(T-t)^2}_{Z_3} - a_4 \underbrace{S_{CD\ i,t}\cdot F_{vol\ i,j,t}\cdot t^2}_{Z_4} \Bigg]$$

Las constantes a_1, a_2, a_3 y a_4 son constantes de valor 1 y con unidades descritas en la Tabla 3.2. Se identifican 4 términos en la función objetivo, donde cada uno de estos tiene una finalidad determinada detallada más adelante. En la Sección 4.3.1 se evalúa el modelo utilizando todas las combinaciones de términos en función objetivo para identificar el impacto real que tienen.

El término Z_1 corresponde al beneficio económico que se obtienen por ganancias de la venta de productos, en este caso se tiene que la ganancia de venta de un producto es el precio de venta P menos el costo de éste C por la demanda $\tilde{Q}$. Se busca que este término haga que la solución prefiera la venta y reposición de productos que generan mejores ganancias, así siempre se tendrá *stock* en tiendas de estos productos. Sólo implementar este término puede producir que productos que generan menores utilidades sean desfavorecidos en la reposición, generando que no se satisfaga toda la demanda, además de afectar visualmente la variedad de productos presentes en la tienda. Para ejemplificar esto, si el algoritmo detecta que para optimizar la utilidad sólo se deben reponer *jeans* de mujer azules, no significa que en la tienda se deba vender sólo este producto. El efecto de la variedad en el *mix* (productos en la tienda) también atrae a los clientes.

El término Z_2 da prioridad a vender la mayor cantidad de productos lo antes posible, sin importar la ganancia que generen. Sólo tener este término puede generar efectos negativos en la solución para la reposición, en lugar de generar una reposición homogénea, prioriza productos con demandas altas sin importar la ganancia que estos generan, dejando con poco *stock* en tiendas los productos de menor demanda. El efecto de vender la mayor cantidad de productos lo antes posible, es principalmente para evitar que estos productos entren en liquidación, es decir, que se reduzca el margen por unidad debido a que el producto no se vende. Hay que recordar que las temporadas avanzan y que nuevas colecciones de productos (tendencias, moda, etc) van dejando obsoletos los productos antiguos.

El término Z_3 es mayor que cero sólo cuando se cumple la demanda, cuando se genera un quiebre de *stock* este término tiene valor cero. Su inclusión en el modelo aporte, de manera explicita, una preferencia por las soluciones que entregan la menor cantidad de quiebres de *stock* posibles, situaciones indeseadas por los *retailers*. Un quiebre de *stock* es muy grave, ya que significa que es un producto que está siendo demandado, pero el *retail* no está siendo capaz de solventar esa demanda en el momento adecuado. Esto posiblemente genera una fuga de clientes y un deterioro de imagen. El cliente finalmente buscará ese producto en otro lugar.

Por último, el término Z_4 es un término que cumple una función de costo, la razón de tener un signo negativo antes de este término. Este costo está asociado a mantener los productos por tiempos prolongados en el centro de distribución, provocando que la solución encontrada busque tener tiendas llenas, o a la cota seleccionada mediante el parámetro B. Este costo no es real para los casos en que el *retail* es dueño de las bodegas de almacenamiento, sin embargo es un costo "virtual" ya que son espacios que son preciados y que no se pueden desperdiciar en productos que son poco demandados. De alguna forma, hay que forzar que estos productos salgan de las bodegas a las tiendas, para liberar espacio.

Cabe destacar que el factor $(T-t)$ presente en los 3 primeros términos tiene como objetivo que se priorice cumplir con la demanda los periodos semanales más próximos.

4.1.2. Restricciones

Las restricciones implementadas están directamente relacionadas con lo que se encuentra a lo largo de la bibliografía, como también a la realidad de la empresa y sus necesidades. Las restricciones utilizadas en el modelo se encuentran desde la Ecuación 4.1 a la Ecuación 4.8. Se presenta a continuación una descripción de cada una de estas restricciones.

Restricción 4.1 Límite de capacidad en tiendas. Cada tienda tiene una capacidad limitada de productos que se pueden almacenar y exhibir. Esta capacidad máxima se representa por el parámetro $B_{j,t}$. Notar que B depende de t, por lo que esta entrada del modelo puede ser modificada manualmente para variar la capacidad de ocupación en la tienda semana a semana. Esto puede ser especialmente útil para guardar espacio en tiendas y utilizarlo semanas previas a un *peak* de ventas.

Restricción 4.2 Límite de *stock* en CD. Responde a una necesidad lógica entre las variables R y S_{CD}. La cantidad de productos que se reponen semanalmente de un SKU no puede ser mayor que la cantidad que se tiene de ese SKU almacenado en el centro de distribución.

Restricción 4.3 Reposición positiva. El flujo de productos es siempre desde el CD a las tiendas (R mayor que cero) y no se devuelven desde las tiendas al CD (R menor que cero).

Restricción 4.4 Continuidad de inventario. Restricción necesaria para guardar una relación entre lo que se repone, lo que se vende, y el inventario guardado. El nombre de continuidad es derivado de las clásicas ecuaciones de continuidad de la física, que buscan señalar, de manera coloquial: "Lo que entra (a las tiendas) más lo que tenía, es igual a lo que sale, más lo que queda almacenado".

Restricción 4.5 Continuidad de *stock* en CD. Análogo a la restricción anterior pero esta vez aplicado al *stock* en el centro de disribución. "Lo que tenía (en el CD), es igual a lo que sale, más lo que queda almacenado".

Restricción 4.6 Mínimo de exhibición en tienda. Cada SKU en cada tienda tiene fijo un mínimo de productos que se deben mantener para exhibirlos en las tiendas, mientras exista *stock* suficiente en el CD para suplir estos mínimos.

Restricción 4.7 Límite de transporte. Existe una limitante logística asociada la cantidad de productos que se pueden repartir a las tiendas. Para llevarlo a un caso realista, esta capacidad límite está asociado a la cantidad de camiones que se tienen para repartir semanalmente, y el volumen de transporte de estos camiones. El parámetro F_{vol} aporta información del espacio que ocupa cada producto, por ejemplo, 10 pares de calcetines utilizan menos espacio que 10 refrigeradores.

Restricción 4.8 Restricción lógica de demanda satisfecha. La demanda que se satisface es F siempre y cuando se tenga *stock* suficiente para suplirla. En caso contrario la demanda satisfecha es el *stock* en la tienda (que corresponde al inventario más la reposición). Se modela de manera computacional esta restricción con la formulación Big-M.

$$\sum_{i=1}^{I} \left(Inv_{i,j,(t-1)} + R_{i,j,t}\right) \cdot F_{vol\ i} \leq B_{j,t} \qquad \forall j \in [1,J],\ \forall t \in [1,T] \tag{4.1}$$

$$\sum_{j=1}^{J} R_{i,j,t} \leq S_{CD\ i,t} \qquad \forall i \in [1,I],\ \forall t \in [1,T] \tag{4.2}$$

$$R_{i,j,t} \geq 0 \qquad \forall i \in [1,I],\ \forall j \in [1,J],\ \forall t \in [1,T] \tag{4.3}$$

$$Inv_{i,j,t-1} + R_{i,j,t} - \tilde{Q}_{i,j,t} - Inv_{i,j,t} = 0 \qquad \forall i \in [1,I],\ \forall j \in [1,J],\ \forall t \in [1,T] \tag{4.4}$$

$$S_{CD\ i,t} + S_{CD\ i,(t-1)} + \sum_{j=1}^{J} R_{i,j,t} = 0 \qquad \forall i \in [1,I],\ \forall t \in [1,T] \tag{4.5}$$

$$Me_{i,j,t} \leq I_{i,j,t-1} \qquad \forall i \in [1,I],\ \forall j \in [1,J],\ \forall t \in [1,T] \tag{4.6}$$

$$\sum_{i=1}^{I} \sum_{j=1}^{J} R_{i,j,t} \cdot F_{vol\ i} \leq Tr_t \qquad \forall t \in [1,T] \tag{4.7}$$

$$min\left(Inv_{i,j,t-1} + R_{i,j,t}\ ,\ F_{i,j,t}\right) = \tilde{Q}_{i,j,t} \qquad \forall i \in [1,I],\ \forall j \in [1,J],\ \forall t \in [1,T] \tag{4.8}$$

4.2. Implementación del modelo

4.2.1. Parámetros generales

En esta Sección se presentan los parámetros por defecto que se serán utilizados en las simulaciones. En caso que una simulación utilice valores distintos a los comunes, será informado de forma explícita. Los valores de los parámetros son presentados en la siguiente lista:

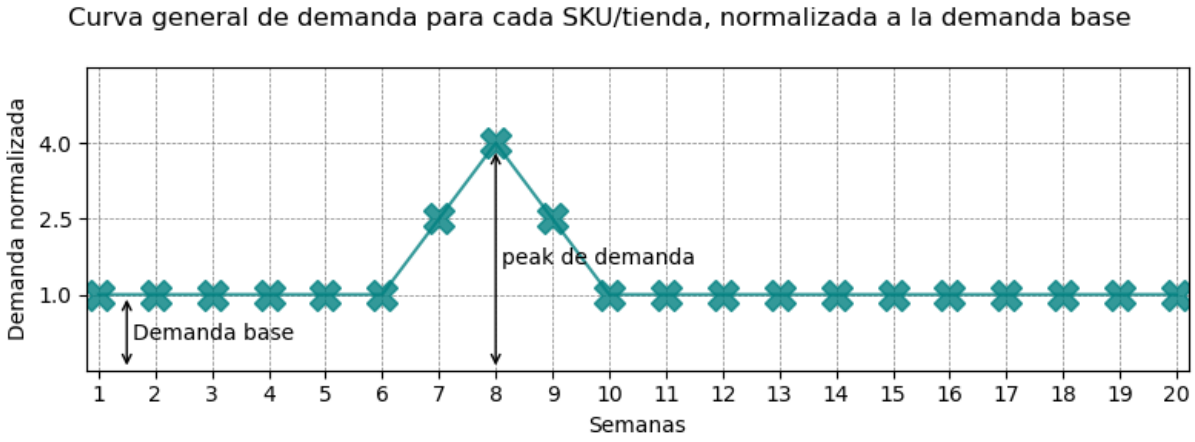

Figura 4.1: Ejemplo de curva de *forercast* de demanda, normalizada a la demanda base (demanda en semana 1).

- ***Forecast* de la demanda (F):** Curva de veinte semanas. se considera la misma

demanda (demanda base) para las semanas 1, 2, 3, 4, 5, 6, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 20. Las semanas 7, 8 y 9 contemplan un *peak* de demanda, la semana 8 es el cuádruple de la demanda base le SKU. La semana 7 tiene una demanda igual al promedio de la demanda en las semanas 6 y 8. La demanda de la semana 9 es el promedio de demanda para las semanas 8 y 10, simulando crecimiento lineal. Un ejemplo cualitativo se presenta en la Figura 4.1.

- **Precio (P):** Se fija un precio original para cada producto en cada tienda, el cual se mantiene por seis semanas, luego de eso comienza la liquidación del SKU con tres semanas al 80 %, luego tres semanas al 70 %, luego 3 semanas al 50 % y finalmente 5 semanas al 30 % del precio original, lo que suma en total veinte semanas. Valores escogidos de manera aleatoria entre 2,990 y 11,990. Un ejemplo cualitativo se presenta en la Figura 4.2.

- **Costo (C):** Corresponde al 30 % del valor original del producto.

- **Volumen de SKUs (F_{vol}):** Se considera que todos los SKUs ocupan el mismo volumen. Para las simulaciones se toma $F_{vol} = 1$ por simplicidad.

- **Capacidad de tiendas (B):** Capacidad, en volumen, de almacenaje en las tiendas para productos. Valores escogidos de manera aleatoria entre $500 \cdot I$ y $1000 \cdot I$. Se escoge un valor inicial de capacidad y se mantiene constante a lo largo de las 20 semanas.

- **Inventario inicial (Inv_0):** Se considera que el inventario de los SKU seleccionados para la optimización no presentan inventario previo en tiendas, como si fueran productos nuevos realizando la primera reposición a tiendas en semana 1.

- **Mínimo de exhibición (Me):** Cantidad mínima de productos que se deben mantener para exhibirlos en las tiendas. Valores escogidos de manera aleatoria entre 40 y 100. Se escoge un valor inicial de mínimos de exhibición y se mantiene constante a lo largo de las 20 semanas.

- **Límite de transporte (Tr):** Capacidad, en volumen, que se tiene para transportar semanalmente la reposición de productos a tiendas. Como ejemplo se puede tomar el espacio total dentro de los camiones de la compañia para transportar productos. Toma el valor $500 \cdot I \cdot T$. Se escoge un valor inicial de capacidad y se mantiene constante a lo largo de las 20 semanas.

- ***Stock* inicial en CD (S_{CD0}):** Cantidad de productos por SKU que se tienen en el centro de distribución antes de que comience la reposición la primera semana.

- **Ventana de semanas (T):** Cantidad de semanas que se consideran en cada optimización. Toma el valor 8 para la mayoría de las simulaciones, en el caso *Naive* tiene valor 1.

