Código	FPI-009
Objeto	Guía de elaboración de Informe final de proyecto
Usuario	Director de proyecto de investigación
Autor	Secretaría de Ciencia y Tecnología de la UNLaM
Versión	5
Vigencia	03/9/2019

Departamento:
Departamento de Ingeniería e Investigaciones Tecnológicas

Programa de acreditación:
PROINCE

Código del Proyecto:
C240

Título del proyecto
Minería de datos y simulación sobre un sistema de stock y transporte

Director: Santa María, Cristóbal Raúl

Codirector: López, Luis

Integrantes:
Bosio, Agustín
Nuñez, Héctor

Alumnos de grado: (Aclarar si tiene Beca UNLaM/CIN)
Bazán, Mariana

Resolución Rectoral de acreditación:
N° 404/21

Fecha de inicio: 01/01/2021

Fecha de finalización: 31/12/2022

A. Desarrollo del proyecto (adjuntar el protocolo)

A.1. Grado de ejecución de los objetivos inicialmente planteados, modificaciones o ampliaciones u obstáculos encontrados para su realización (desarrolle en no más de dos (2) páginas)

El modelo desarrollado permite en primer término recomendar, a partir de la explotación de los datos realizada una forma de registro de operaciones que permita evaluar, programar y controlar, en forma más automatizada, la actividad de la empresa en función de satisfacer la demanda con mínimo costo. Tal cual muestran los resultados alcanzados, se requiere ajustar el perfil estadístico de la demanda y de la recuperación de pallets para validar los parámetros y calibrar el modelo de acuerdo a la operatoria real observada. La estructura elegida facilita incorporar la variación de todos los costos reemplazando valores en sus planillas de entradas. De igual manera pueden considerarse diferentes situaciones de pérdida o necesidad de reparación de material. La capacidad de simular la operatoria para siguientes períodos permite predecir comportamientos e identificar problemas para tomar decisiones adecuadas. En particular la posibilidad de observar mensualmente el stock en cada paletera según el esquema de demandas y recuperación de material en cada lapso temporal facilita las decisiones en pos de cubrir los "cuellos de botella" que dejan insatisfecha la demanda aun cuando se dispone de stock en forma global.

La continuidad del trabajo debiera orientarse a considerar el registro completo de la actividad por tipo de pallets, cosa que las estadísticas tomadas hasta aquí en realidad no permiten, para facilitar un análisis más fino de la demanda, del stock y de los costos. Por otra parte, con datos obtenidos con esa precisión cobraría mayor sentido obtener la simulación sobre el comportamiento observado de la demanda y la recuperación. También sería de mayor relieve, en ese caso, programar una forma automática de compensación de los faltantes de stock, allí donde los hubiere, que tuviera un mínimo costo. Estas son entonces las direcciones en las que debe proseguir el desarrollo del modelo.

En líneas generales se considera haber cumplido con los objetivos fijados para el proyecto pues se ha desarrollado el modelo pretendido. Por un lado, una vez estudiada la demanda individual por cliente se juzgó necesario integrarla en una única demanda mensual por planta o paletera. Se estableció un modelo porcentual para las roturas y otro para el material desechado. El análisis por agrupamientos de clientes, dadas las variaciones del mismo, condujo a la necesidad de observar los "cuellos de botella" del lado de la demanda sobre las paleteras y plantas, cuando estas no pueden satisfacerla. Se formuló un modelo de stock dinámico seleccionando al mes como la unidad temporal. Se estableció la forma adecuada de calcular los costos totales. Se construyó un modelo de simulación sobre hipótesis de demanda, roturas, desechos, costos de adquisición y transporte que permite estudiar y predecir el comportamiento del sistema más allá del período anual que involucraban los datos. Y finalmente dicho modelo teórico se programó en lenguaje c desarrollando un ejecutable que permitió distintas pruebas y ajustes. (Ver anexo). Los resultados más relevantes fueron presentados durante la realización del XV Congreso Internacional de Ingeniería Industrial y Afines COINI2022 y están disponibles en su libro de resúmenes.

Corresponde señalar que todo este trabajo se realizó a pesar de haberse jubilado el Ing. López, subdirector del proyecto quien no obstante siguió colaborando ad-honorem para poder finalizar los temas de programación de la simulación. También el Ing. Bosio, quien poseía la experiencia en el trabajo que es simulaba renunció a su cargo en UNLaM en abril de 2022 para irse a trabajar al

extranjero aunque siguió en contacto asesorando sobre las hipótesis elegidas y la Ing Bazán que comenzó el trabajo de investigación siendo alumna y en octubre de 2022, ya recibida, dejó la tarea por trasladarse al extranjero para cursar una maestría no relacionada con los temas de logística. Así el trabajo fue culminado por el resto del equipo con las obvias dificultades de no contar con tiempo para reuniones y análisis presenciales.