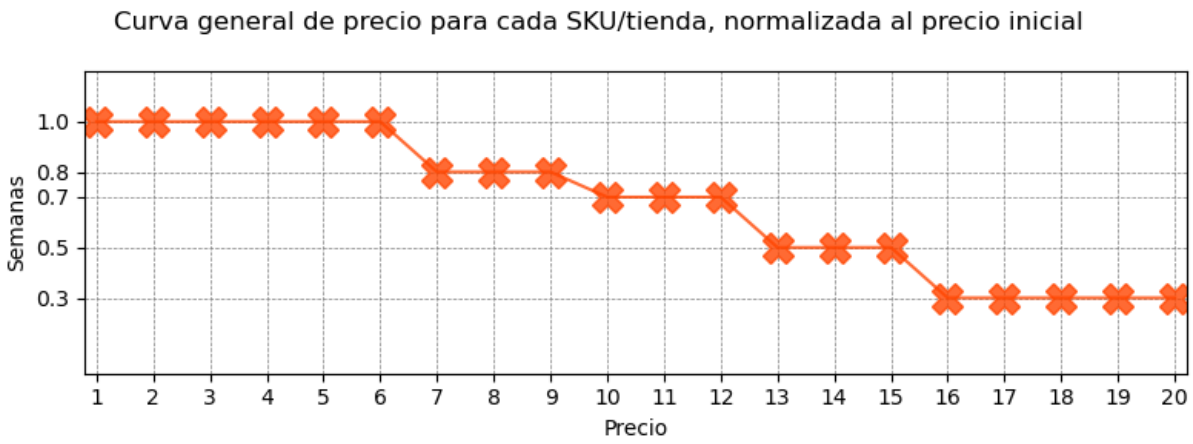

Figura 4.2: Ejemplo de curva de precio, normalizada al precio inicial del producto en esa tienda (precio en semana 1).

Se utiliza una semilla (ver Sección 3.1) para que los valores escogidos aleatoriamente siempre sean los mismos para simulaciones distintas.

Los valores que se señalan en las Figuras presentes en el trabajo están normalizadas por algún factor para facilitar el análisis cualitativo, para esto se definen cuatro índices distintos, asociados a cada uno de los factores de normalización: índice de demanda, índice de ocupación, índice de *stock* CD y índice de transporte:

1. **Índice de demanda** es definido para implementarlo a nivel de SKU/tienda, facilitando la visualización del *stock* de cada SKU en cada tienda (inventario más reposición). Con esto, todos los valores de reposición e inventario están divididos por el valor del *forecast* de demanda en la semana 8, que corresponde al valor máximo de la demanda. Este índice toma valores en el rango $[0, B_{j,t}/F_{i,j,8}]$. Si se considera como ejemplo el SKU i en la tienda j, el valor 0 del índice de demanda corresponde a que no hay *stock* del SKU i en la tienda j, por otro lado, un valor $B_{j,t}/F_{i,j,8}$ indica que está ocupada toda la capacidad de la tienda j sólo con productos del SKU i.

$$\text{Índice de demanda } (i,j,t) = \frac{Inv_{i,j,t-1} + R_{i,j,t}}{F_{i,j,8}} \tag{4.9}$$

2. **Índice de ocupación** corresponde a la fracción de la capacidad de la tienda $B_{j,t}$ que está ocupada. Este índice toma valores en el rango $[0, 1]$. Valor 0 corresponde a tienda sin productos y valor 1 corresponde a tienda ocupada completamente.

$$\text{Índice de ocupación } (j,t) = \frac{1}{B_{j,t}} \sum_{i=1}^{I} \left(Inv_{i,j,(t-1)} + R_{i,j,t} \right) \cdot F_{vol\ i} \tag{4.10}$$

3. **Índice de *stock* en CD** corresponde a la fracción del *stock* inicial en el centro de distribución. Los valores de S_{CD} se dividen en el *stock* inicial del SKU correspondiente. Este índice toma valores en el rango $[0, I]$. La primera semana siempre toma el valor I.

$$\text{Índice de demanda } (i,t) = \sum_{i=1}^{I} \frac{S_{CD\ i,t}}{S_{CD0\ i}} \tag{4.11}$$

4. **Índice de transporte** es utilizado para representar las reposiciones semanales en función del límite de trasporte semanal. Este índice toma valores en el rango $[0, 1]$. Índice de transporte igual a 1 implica que todos los camiones se están utilizando a toda su capacidad.

$$\text{Índice de transporte } (t) = \frac{1}{Tr_t} \sum_{i=1}^{I} \sum_{j=1}^{J} R_{i,j,t} \cdot F_{vol\ i} \tag{4.12}$$

4.2.2. Parámetros particulares

En esta Sección se detallan los valores que toman los parámetros utilizados específicamente para los casos con dos tiendas y dos SKUs. Por un tema práctico, sólo se presentan los parámetros con poco tiendas y pocos SKUs (Tabla 4.1). Los parámetros que se presentan son: demanda base ($F_{i,j,1}$), precio inicial ($P_{i,j,1}$), capacidad de tiendas inicial ($B_{j,1}$), *stock* inicial en CD (S_{CD0}), capacidad de transporte inicial (Tr_1) y mínimos de exhibición iniciales ($Me_{i,j,1}$).

Tabla 4.1: Parámetros utilizados en todas las simulaciones donde se tienen dos tiendas y dos SKUs. Solo corresponden a los valores iniciales, que entran al modelo en la semana 1. Los valores que toman estos parámetros para otras semanas se mencionan en la Sección 4.2.1, en esa misma Sección se detalla el valor que toman otros parámetros no mencionados aquí. La última columna (Uds.) corresponde a las unidades en que están los valores de cada producto, los cuadros vacíos corresponden a cantidad de productos.

	Tienda 01		**Tienda 02**		
	SKU 01	**SKU 02**	**SKU 01**	**SKU 02**	**Uds.**
Mínimos de exhibición iniciales	100	50	50	40	
Demanda base	200	150	250	100	
Precio inicial	2,990	11,990	3,990	10,990	CLP
Capacidad de tiendas inicial	1,000		800		m^3
Capacidad de transporte inicial	2,000				m^3

	SKU 01	**SKU 02**	**Uds.**
***Stock* inicial en CD**	10,000	20,000	

4.2.3. Implementación por ocho semanas

Para evitar quiebres de *stock*, o *sobre-stock*, se implementa un modelo que utilice el *forecast* de la demanda futura, lo que permite equipar las tiendas con las cantidades que aseguren un cumplimiento de la demanda. Para efectos de la implementación de este trabajo, se asume que

el *forecast* es un *input* conocido. Se escribe un *script* en Python que realice la optimización de ocho semanas, lo que permite anticiparse a la demanda en dos meses por adelantado aproximadamente. Se escoge este lapso temporal debido a que actualmente, en la empresa, las reposiciones para semanas de gran demanda, como navidad, comienzan con dos meses de anticipación. No se utiliza un número mayor se semanas debido a que el *forecast* se vuelve menos preciso. Se le llama a la implementación de ésta Sección 4.2.3 como "caso normal", término que se utiliza más adelante para referirse a la implementación de ventana móvil de ocho semanas.

Las simulaciones contemplan trece iteraciones en el algoritmo de optimización (trece optimizaciones tipo "*rolling*", considerando una "ventana movil" de ocho semanas) para generar un calendario de reposición de trece semanas, a excepción de la implementación *Naive*, la cual contempla veinte iteraciones. En cada iteración se consideran ocho semanas. De los resultados obtenidos sólo son importantes, y por lo tanto los valores que se fijan, de la primera semana dentro de la iteración y los resultados de las restantes siete semanas se descartan. Se realiza de esta manera para que cada reposición en el calendario semanal considere las ocho semanas por adelantado y también permite incorporar a la implementación un *forecast* actualizado semana a semana. Un ejemplo del método descrito anteriormente se muestra en la Figura 4.3, se describe cualitativamente como se genera el gráfico (a) de la Figura 4.4.

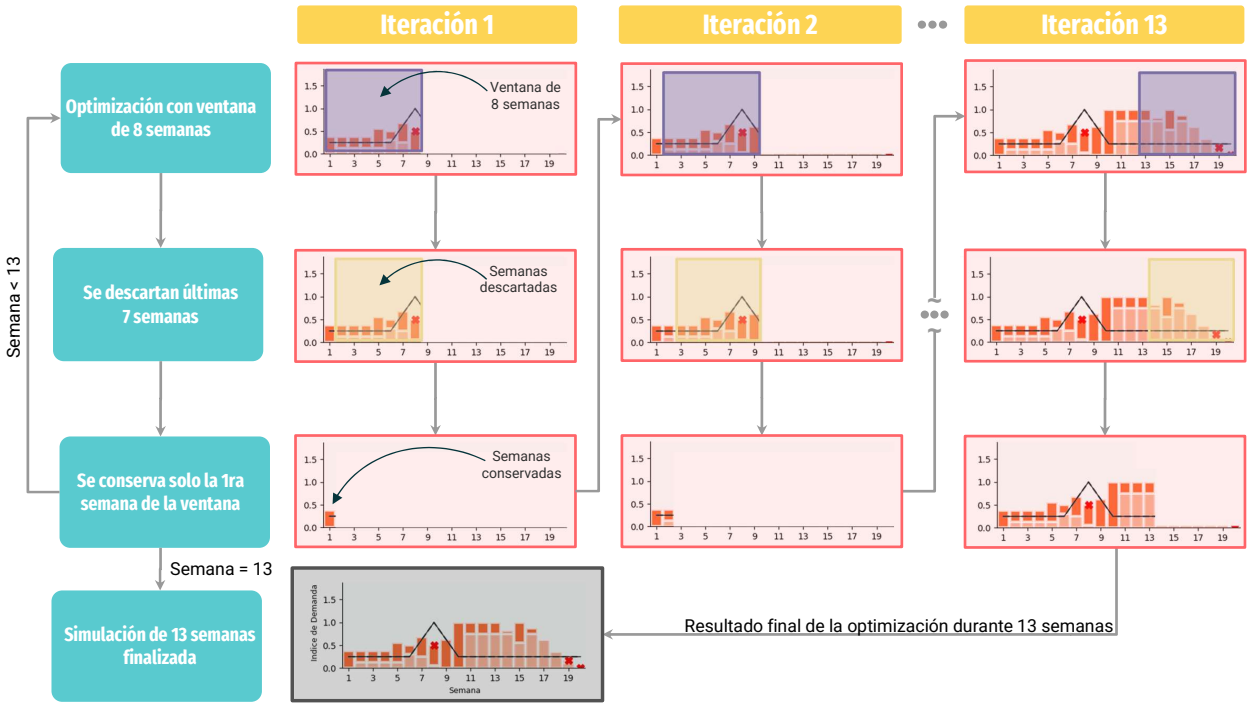

Figura 4.3: Ejemplo de implementación con ventana de ocho semanas, dos SKUs y dos tiendas

Se toma como supuesto que no hay ingreso de productos al centro de distribución en las simulaciones, sólo se contempla el la cantidad inicial ya existente en el CD. Las simulaciones de esta sección de realizan con el *solver* Gurobi.

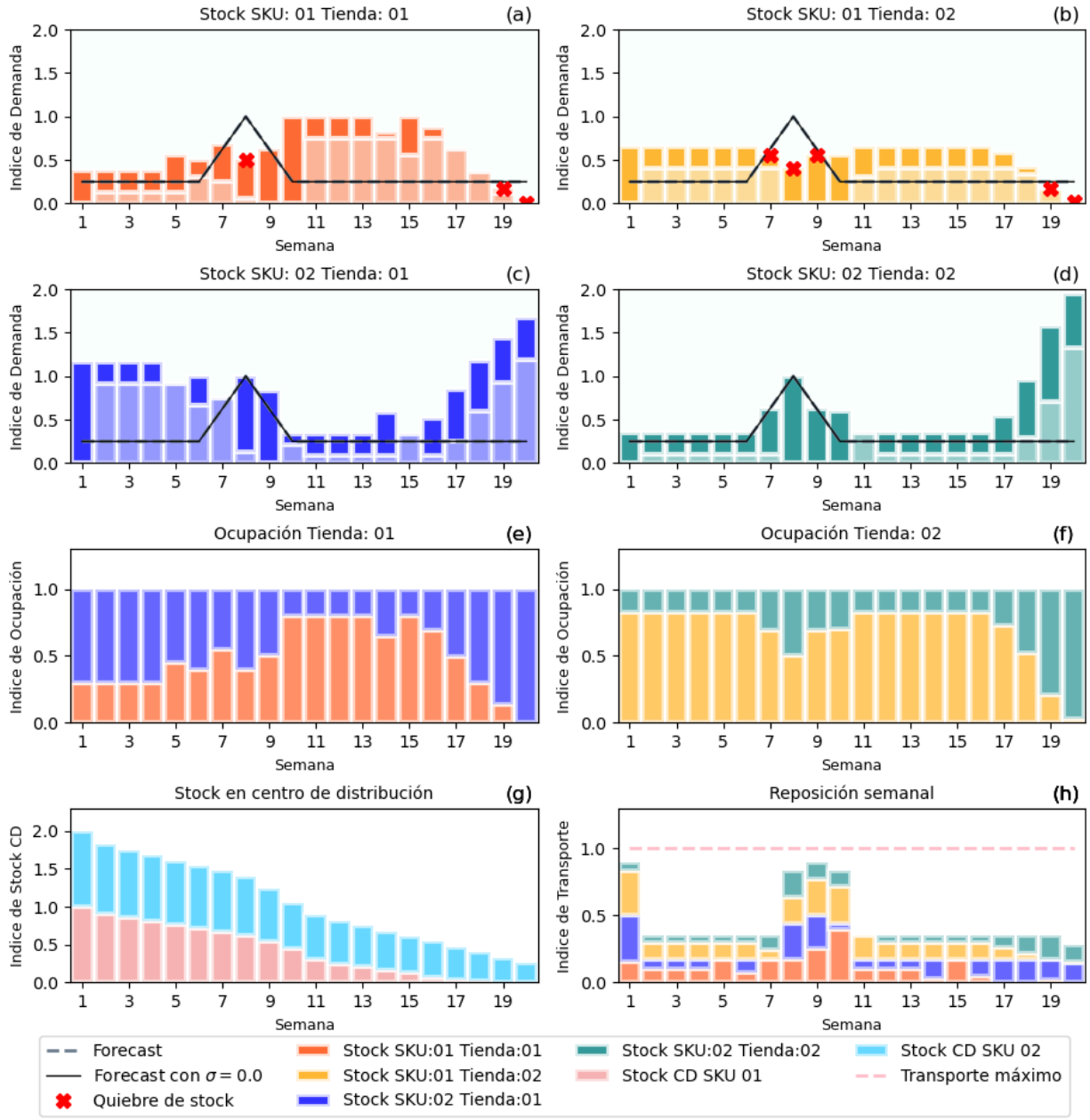

Figura 4.4: Resultados implementación de trece iteraciones con ventana temporal de ocho semanas. Se consideran dos SKUs y dos tiendas, con parámetros señalados en las Secciones 4.2.1, 4.2.2 y 4.2.3.

Los resultados presentes en la Figura 4.4 corresponden a la implementación del modelo considerando dos SKUs y dos tiendas. La implementación de esta Sección 4.2.3, y la siguiente Sección 4.2.4, se realizan en R5 (ver Tabla 3.1). Los SKUs utilizados tienen contribuciones distintas (aportan distintas utilidades). Las barras presentes en los gráficos de la Figura tienen distintos colores dependiendo del SKU. Los colores naranjo, amarillo y rosa están asociados

al SKU 01, y los colores azul, celeste y verde asociados al SKU 02. Cada gráfico contempla veinte semanas, de las cuales las trece primeras son reposiciones fijas, y las últimas 7 son las descartadas de la iteración número 13. Los gráficos están normalizados según se explica en la Sección 4.2.1.

La misma Figura 4.4 contiene 8 gráficos. Los gráficos (a), (b), (c) y (d) corresponden al *stock* de un SKU en una determinada tienda, según corresponde. Cada barra en estos gráficos representa el *stock* al inicio de la semana, la parte más clara de la barra corresponde al inventario restante de la semana anterior, y la más oscura la reposición hecha la semana actual. Se presenta en linea discontinua el *forecast* de demanda y en línea continua la curva con ruido (explicado en más detalle en la Sección 4.4.1), estas curvas coinciden cuando el ruido tiene como parámetro $\sigma = 0$. Están presentes en los gráficos (a) y (b) marcadores en forma de `X` rojas sobre las barras en las cuales no se alcanza el valor de la demanda, lo que llamamos como un quiebre de *stock*. Cada barra en los gráficos (e) y (f) representa la cantidad de productos en la tienda al inicio de la semana correspondiente, se diferencian con colores los distintos SKUs en la tienda. El gráfico (g) muestra el decaimiento del *stock* de cada SKU en el centro de distribución. Se diferencian con colores los distintos SKUs en el CD. Por último, en el gráfico (h), se presentan las reposiciones semanales (R) de cada SKU a cada tienda, normalizado a la capacidad de transporte. Este gráfico permite conocer la capacidad que se tiene para seguir trasportando productos.

De la Figura 4.4 se puede destacar que las tiendas están totalmente ocupadas con índice de ocupación 1, lo que genera que el SKU 01 presente quiebres de *stock* en las semanas 7, 8 y 9, que son las semanas de mayor demanda. Cabe destacar que la reposición semanal sólo en algunas semanas se acerca a alcanzar el máximo, con valores superiores a 0.8 de índice de transporte en las semanas 1, 8, 9 y 10. Por último, mencionar que el SKU 01 agota el *stock* en el centro de distribución en la semana 17, esto guarda relación con las marcas de quiebres presentes en las semanas 19 y 20 de los gráficos (a) y (b).

4.2.4. Implementación *Naive*

Se utiliza la misma implementación que en la Sección 4.2.3, variando sólo el tamaño de la ventana, que en este caso es 1. Se llama a esta implementación *Naive* debido a que en cada iteración semanal no ve la demanda futura, por lo que ingenuamente puede asignar reposiciones que traen un mayor valor de la función objetivo en el corto plazo, sin embargo pueden causar quiebres de *stock* a futuro a causa de una mala distribución del *stock*, de cada SKU, dentro de las tiendas. Se presenta en la Figura 4.5 los resultados de la implementación descrita anteriormente.

Si bien los valores en los gráficos se presentan normalizados, la Tabla 4.2 muestra los valores obtenidos en la implementación de esta sección y la anterior (Sección 4.2.3). Se puede apreciar que al agregar una ventana temporal más grande se puede anticipar a demandas futuras y reponer en consecuencia, lo que se traduce en mayor cantidad de unidades vendidas, mayores ingresos y menos quiebres de stock. Sin embargo, tener una ventana temporal de 8 semanas genera un algoritmo más complejo, demorando el tripe para dos SKUs y dos tiendas.

De las Figuras 4.4 y 4.5 se puede destacar las diferencias visuales el los gráficos (h). La

Figura 4.5 muestra exactamente la misma reposición entre las semanas 2 y 6, mientras que en la Figura 4.4 se ve una diferencia a partir de la semana 5.

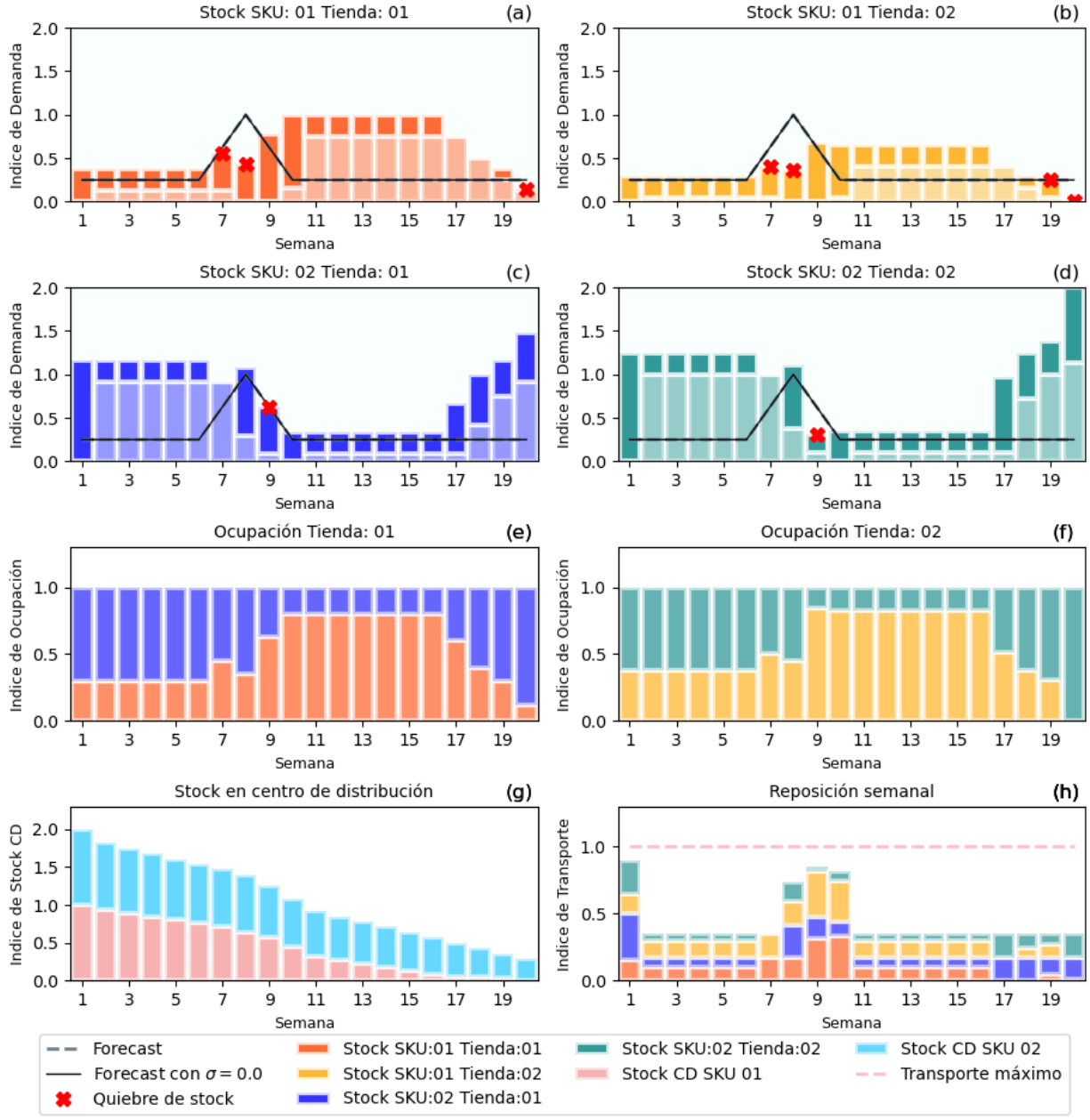

Figura 4.5: Resultados implementación de trece iteraciones con ventana temporal de una semanas (implementación *Naive*). Se consideran dos SKUs y dos tiendas, con parámetros señalados en las Secciones 4.2.1, 4.2.2 y 4.2.3.

Tabla 4.2: Rendimiento implementación con ventana de ocho semanas *vs* caso Naive. Los resultados en esta Tabla contemplan sólo las semanas entre la 1 y la 13, los resultados desde la semana 14 a la 20 se desestiman (detalles en Sección 4.2.3).“# de unidades” corresponde a la cantidad de productos que satisfacen la demanda, dentro del modelo se representa con la variable $\tilde{Q}$. “# de quiebres” corresponde a la cantidad de quiebres de *stock* detectados en los resultados entre las semanas 1 y 13, los que se pueden identificar rápidamente, de manera visual, de las Figuras 4.4 y 4.5. El tiempo de la última columna corresponde al tiempo de ejecución del *script* en Python, en R5. Las utilidades se presentan en millones de pesos chilenos (MCLP) y corresponde a la contribución por venta, esto es exactamente el término Z_1 de la función objetivo, detallada en la Sección 4.1.1.

Caso	**Ventana [semanas]**	**Utilidades [MCLP]**	**# de unidades**	**# de quiebres**	**Tiempo [s]**
Naive	1	42.2	11809	6	0.07
Normal	8	43.3	12148	4	0.25

Tabla 4.3: Rendimiento variando términos de la función objetivo. Se consideran dos tiendas y dos SKUs. Columnas en la Tabla y parámetros utilizados se detallan en la Tabla 4.2.

Términos en F.O.				**Utilidades**	**# de unidades**	**# de quiebres**	**Tiempo**
$\mathbf{Z_1}$	$\mathbf{Z_2}$	$\mathbf{Z_3}$	$\mathbf{Z_4}$	**[MMCLP]**			**[s]**
✓				42.38	11450	4	0.24
	✓			38.92	11450	5	0.24
		✓		37.67	10610	4	0.25
			✓	42.07	11279	4	0.29
✓	✓			42.39	11450	4	0.24
✓		✓		42.39	11450	4	0.24
✓			✓	42.39	11450	4	0.29
	✓	✓		39.29	11220	4	0.29
	✓		✓	42.07	11280	4	0.28
		✓	✓	39.29	11219	4	0.30
✓	✓	✓		42.39	11450	4	0.24
✓	✓		✓	42.39	11450	4	0.23
✓		✓	✓	42.38	11448	4	0.25
	✓	✓	✓	39.29	11220	4	0.32
✓	✓	✓	✓	42.39	11450	4	0.32
Promedio				41.18	11325	4.1	0.27
Mínimo				37.67	10610	4	0.23
Máximo				42.39	14450	5	0.32
Desv. Est.				1.66	214	0.25	0.03

4.3. Pruebas sobre la implementación

4.3.1. Variación de términos en función objetivo

En esta Sección se ven los resultados al probar con cada una de las combinaciones de términos el función objetivo mostrada en la Sección 4.1.1. Los resultados que se muestran en la Tabla 4.3, tienen la misma combinación de parámetros que en la Sección 4.2.3, sólo varía la función objetivo. La resolución del problema planteado se realiza mediante el *solver* Gurobi, a través de un *script* de Python.

Tabla 4.4: Rendimiento variando términos de la función objetivo. Se consideran diez tiendas y diez SKUs. Columnas en la Tabla y parámetros utilizados se detallan en la Tabla 4.2, a excepción de la columna Utilidades, cuyas unidades en esta Tabla son miles de millones de pesos (MMCLP).