B. Principales resultados de la investigación

B.1. Publicaciones en revistas (informar cada producción por separado)

Artículo 1:	
Autores	
Título del artículo	
N° de fascículo	
N° de Volumen	
Revista	
Año	
Institución editora de la revista	
País de procedencia de institución editora	
Arbitraje	Elija un elemento.
ISSN:	
URL de descarga del artículo	
N° DOI	

B.2. Libros

Libro 1	
Autores	
Título del Libro	
Año	
Editorial	
Lugar de impresión	
Arbitraje	Elija un elemento.
ISBN:	
URL de descarga del libro	
N° DOI	

B.3. Capítulos de libros

Autores	Agustín Bosio, Mariana Bazán, Héctor Núñez, Luis López, Cristóbal R. Santa María
Título del Capitulo	Gestión de Operaciones y Logística. Artículo: Explotación de datos y simulación sobre un sistema de stock y transporte
Título del Libro:	Libro de Resúmenes del XV COINI 2022
Año	2022
Editores del libro/Compiladores	Editorial de la Universidad Nacional de Mar del Plata (Eudem)
Lugar de impresión	Mar del Plata
Arbitraje	SI
ISBN:	978-987-811-066-0
URL de descarga del capítulo	http://www.dii.fi.mdp.edu.ar/index.php/noticias/9-blog/204-lr-coini2022
N° DOI	

B.4. Trabajos presentados a congresos y/o seminarios

Autores	Agustín Bosio, Mariana Bazán, Héctor Núñez, Luis López, Cristóbal R. Santa María
Título	Explotación de datos y simulación sobre un sistema de stock y transporte
Año	2022
Evento	XV COINI 2022
Lugar de realización	Mar del Plata
Fecha de presentación de la ponencia	*10/11/2022*
Entidad que organiza	Universidad Nacional de Mar del Plata. Facultad de Ingeniería.
URL de descarga del trabajo	http://www.dii.fi.mdp.edu.ar/index.php/noticias/9-blog/204-lr-coini2022

B.5. Otras publicaciones

Autores	
Año	
Título	
Medio de Publicación	

C. Otros resultados. Indicar aquellos resultados pasibles de ser protegidos a través de instrumentos de propiedad intelectual, como patentes, derechos de autor, derechos de obtentor, etc. y desarrollos que no pueden ser protegidos por instrumentos de propiedad intelectual, como las tecnologías organizacionales y otros. Complete un cuadro por cada uno de estos dos tipos de productos.

C.2. Otros desarrollos no pasibles de ser protegidos por títulos de propiedad intelectual. Indicar: Producto y Descripción.

Producto	Descripción
Software Simulador	Programa ejecutable en lenguaje C

E. Otros recursos humanos en formación: estudiantes/ investigadores (grado/posgrado/ posdoctorado)

Apellido y nombre del Recurso Humano	Tipo	Institución	Período (desde/hasta)	Actividad asignada[1]
Bazan, Mariana	Estudiante	DIIT - UNLaM	01/01/2022 – 31/12/2022	Carga y limpieza de datos, pruebas de simulación, colaboración en la confección de artículos e informes

Cuerpo de anexos:

- Anexo I: Copia de cada uno de los trabajos mencionados en los puntos B, C y D, y certificaciones cuando corresponda.[2]
- Anexo II:
  - FPI-013: Evaluación de alumnos integrantes. (si corresponde)
  - FPI-014: Comprobante de liquidación y rendición de viáticos. (si corresponde)
  - FPI-015: Rendición de gastos del proyecto de investigación acompañado de las hojas foliadas con los comprobantes de gastos.
  - FPI-035: Formulario de reasignación de fondos en Presupuesto.
- Anexo III: Alta patrimonial de los bienes adquiridos con presupuesto del proyecto (FPI 017)
- Nota justificando baja de integrantes del equipo de investigación.

[1] Descripción de la/s actividad/es a cargo (máximo 30 palabras)
[2] En caso de libros, podrá presentarse una fotocopia de la primera hoja significativa o su equivalente y el índice.