Términos en F.O.				**Utilidades**	**# de unidades**	**# de quiebres**	**Tiempo**
$\mathbf{Z_1}$	$\mathbf{Z_2}$	$\mathbf{Z_3}$	$\mathbf{Z_4}$	**[MMCLP]**			**[s]**
✓				1252	319709	18	5.33
	✓			1245	318844	24	4.77
		✓		1239	317549	24	56.88
			✓	1239	318447	25	7.11
✓	✓			1256	320666	18	3.87
✓		✓		1252	319385	18	13.82
✓			✓	1258	321358	15	6.13
	✓	✓		1228	314927	28	54.26
	✓		✓	1251	320554	26	5.72
		✓	✓	1226	314811	29	13.71
✓	✓	✓		1254	320158	17	12.70
✓	✓		✓	1250	322168	14	7.23
✓		✓	✓	1250	318768	20	17.87
	✓	✓	✓	1240	318335	22	10.16
✓	✓	✓	✓	1254	320192	17	11.49
Promedio				1246	319058	21	15.40
Mínimo				1226	314811	14	3.87
Máximo				1258	322168	29	56.88
Desv. Est.				9	2020	5	16.23

Se destacan algunos valores de la Tabla 4.3: cuando el término Z_1 no es considerado (término que incluye precios y costos) se ven disminuidas las ganancias económicas en la mayoría de los casos. También cuando se incluye Z_2 sin el término Z_1, en general, se mantiene el valor de cantidad de unidades.

En la Tabla 4.4 se tienen resultados para un mayor número de SKUs y tiendas (diez SKUs y diez tiendas), con el objetivo de hacer más notorias las diferencias y ver patrones más marcados en el desempeño de cada función objetivo. Se destaca que la desviación estándar de

las ganancias, y las unidades, es menor al 1 %. La desviación estándar del número de quiebres es del 24 %. El tiempo de ejecución es la métrica de mayor variabilidad, donde el máximo supera en más de 3 veces al promedio.

4.3.2. Pruebas de estrés

Es importante conocer la complejidad de un algoritmo, que se traduce en mayor uso de memoria y tiempo de cómputo. En la Tabla 4.5 se presenta la cantidad de variables de decisión que participan en la optimización. También se señalan la cantidad de restricciones del problema, que dependen de I, J y T (cantidad de SKUs, tiendas, y ventana de semanas, respectivamente).

En las Tablas 4.6 y 4.7 los tiempos de ejecución de una iteración de optimización para la reposición, con una ventana temporal de 8 semanas, variando la cantidad de SKUs y tiendas, utilizando dos *solvers* distintos: Gurobi y PuLP CBC, respectivamente.

Tabla 4.5: Cantidad de variables y restricciones en el modelo desarrollado, donde I es la cantidad de SKUs, J la cantidad de tiendas y con ventana temporal de T.

Variables	**Restricciones**
$4\ IJT + IT$	$7\ IJT + 2\ IT + JT + T$

Dado lo que se señala en la Tabla 4.5, la cantidad de variables y restricciones es lineal con respecto a I, J o T, sin embargo, aumentar estos 3 parámetros a la vez hace que crezca rápidamente estos números. Se utilizan los resultados presentes en las Tablas 4.6 y 4.7, combinados con las formulas de la Tabla 4.5, para obtener el aumento en los tiempos de ejecución en función de la cantidad de variables y cantidad de restricciones. Estos resultados se presentan en forma de gráficos en las Figuras 4.6 y 4.7, donde se aprecia que las curvas de tiempo en función de número de variables y número de restricciones son muy similares, sin embargo cambian los valores en el eje horizontal. Se puede destacar también, de las mismas Figuras, que para el *solver* PuLP CBC se tienen puntos más cercanos entre ellos, pareciendo más una curva, y los puntos en la Figura 4.6 (para Gurobi) están más dispersos. Los valores de tiempos de ejecución son menores para el *solver* Gurobi, donde para 50000 variables, Gurobi demora alrededor de 20 segundos, mientras PuLP CBC demora 80 segundos aproximadamente.

Tabla 4.6: Tiempos de ejecución, *solver* Gurobi, T = 8. Por un tema práctico, no técnico, se representan como ">200" los valores mayores a 200 segundos. Se utilizan los parámetros señalados en las Secciones 4.2.1 y 4.2.3.

		# de tiendas							
		2	**8**	**15**	**20**	**25**	**30**	**40**	**50**
# de SKUs	**2**	0.01	0.07	0.10	0.10	0.13	0.16	0.22	0.29
	6	0.05	0.15	0.27	0.38	0.45	0.68	0.88	1.14
	10	0.07	0.26	0.48	0.66	0.90	1.05	1.61	2.29
	20	0.13	0.52	1.14	1.64	2.15	2.79	4.32	6.01
	50	0.38	1.59	3.90	8.66	12.19	13.84	38.37	27.12
	100	0.76	3.37	9.89	27.90	58.01	37.63	114.90	189.45
	200	1.59	20.19	31.56	133.42	196.84	>200	>200	>200
	500	8.37	54.82	153.99	>200	>200	>200	>200	>200

Tabla 4.7: Tiempos de ejecución, *solver* PuLP CBC, T = 8. Por un tema práctico, no técnico, se representan como ">200" los valores mayores a 200 segundos. Se utilizan los parámetros señalados en las Secciones 4.2.1 y 4.2.3.

		# de tiendas							
		2	**8**	**15**	**20**	**25**	**30**	**40**	**50**
# de SKUs	**2**	0.06	0.26	0.28	0.41	0.47	0.57	0.87	1.25
	6	0.12	0.63	1.28	10.97	2.30	5.3	3.26	>200
	10	0.15	0.76	2.27	2.53	3.97	5.82	7.55	>200
	20	0.42	2.15	5.31	7.19	7.87	10.03	21.11	40.03
	50	1.11	6.41	24.02	31.83	>200	67.38	149.28	>200
	100	2.35	21.63	78.95	150.52	>200	>200	>200	>200
	200	3.02	124.08	>200	>200	>200	>200	>200	>200
	500	10.81	>200	>200	>200	>200	>200	>200	>200

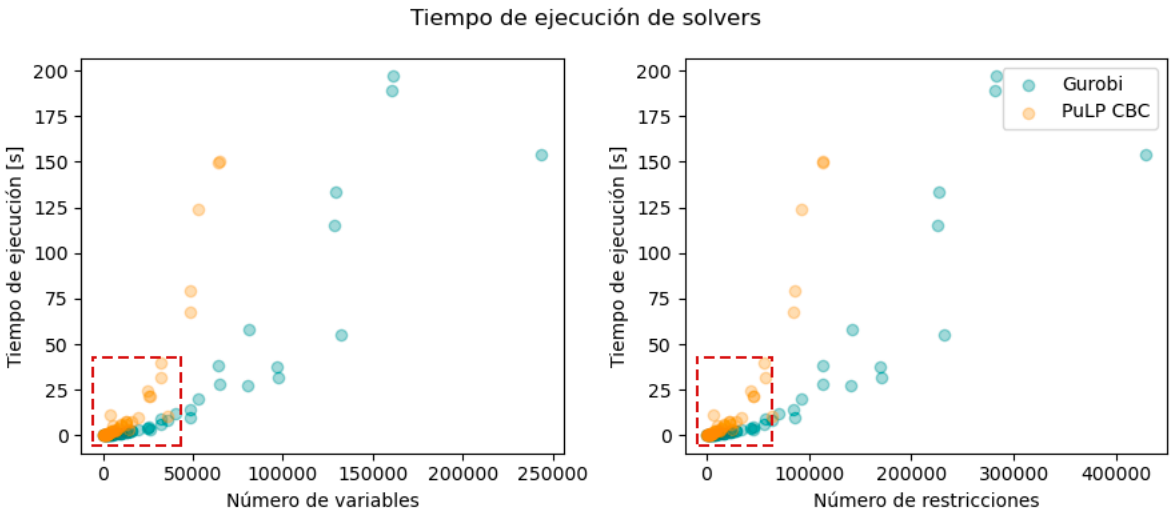

Figura 4.6: Relación entre tiempo de ejecución y cantidad de variables y restricciones en el modelo, con *solvers* Gurobi y PuLP CBC. La sección dentro del rectángulo punteado rojo se muestra en la Figura 4.7.

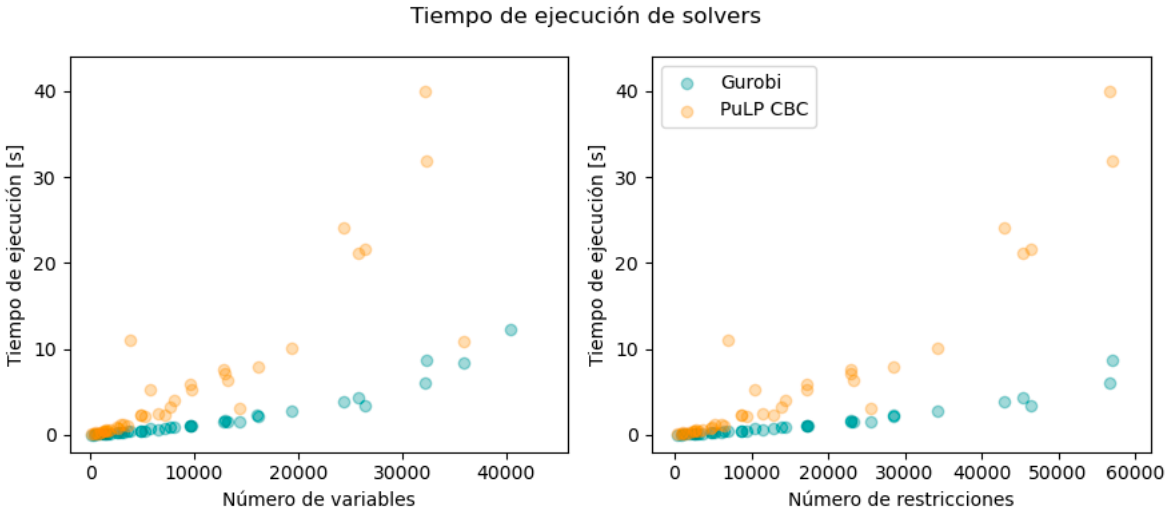

Figura 4.7: *Zoom* de Figura 4.6. Relación entre tiempo de ejecución y cantidad de variables y restricciones en el modelo, con *solvers* Gurobi y PuLP CBC.

4.4. Pruebas con entradas modificadas

4.4.1. Ruido en *forecast* de demanda

El modelo debe ser robusto a variaciones en los parámetros, por lo que se realiza un análisis de sensibilidad al error de esta predicción. El único parámetro que tiene un error considerable es el *forecast* de la demanda. Para poder evaluar la robustez se procede optimizando de la

misma manera que en la Sección 4.2.3, con la diferencia que, semana a semana, se corrigen las cantidades demandadas, de cada SKU en cada tienda, al añadirle un ruido al *forecast* de la demanda F. Esto permite evaluar métricas como número de quiebres, utilidades y cantidad de unidades, cuantificando la diferencia entre el caso sin error (visto en las Secciones anteriores), con otros casos donde la demanda real difiere del *forecast* utilizado.

El ruido se obtiene de manera aleatoria siguiendo una distribución normal, o densidad de probabilidad para la distribución Gaussiana:

$$p(x) = \frac{1}{\sqrt{2\pi\tilde{\sigma}^2}} e^{-\frac{(x-\mu)^2}{2\tilde{\sigma}^2}} \tag{4.13}$$

En este caso, x corresponde al *forecast* con error ($F_{ruido\ i,j,t}$). Se utilizan los valores $\mu = F_{i,j,t}$ y desviación estándar $\tilde{\sigma} = \mu * \sigma$. Las pruebas que se realizan en esta sección, corresponden a implementar el algoritmo variando el valor de σ, a mayor valor de σ se espera que $F_{ruido\ i,j,t}$ se aleje más del *forecast* utilizado. Cabe destacar que con $\sigma = 0$ se recupera el caso original sin ruido presentado en la Sección 4.2.3.

Tabla 4.8: Desempeño con ruido en forecast de demanda I = 2, J = 2, T = 8. Las columnas presentes son las mismas que se encuentra descritas de manera detallada en la Tabla 4.2

σ	**MAPE [%]**	**Utilidades [MCL$]**	**# de unidades**	**# de quiebres**
0.0	0	43.28	12148	4
0.2	13	43.47	12397	7
0.6	37	42.57	12585	8

Tabla 4.9: Desempeño con ruido en *forecast* de demanda I = 20, J = 20, T = 8. Los resultados en esta Tabla contempla sólo las semanas entre la 1 y la 13 los resultados desde la semana 14 a la 20 se desestiman (detalles en Sección 4.2.3).# de unidades corresponde a la cantidad de productos que satisfacen la demanda, dentro del modelo se representa con la variable $\tilde{Q}$, estos están en miles de unidades, para facilitar la lectura. # de quiebres corresponde a la cantidad de quiebres de *stock* detectados en los resultados entre las semanas 1 y 13. Las utilidades se presentan en billones de pesos chilenos (BCLP) y corresponde a la contribución por venta, esto es exactamente el término Z_1 de la función objetivo, detallada en la Sección 4.1.1.

σ	**MAPE [%]**	**Utilidades [BCL$]**	**# de unidades [miles]**	**# de quiebres**
0.0	0	5.10	1.30	67
0.1	8	5.07	1.29	224
0.2	16	5.01	1.27	389
0.3	24	4.94	1.25	514
0.4	32	4.85	1.23	636
0.5	40	4.79	1.21	726
0.6	47	4.73	1.20	756
0.7	54	4.70	1.19	846
0.8	61	4.70	1.17	855
0.9	67	4.71	1.19	858
1.0	72	4.70	1.18	914

Dadas las características de la distribución utilizada para modelar el error en el *forecast*, se tiene que la curva con error no tiene sesgo. Por lo anterior el error puede ser positivo o negativo con la misma probabilidad y para periodos muy grandes la suma total de los errores se espera que sea muy cercano a cero.

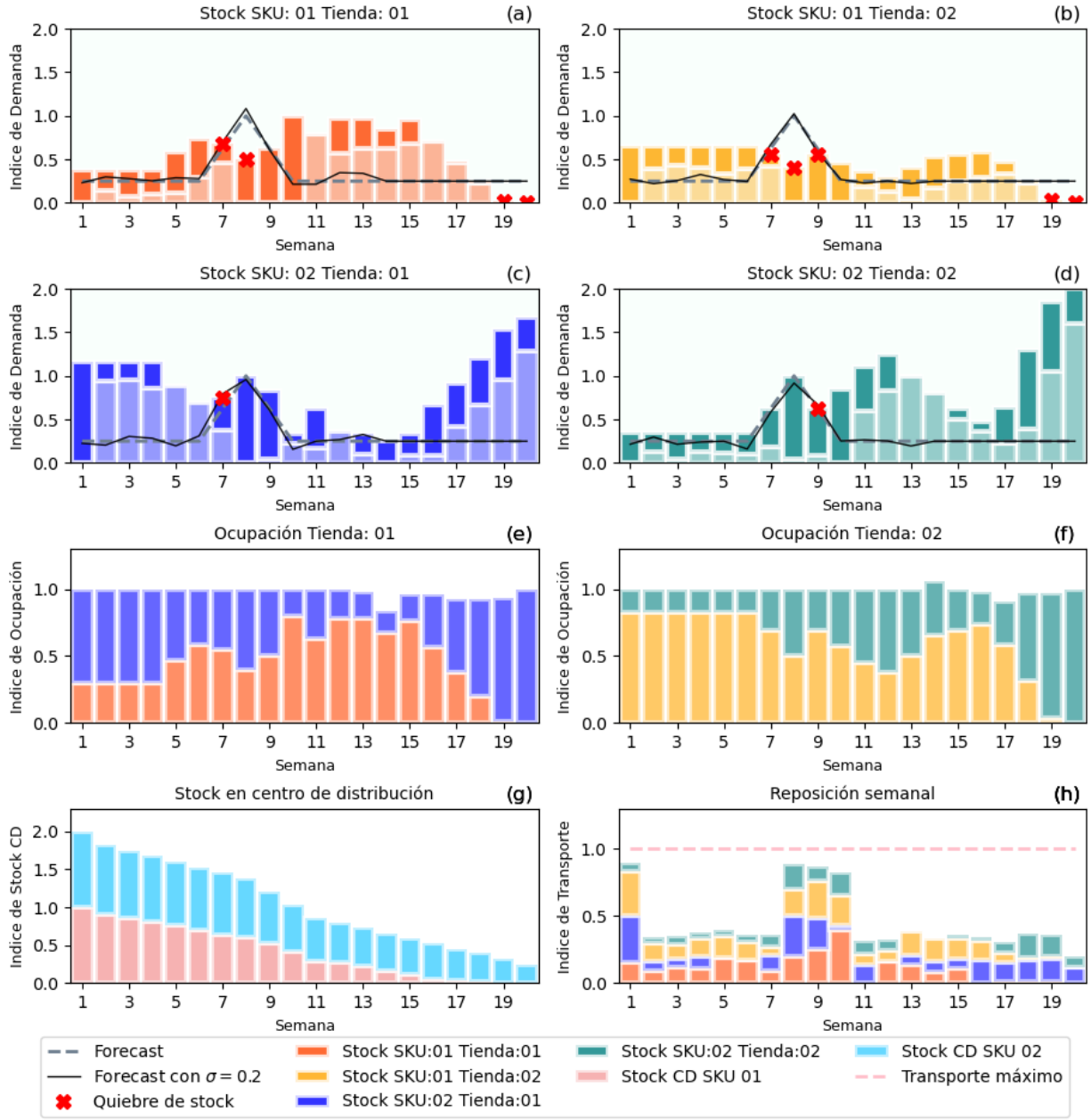

Figura 4.8: Resultados para inducción de ruido, $\sigma = 0.2$. Se utilizan dos SKUs y dos tiendas. Los parámetros distintos a σ son los mismos implementado en la Figura 4.4.

En las Figuras 4.8 y 4.9 se ve en los gráficos (a), (b), (c) y (d) que la curva con ruido se separa del *forecast* original, lo que genera que las barras de *stock* de los SKUs, en cada tienda, no sean iguales para ciertas semanas, como pasa en el caso sin ruido de la Figura 4.4. Se destaca de la Tabla 4.9 como aumenta la cantidad de quiebres al aumentar el error del

forecast, aumentando cerca de 14 veces al pasar de un caso sin error a otro con un MAPE del 72 %.

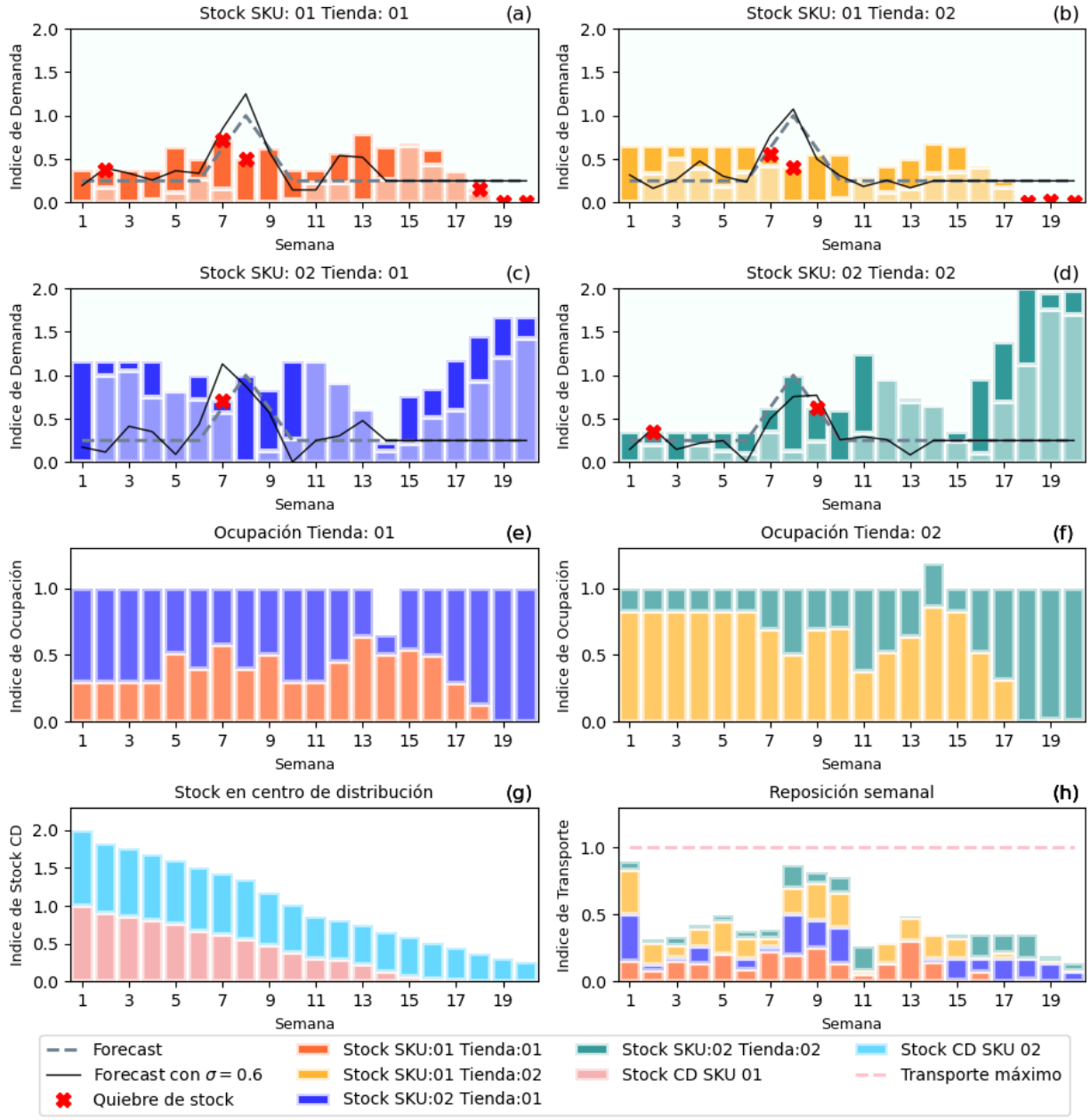

Figura 4.9: Resultados para inducción de ruido, $\sigma = 0.6$. Se utilizan veinte SKUs y veinte tiendas. Los parámetros distintos a σ son los mismos implementado en la Figura 4.4.

4.4.2. Limitantes de capacidad

Hasta ahora se han realizado con simulaciones manteniendo fijos los parámetros de capacidad máxima en tiendas y capacidad máxima de transporte, para facilitar el análisis. Sin embargo, es necesario evaluar el comportamiento del modelo cuando estos parámetros son distintos. Por lo anterior, en esta Sección se evalúan seis casos diferentes, destacando los

resultados más importantes de cada uno. Para presentar los resultados se utilizan Figuras, y la Tabla 4.10, como resumen de la cantidad de quiebres. Estas implementaciones consideran dos tiendas y dos SKUs, lo que permite un análisis visual mediante gráficos.

Se llama caso normal a la implementación de la Sección 4.2.3 donde se tiene una cantidad total de quiebres igual a 4 (1 en la tienda 01 y 3 en la tienda 02). Se destaca de esa implementación los gráficos (e) y (f) de la Figura 4.4 donde se presenta una ocupación completa de las tiendas.

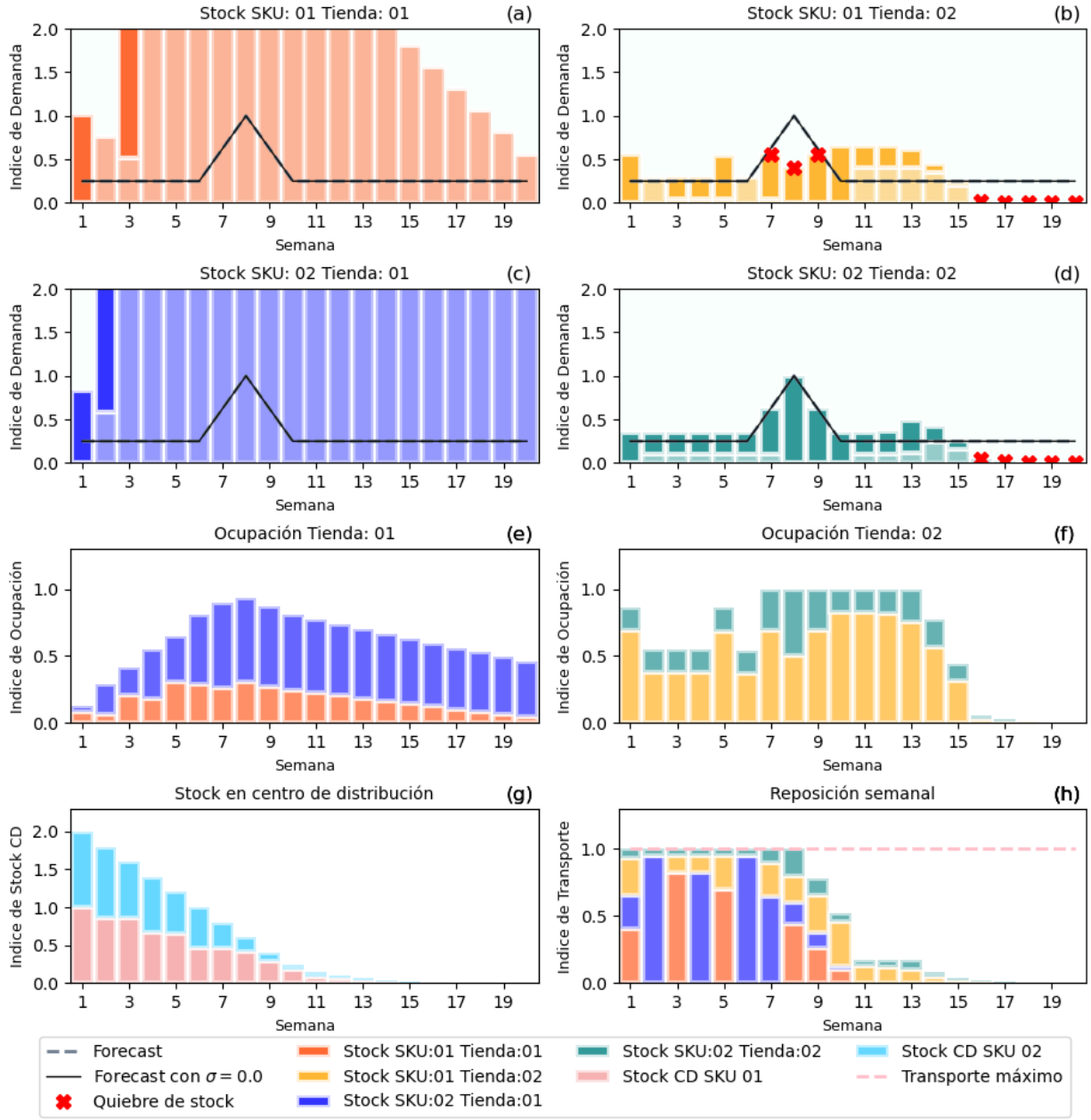

Figura 4.10: Se aumenta el límite de capacidad en tienda 01 para almacenar productos. Los parámetros utilizados son los mismos de la Figura 4.4, modificando sólo $B_{1,t}$.

Como primera modificación a parámetros, con respecto a la Sección 4.2.3, se aumenta la

capacidad de almacenaje en diez veces en la tienda 01. El aumento es considerablemente alto con el fin generar un gran contraste con respecto al caso normal y facilitar la visualización de cambios. La Figura 4.10 muestra los resultados, en donde se ve que el índice de transporte se mantiene en 1 en las primeras semanas hasta que se agota el *stock* en el centro de distribución. La demanda en esa tienda se cumple con creces lo que deja quiebres de *stock* sólo en la tienda 02 (3).

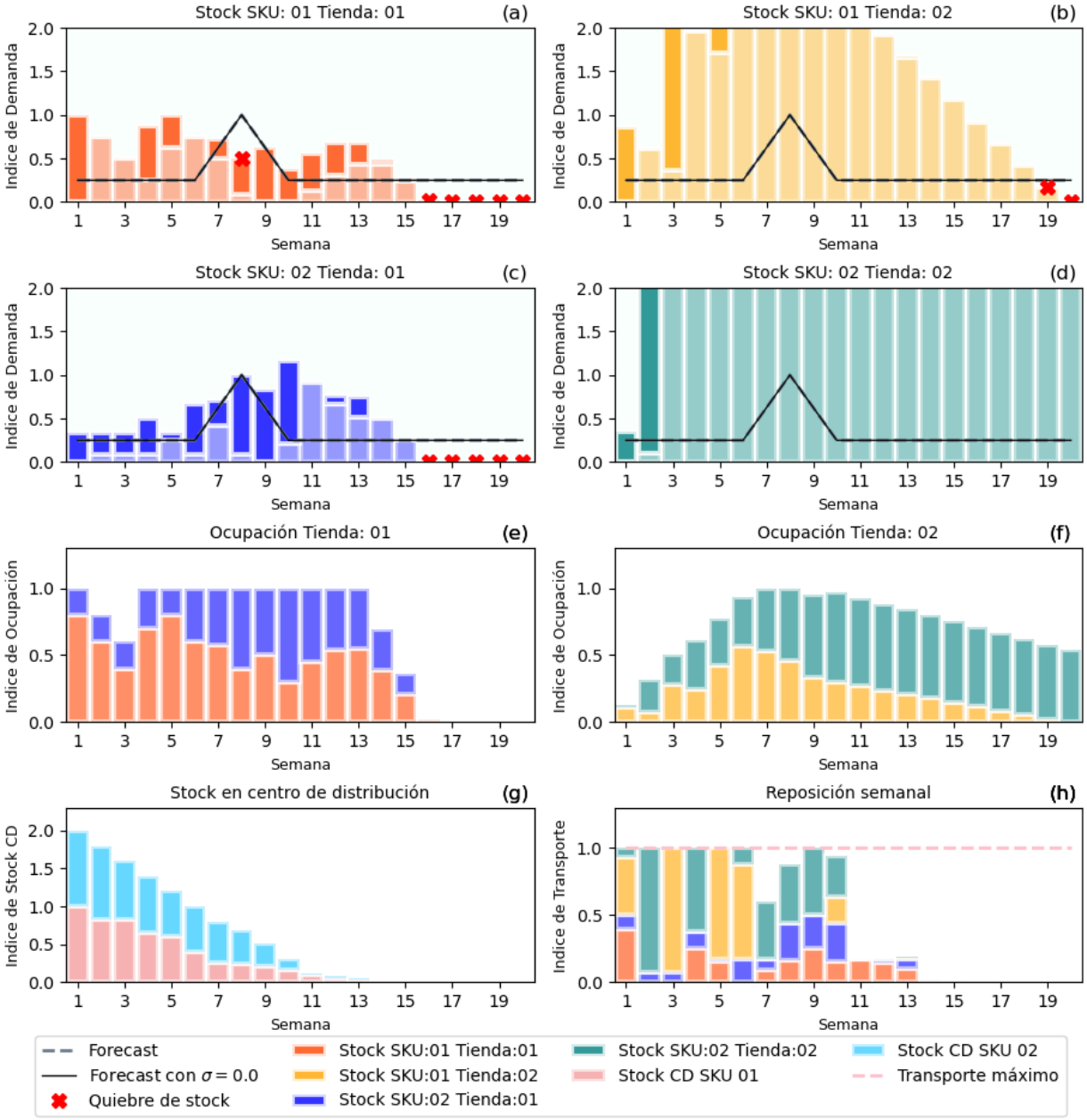

Figura 4.11: Se aumenta el límite de capacidad en tienda 02 para almacenar productos. Los parámetros utilizados son los mismos de la Figura 4.4, modificando sólo $B_{2,t}$.

En la Figura 4.11 se presenta el caso en que la capacidad de la tienda 02 aumenta diez veces, y la capacidad de la tienda 01 vuelve a la original. La visualización es similar a la Figura

4.10, siendo ahora la tienda 02 la que satisface toda la demanda. El índice de transporte es 1 hasta que el índice de ocupación en la tienda 02 se acerca a 1. Nuevamente, decae rápido el *stock* de ambos SKUs en el centro de distribución.

En la Figura 4.12 se aumentan tanto el límite de capacidad de la tienda 01, como el límite de la tienda 02, en diez veces el valor original ($B_{1,t}$ y $B_{2,t}$). En las primeras cuatro semanas, el SKU 02 tiene un bajo *stock* en ambas tiendas. El índice de transporte está al máximo y el *stock* en el CD se agota en la semana 11.

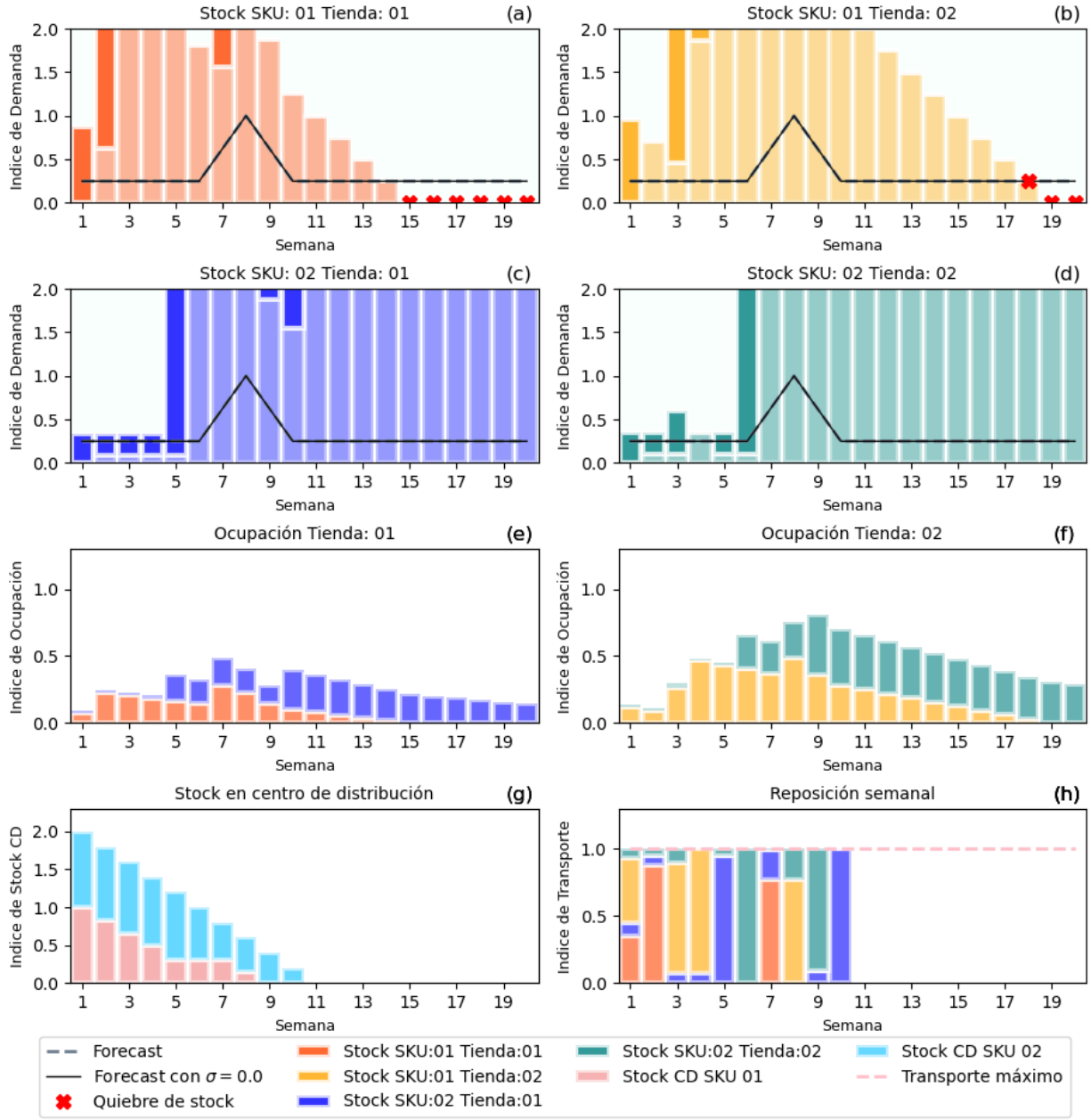

Figura 4.12: Se aumenta el límite de capacidad en tiendas 01 y 02 para almacenar productos. Los parámetros utilizados son los mismos de la Figura 4.4, aumentando tanto $B_{1,t}$ como $B_{2,t}$.

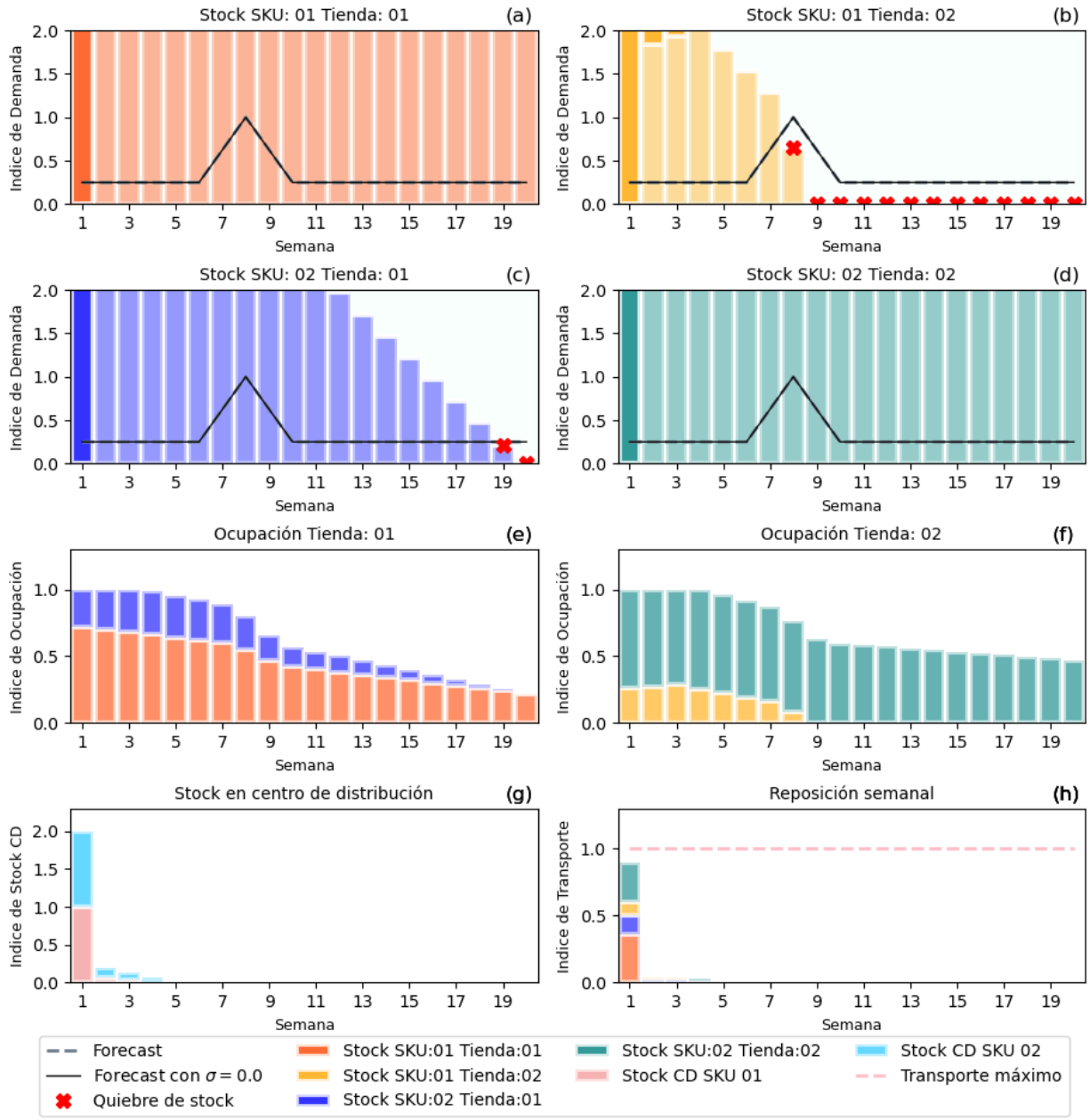

Figura 4.13: Se aumenta el límite de capacidad en tiendas 01 y 02 para almacenar productos, más aumento de límite de transporte. Los parámetros utilizados son los mismos de la Figura 4.4, aumentando $B_{1,t}$, $B_{2,t}$ y Tr_t.

En la Figura 4.13 se presenta un caso ideal, aumentando la capacidad en las tiendas 01 y 02 como también la capacidad de transporte, todo en un factor 10. En este escenario, todo el *stock* en el CD se puede almacenar en las tiendas, lo que provoca que siempre se pueda satisfacer la demanda, que está muy por debajo del *stock* en tiendas. Sólo se genera falta de *stock* cuando se acaba el SKU 01 en la tienda 2, produciendo quiebres de la semana 8 a la semana 13 en esa tienda.

Para terminar, en la Figura 4.14 se utilizan los parámetros originales de la Sección 4.2.3, pero ahora se modifica sólo la capacidad de transporte semanal, la cual se disminuye en un

70 %. El resultado de esto es una gran aumento en el número de quiebres, especialmente en el SKU 01, donde para la tienda 01 nunca se satisface la demanda. Cabe destacar, que el índice de transporte en este caso es 1.

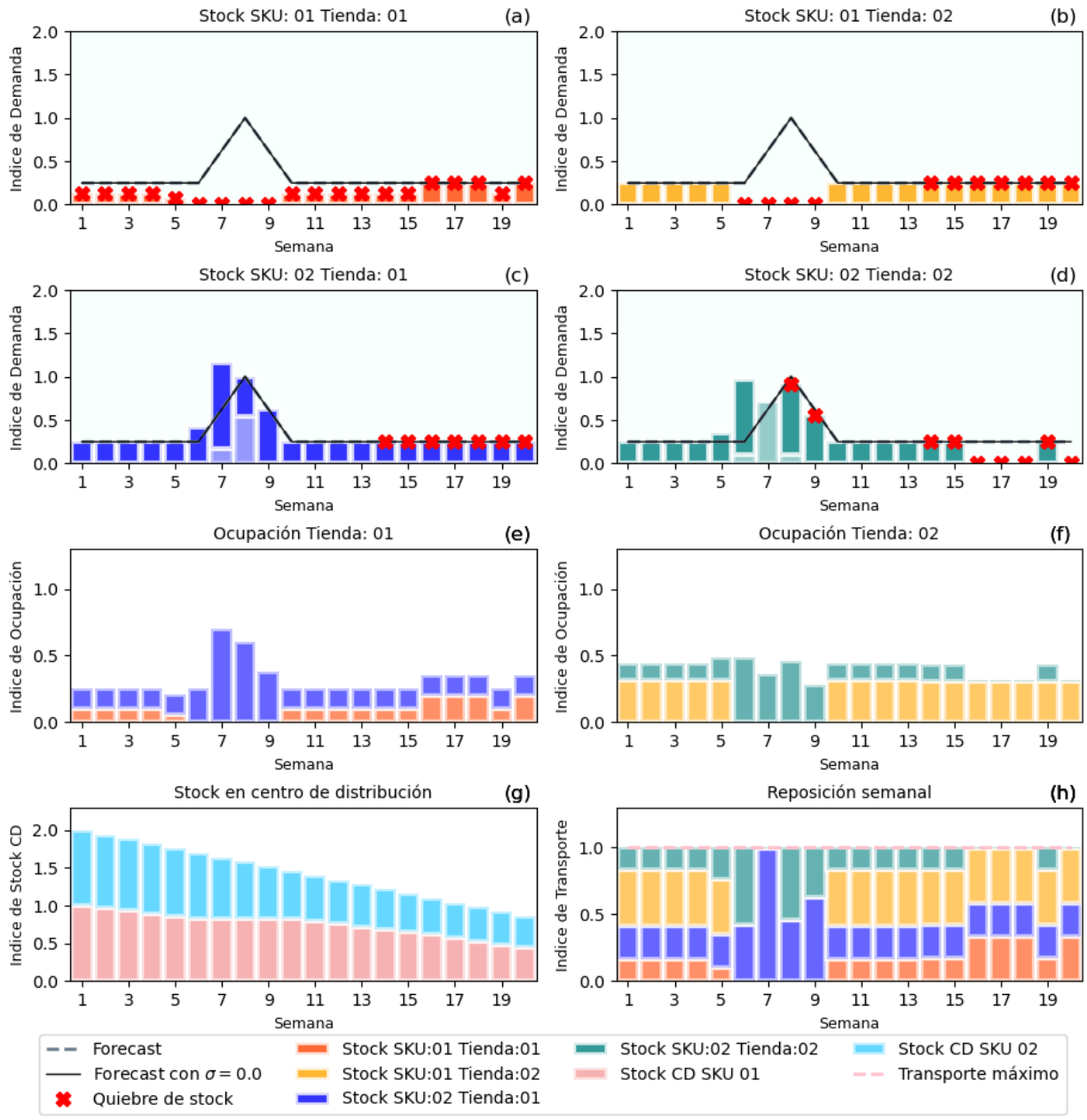

Figura 4.14: Se disminuye en un 70 % el límite de transporte Tr_t, los demás parámetros son los mismos utilizados en la Figura 4.4.

	# de quiebres		
	en tienda 01	**en tienda 02**	**Total**
Caso normal	1	3	4
Aumento en $B_{1,t}$	0	3	3
Aumento en $B_{2,t}$	1	0	1
Aumento en $B_{1,t}$ y $B_{1,t}$	0	0	0
Aumento en $B_{1,t}$, $B_{1,t}$ y Tr_t	0	6	6
Disminución en Tr_t	13	6	19

Tabla 4.10: Resumen de quiebres de *stock*, cambiando límites de transporte y capacidad en tiendas.

Capítulo 5

Análisis y discusión

5.1. Tamaño de ventana temporal

La implementación con mejores resultados corresponde a la que contempla una ventana temporal mayor, en este caso de ocho semanas. Los resultados obtenidos en las Secciones 4.2.3 y 4.2.4 muestran que se obtienen mayores utilidades, y una mayor cantidad de productos que satisfacen la demanda, para el caso de ocho semanas. El caso *Naive* muestra menores tiempos de ejecución, lo que es deseable, pero no corresponde a los objetivos principales de este trabajo. Lo que se busca primordialmente es obtener un buen calendario de reposición, que minimice los quiebres de *stock* y genere el mayor beneficio. El beneficio puede ser económico o en volumen de ventas (cantidad).

La implementación *Naive* no obtiene el óptimo porque no distribuye a tiempo los productos en las tiendas de manera de satisfacer la demanda futura. Al tener una limitación de la cantidad de productos en las tiendas, la implementación de ocho semanas de ventana temporal (Sección 4.2.3) también presenta quiebres, sin embargo se espera que con una capacidad disponible mayor en las tiendas, se pueda tener el inventario necesario para satisfacer toda la demanda. Evidencia de esto se tiene al modificar los límites de capacidad de almacenaje en tiendas (Sección 4.2.3), donde la Tabla 4.10 muestra que aumentando estos límites no se observan quiebres de *stock*.

5.2. Términos en función objetivo

Implementar distintos términos en la función objetivo permite encaminar la solución hacia un calendario de reposición más atractivo para los encargados de la planificación y para la compañía. Un ejemplo de esto es la capacidad del modelo de generar soluciones que prioricen satisfacer la demanda en las semanas iniciales, lo que se ve a lo largo de todas la Figuras de la Sección 4.

Las Tablas 4.3 y 4.4 muestran cómo se mueve la solución generando distintos valores de

utilidades y cantidad de unidades de productos que satisfacen la demanda, a partir de distintas configuraciones de términos en la función objetivo. Se ve entonces, la función objetivo, como una herramienta muy valiosa que da flexibilidad al modelo y permite, a la tienda de *retail* en cuestión, personalizar la reposición dependiendo de las necesidades que se tengan.

Es especialmente notable que las mayores utilidades no se generan al incluir sólo el término Z_1 (termino que prioriza las utilidades) sino que entrega mayores utilidades la combinación de éste junto con otros términos, implicando que es más beneficioso a largo plazo distribuir los productos de mejor manera, que priorizar las ventas de productos con mayor margen de utilidades en el corto plazo. Esta información es entregada por la Tabla 4.4.

La desviación estándar de las utilidades es baja comparada con el promedio (Tabla 4.4), lo que permite elegir la combinación términos que más convengan al planificador, sin sacrificar gran porcentaje de utilidades.

5.3. Complejidad del modelo y *solvers*

La Tabla 4.5 muestra que el número de restricciones y variables dependen del producto de la cantidad de tiendas y SKUs que se consideren en la optimización, lo que hace que crezcan rápidamente al aumentar estos números, siendo comprobado en la Figura 4.6. Si bien la cantidad de tiendas a considerar y ventana de semanas tienen una cota máxima bastante baja (50 tiendas y 8 semanas), la cantidad de SKUs que manejan los *retailers* es muy elevada, lo que sobrecarga al modelo con variables y restricciones.

Cabe destacar que tanto la cantidad de variables, como la cantidad de restricciones, son lineales con respecto a la cantidad de SKUs (I), la cantidad de tiendas (J) y la cantidad de semanas por ventana de optimización (T). A modo de ejemplo se puede fijar el número de semanas en ocho ($T = 8$), que corresponde a reponer dos meses por adelantado, y se puede fijar la cantidad de tiendas en cincuenta, que corresponde a la cantidad máxima de tiendas de la compañía seleccionada ($J = 50$). Siguiendo las expresiones de la Tabla 4.5 se llega a:

$$\text{Cantidad de variables} = 1,608\ I \tag{5.1}$$

$$\text{Cantidad de Restricciones} = 2,816\ I + 408 \tag{5.2}$$

Queda explícito en las ecuaciones 5.1 y 5.2 que las cantidades descritas son lineales con respecto al número total de SKUs. Al observar las Figuras 4.6 y 4.7, se puede notar que el tiempo de ejecución no es lineal con respecto a la cantidad de variables y, por todo lo dicho anteriormente, tampoco es lineal con respecto a la cantidad de SKUs. Con esto se deduce que aumentar la cantidad de SKUs en el modelo genera cada vez un crecimiento mayor en los tiempos de ejecución, la pendiente va aumentando.

Los *solvers* utilizados para la optimización fueron adecuados, encontrando la solución exacta para cada simulación. Las diferencias se presentan al comparar los tiempos de ejecución, donde Gurobi es notablemente más rápido para entregar la solución. Para 8,080 variables PuLP CBC demora cuatro veces más del tiempo de ejecución que demora Gurobi, mientras que para 52,800 variables el tiempo para PuLP CBC es 6 veces mayor aproximadamente.

Para ambos *solvers* los tiempos de ejecución crecen más rápido que de manera lineal, con respecto a la cantidad de variables.

5.4. Parámetros

Mediante los resultados de la Sección 4.4.2 se entiende que los límites logísticos son capaces de moldear la reposición. Este hecho corresponde a una herramienta útil para los planificadores, dado que se pueden mantener, por ejemplo, los valores de B más bajos del máximo, reservando espacio para futuras alzas el la demanda. En caso de ser necesario, se puede distribuir los productos de la manera óptima al acercarse a fechas críticas.

El único parámetro que no es un dato fijo es el *forecast* de demanda. Esto es porque la predicción de demanda tiene asociado un error. Los resultados muestran que errar en la predicción de la demanda genera mayores quiebres de *stock* y menores utilidades, estas diferencias depende del MAPE que se tenga. Para un MAPE del 16 % las perdidas de utilidades son del orden del 1.8 %, mientras que para un MAPE del 40 % la disminución de utilidades es de un 6.1 %.

5.5. Limitantes logísticas

Las Figuras de la Sección 4.4.2 (Figuras 4.10 - 4.14) muestran reposiciones muy distintas dependiendo del escenario que se tenga. Las distintas combinaciones de valores para los límites logísticos influyen notoriamente en la solución entregada por el *solver* lineal. Por lo anterior, se encuentran justificadas las Restricciones 4.1 y 4.7 definidas en la etapa de modelado, son restricciones necesarias para moldear la solución de manera favorable.

Capítulo 6

Conclusiones

Se define matemáticamente un modelo de optimización lineal que guarda estrecha relación con el problema real de optimización de reposición de productos en el *retail*. El modelo presenta una función objetivo lineal lo suficientemente flexible para cumplir con las necesidades de la empresa. Esta función contempla cuatro términos distintos que, mediante una combinación de estos, se pueden obtener distintos calendarios de reposición, dependiendo de lo que se quiera optimizar. Los términos están orientados a maximizar las utilidades, maximizar la cantidad de ventas, vender lo antes posible y disminuir los quiebres de *stock*. Al incluir sólo el término de beneficio (Z_1) en la función objetivo no se obtiene el mejor resultado, cuando además se añade a la función objetivo el cuarto término (quedando la función objetivo como $Z_1 + Z_4$) las utilidades aumentan en 6 MMCLP, aumenta la demanda satisfecha en 1,649 unidades y disminuye la cantidad de quiebres de *stock* de 18 a 15.

De la misma manera, las restricciones modeladas son reflejo de restricciones logísticas reales de empresas de *retail*, contemplando las capacidades de almacenaje en tiendas y centro de distribución, como también la capacidad logística de transporte. Los resultados muestran que las restricciones logísticas afectan la reposición, lo que se ve reflejado en cantidad de quiebres de *stock*, por lo que es importante incluirlas. Para dos tiendas y dos SKUs la cantidad de quiebres de *stock* aumenta de 4 a 19 cuando se tiene poca capacidad de transporte.

Se realiza una implementación exitosa de los de algoritmos que permiten utilizar *solvers* lineales para realizar la optimización, realizado mediante *scripts* en Python. Se realizan variadas simulaciones que permiten evaluar el modelo definido con anterioridad. En este sentido, se implementaron los *solvers* Gurobi y PuLP CBC. Las simulaciones contemplan reposiciones semanales, y buscan anticiparte a aumentos futuros en la demanda, para reponer proactivamente y evitar quiebres de *stock*. La implementación es exitosa en este sentido, donde las simulaciones con mayores ventanas temporales generan mejores reposiciones. Al aumentar la ventana temporal de optimización de 1 a 8 semanas se ve un aumento en las utilidades de un 2.6 %, un aumento de 1,649 unidades en la demanda satisfecha y una disminución de 6 a 4 quiebres de *stock*.

Las métricas implementadas entregan información del desempeño de las simulaciones, permitiendo un análisis directo. Estas métricas son tiempos de ejecución de los algoritmos, cantidad de quiebres de *stock* y demanda satisfecha (en cantidad de unidades). Las visualiza-

ciones implementadas aportan información muy valiosa para evaluar, de manera cualitativa, el desempeño del modelo, permitiendo observar las reposiciones e inventarios cuando se utilizan pocos SKUs y tiendas en las simulaciones. Entregan también de manera visual dónde, y cuándo, se producen quiebres de *stock*.

Para terminar, se destaca que todos los objetivos específicos del trabajo fueron cubiertos de manera exitosa, obteniendo un modelo matemático de optimización lineal de enteros para la reposición de productos en el *retail*. Se logra la solución de este modelo mediante *solvers* lineales existentes, y la implementación de métricas, en Python, que permiten evaluar cada una de las simulaciones realizadas con los *solvers*.

6.1. Contribución

Este trabajo contribuye a la automatización y centralización de procesos dentro de la empresa. Es un primer paso en generar herramientas computacionales útiles para la compañía que sean flexibles, para ajustarse a sus necesidades. Apunta también en la dirección de disminuir los empujes manuales realizados por *planners*, cuando se requiere reponer en fechas de alta demanda.

Se destaca la oportunidad generada para probar distintas herramientas computacionales, como por ejemplo el *solver* lineal de Gurobi, y evaluar su desempeño. Es especialmente útil para que futuros proyectos tengan antecedentes de la utilización de estas herramientas.

6.2. Trabajo futuro

Por ahora el trabajo contempla simulaciones que no trabajan con datos reales, por lo que se propone buscar un acople entre datos que se manejan en la empresa con este modelo implementado en simulaciones ideales. Esto está fuera de los alcances de este proyecto, pero es el paso siguiente para generar una herramienta útil.

Los resultados de la evaluación de complejidad de las simulaciones arrojan que la implementación es costosa en tiempo, cantidad de variables y restricciones utilizadas. Es por esto que se propone avanzar en tres direcciones para lograr menores tiempos de ejecución: modificar los *scripts* en Python utilizados para que el código permita una solución más rápida de los *solvers*. Por otro lado se propone investigar modificaciones al modelo que permitan tener un menor número de restricciones y variables. Por último, se pueden utilizar métodos heurísticos para obtener soluciones aproximadas (como lo hacen los trabajos señalados en la Sección 2.2), remplazando los métodos exactos utilizados por los *solvers* lineales. Se propone realizar una agrupación en tiendas, o SKUs, que disminuya la cantidad de variables a utilizar.

Se propone avanzar en caracterizar el error real del *forecast* y replicar las simulaciones generando un error que tenga cierto sesgo, con lo que al sumarlo en el tiempo no sea aproximadamente cero. También, trabajar con un *forecast* estocástico y generar los escenarios de optimización al trabajar con los intervalos de confianza del *forecast*.

Por ahora se sabe que el tiempo de ejecución tiene un crecimiento más rápido que uno lineal con respecto a la cantidad de variables, sin embargo este crecimiento no está cuantificado. Se propone cuantificar esta relación, realizando simulaciones con mayor cantidad de variables que las que se presentan en este trabajo realizando ajustes polinomiales y exponenciales.

Bibliografía

[1] Narendra Agrawal and Stephen Smith. *Retail Supply Chain Management: Quantitative Models and Empirical Studies*, volume 122. , 01 2009.

[2] Narendra Agrawal and Stephen Smith. Optimal inventory management for a retail chain with diverse store demands. *European Journal of Operational Research*, 225:393–403, 03 2013.

[3] Gerardo Berbeglia and Gwenaël Joret. Assortment optimisation under a general discrete choice model: A tight analysis of revenue-ordered assortments. *CoRR*, abs/1606.01371, 2016.

[4] Bertsimas, Dimitris and Tsitsiklis, John. *Introduction to Linear Optimization*. Athena Scientific, 1998.

[5] Jens Clausen. *Branch and Bound Algorithms-Principles and Examples*. , 2003.

[6] Clerc M. Standard Particle Swarm Optimisation. [Online]. Available in: `https://hal.archives-ouvertes.fr/file/index/docid/764996/filename/SPSO_descriptions.pdf`.

[7] Cococcioni, Marco and Fiaschi, Lorenzo. The Big-M method with the numerical infinite M. *Optimization Letters*, 2021. Available in: `https://doi.org/10.1007/s11590-020-01644-6`.

[8] Computational Infrastructure for Operations Research. COIN-OR Branch-and-Cut solver. [Online]. Available in: `https://github.com/coin-or/Cbc`.

[9] Computational Infrastructure for Operations Research. COIN-OR PuLP. [Online]. Available in: `https://github.com/coin-or/pulp`.

[10] Computational Infrastructure for Operations Research. Documentación Biblioteca PuLP. [Online]. Available in: `https://pypi.org/project/PuLP/`.

[11] George B. Dantzig. Linear programming and extensions, 1963.

[12] Davis, James M. and Gallego, Guillermo and Topaloglu, Huseyin. Assortment Optimization Under Variants of the Nested Logit Model. *Operations Research*, 2014. Available in: `https://doi.org/10.1287/opre.2014.1256`.

[13] Antoine Desir, Vineet Goyal, Srikanth Jagabathula, and Danny Segev. Assortment optimization under the mallows model. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[14] Jacques Desrosiers and Marco Lübbecke. *Branch-Price-and-Cut Algorithms*, chapter . , 01 2011.

[15] Désir, Antoine and Goyal, Vineet and Jagabathula, Srikanth and Segev, Danny. Mallows-Smoothed Distribution over Rankings Approach for Modeling Choice. *Operations Research*, 2017. Available in: `https://doi.org/10.1287/opre.2020.2085`.

[16] Efficient Costumer Response Community. Optimal shelf availability. [Online], 2003. Available in: `https://ecr-community.org/wp-content/uploads/2016/10/ecr-europe-osa-optimal-shelf-availability.pdf`.

[17] Marshall Fisher and Ramnath Vaidyanathan. A demand estimation procedure for retail assortment optimization with results from implementations. *Management Science*, 60:2401–2415, 10 2014.

[18] Armin Fügenschuh and Alexander Martin. *Computational Integer Programming and Cutting Planes*, volume 12 of *Handbooks in Operations Research and Management Science*. Elsevier, 2005.

[19] Gurobi Optimization. Documentation website. [Online]. Available in: `https://www.gurobi.com/documentation/`.

[20] Gurobi Optimization. Indicator constraints, documentación de Model.addGenConstrIndicator(). [Online]. Available in: `hhttps://www.gurobi.com/documentation/9.5/refman/py_model_agc_indicator.html`.

[21] IBM. ILOG CPLEX Optimization Studio. [Online]. Available in: `https://www.ibm.com/cl-es/products/ilog-cplex-optimization-studio`.

[22] Igor Griva, Stephen G. Nash, Ariela Sofer. *Linear and nonlinear optimization*. Society for Industrial and Applied Mathematics, 2009.

[23] Bernhard Korte and Jens Vygen. *The Traveling Salesman Problem*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[24] E.C. Laskari, K.E. Parsopoulos, and M.N. Vrahatis. Particle swarm optimization for integer programming. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1582–1587 vol.2, 2002.

[25] Kun Li and Huixin Tian. An improved particle swarm optimization for selective single machine scheduling with sequence dependent setup costs and downstream demands. *Mathematical Problems in Engineering*, 2015:1–11, 07 2015.

[26] Shaohui Ma and Robert Fildes. A retail store sku promotions optimization model for category multi-period profit maximization. *European Journal of Operational Research*, 260(2):680–692, 2017.

[27] Mecanex. Sobrestock: estrategias para evitarlo. [Online]. Available in: https://www.mecalux.es/blog/sobrestock.

[28] John E. Mitchell. Branch-and-Cut Algorithms for Combinatorial Optimization Problems, 1988.

[29] Samuel Nucamendi and Miguel Angel Moreno Miguez. Simple and efficient tool for reduction in total inventory for mexican fashion retail industries, 03 2016.

[30] Numpy. Documentación random.randint. [Online]. Available in: https://numpy.org/doc/stable/reference/random/generated/numpy.random.randint.html.

[31] Numpy. Documentación random.seed . [Online]. Available in: https://numpy.org/doc/stable/reference/random/generated/numpy.random.seed.html.

[32] OBS Business School. Sobrestock: cinco problemas que debes gestionar. [Online]. Available in: https://www.obsbusiness.school/blog/sobrestock-cinco-problemas-que-debes-gestionar.

[33] Oracle Netsuit. Retail Inventory Management: What It Is, Steps, Practices and Tips. [Online], 09 2020. Available in: https://www.netsuite.com/portal/resource/articles/inventory-management/retail-inventory-management.shtml.

[34] Pablo A. Barberis A. Quiebres de stocks y su impacto en la demanda. [Online], 2012. Available in: http://www.emb.cl/negociosglobales/articulo.mvc?xid=191.

[35] Florian A. Potra and Stephen J. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1):281–302, 2000. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

[36] PWC. Retail inventory management: An intricate balancing act. [Online], 02 2015. Available in: https://www.pwc.ru/en/retail-consumer/publications/assets/retail-inventory-management.pdf.

[37] Rooderkerk, Robert P. and van Heerde, Harald J. and Bijmolt, Tammo H. A. Optimizing Retail Assortments. *Marketing Science*, 2013. Available in: https://doi.org/10.1287/mksc.2013.0800.

[38] Lidia Sanchez-Ruiz, Beatriz Blanco, and Asta Kyguoliene. A theoretical overview of the stockout problem in retail: from causes to consequences. *Management of Organizations: Systematic Research*, 79:103–116, 06 2018.

[39] Savi, Marcelo A. Jamian, J. J. Abdullah, M. N. Mokhlis, H. Mustafa, M. W. Bakar, A. H. A. Global Particle Swarm Optimization for High Dimension Numerical Functions Analysis. *Journal of Applied Mathematics*, 2014. Available in: https://doi.org/10.1155/2014/329193.

[40] Slimstock. Los quiebres de stock en el retail. [Online]. Available in: https://www.slimstock.com/cl/los-quiebres-de-stock-en-retail/.

[41] Slimstock. Soluciones innovadoras compatibles con IA para controlar tu cadena de suministro. [Online]. Available in: https://www.slimstock.com/cl/soluciones/.

[42] Oğuz Solyalı and Haldun Süral. The one-warehouse multi-retailer problem: Reformulation, classification, and computational results. *Annals of Operations Research*, 196:1–25, 01 2009.

[43] Strang, Gilbert. Karmarkar's algorithm and its place in applied mathematics. *The Mathematical Intelligencer*, 1987. Available in: https://doi.org/10.1007/BF03025891.

[44] Tamás Terlaky. *Interior Point Methods of Mathematical Programming*. Springer, Boston, MA, 2013.

[45] Telematel. Consejos para evitar el sobrestock en tu distribuidora de materiales. [Online]. Available in: https://www.telematel.com/blog/sobrestock-exceso-de-stock-telematel/.

[46] Towards data science. Linear programming and discrete optimization with Python using PuLP. [Online]. Available in: https://towardsdatascience.com/linear-programming-and-discrete-optimization-with-python-using-pulp-449f3c5f6e99.

[47] Universidad ESAN. ¡Cuidado con el quiebre del stock! [Online]. Available in: https://www.esan.edu.pe/apuntes-empresariales/2016/03/cuidado-con-el-quiebre-del-stock/.

[48] Wolsey, L.A. *Integer Programming*. John Wiley & Sons, 1998.

[49] S.J. Wright. *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics, 1997.

[50] Wen Yang, Felix Chan, and Vikas Kumar. Optimizing replenishment polices using genetic algorithm for single-warehouse multi-retailer system. *Expert Syst. Appl.*, 39:3081–3086, 02 2012.

[51] Zhu, Zhi-yu Liu, Chunmei Song, Yinglei Rana, Mohammad N.A. Sun, Jinhong Li, Guoliang Perera, Ricardo. Optimizing High-Dimensional Functions with an Efficient Particle Swarm Optimization Algorithm. *Mathematical Problems in Engineering*, 2020. Available in: https://doi.org/10.1155/2020/5264547.