Cristóbal R. Santa María
Firma y aclaración
del director del proyecto.

Lugar y fecha: San Justo, 10 de Marzo de 2023

- Presentar una copia impresa firmada del presente documento junto con los Anexos, y enviar todo en archivo PDF por correo electrónico a la Secretaría de Investigación Departamental. **Límite de entrega: 28 de febrero de 2020**

Anexo I

Libro de Resúmenes del XV COINI 2022

Ya se encuentra disponible para la descarga el **Libro de Resúmenes del XV COINI 2022**

ISBN 978-987-811-066-0

Puede descargarlo desde el QR o haciendo **Click Aquí**

En este libro encontrará toda la información necesaria para poder seguir y disfrutar del XV COINI 2022.

Minería de datos y simulación sobre un sistema de stock y transporte

Agustín Bosio, Mariana Bazán, Héctor Núñez, Luis López, Cristóbal R. Santa María

DIIT-UNLaM
Florencio Varela 1903 San Justo Pcia. de Buenos Aires
54-011-44808952
abosio@unlam.edu.ar
marianasbazan@gmail.com
hnuniez@unlam.edu.ar
llopez@ing.unlam.edu.ar
csantamaria@unlam.edu.ar

RESUMEN

Las técnicas de modelado aplicadas a distintas ramas de la industria y los servicios han demostrado su eficiencia para planificar, controlar y optimizar costos. Sin embargo, en muchos campos por simple hábito, por urgencia de los tiempos de entrega, o por considerar costosa o engorrosa su implementación, no se hace uso de esta poderosa herramienta para la decisión óptima. Con el ejemplo de operatoria de una empresa del área de logística, dedicada al alquiler de pallets, se pretende mostrar la eficacia del modelado frente al uso de métodos heurísticos basados en la idoneidad de los operadores. Desde distintos almacenes ubicados en el país la empresa distribuye a clientes de todas las provincias los pallets, organizando el transporte, la recuperación y la reparación de acuerdo a la demanda general. Se involucran cuestiones de registro y proceso de los datos, de stocks, de demanda, de transporte, de reparación e inutilización, en los costos de una operatoria que presenta "cuellos de botella" en los cuales no llega a satisfacerse toda la demanda. Con base en la explotación de los datos existentes se determinaron los patrones de comportamiento del sistema. Dado esto se propuso entonces realizar un modelo de simulación que vincule todos estos aspectos y de una respuesta óptima en términos de costos y beneficios. Para ello se combinan técnicas estadísticas, de explotación de datos y de simulación cuyo empleo podrá servir como marco metodológico general para casos similares. Este desarrollo se realiza dentro del Programa de Incentivos a la Investigación en la Cátedra de Investigación Operativa de UNLAM

Palabras Claves: Stock-Demanda-Transporte-Minería de Datos-Simulación

ABSTRACT

Modeling techniques applied to different branches of industry and services have demonstrated their efficiency in planning, controlling and optimizing costs. However, in many fields due to simple habit, the urgency of time, or because its implementation is considered costly or cumbersome, this powerful tool is not used. The operation of a company in the logistics area, dedicated to the rental of pallets, is presented here as an example, and it is intended to show the effectiveness of data mining techniques, together with simulation modeling, compared to the use of methods based on heuristics, and suitability of operators. From different plants located in the country, the company distributes its stock pallets to its customers in all provinces, organizing transport, recovery and repair according to general demand. Issues of stocks, demand, repair and disablement are then involved in the costs of an operation that presents "bottlenecks" over time, during which not all demand is satisfied. Based on the exploitation of existing data that determines the behavior patterns of such a system, it is, then proposed to formulate a simulation model that links them and provides an optimal response in terms of costs and benefits. Statistical, data mining and simulation techniques are combined whose use also aims to provide a general methodological framework to address similar cases. This development is carried out within the Program of Incentives for Research in the Chair of Operational Research of UNLAM

Keywords: Stock-Demand-Transportation-Data Mining-Simulation

1. INTRODUCCIÓN

Se parte del conocimiento experimental de la operatoria de una empresa con actuación en el ámbito nacional que adquiere, almacena, distribuye en alquiler, repara y repone cantidades variables de un producto de uso masivo en logística. Se trata de pallets. Para ello dispone de varias plantas ubicadas en distintos puntos geográficos del país en tanto sus clientes se hallan diseminados también sobre todo el territorio nacional. El stock con que se cuenta para esta operatoria está mayormente distribuido entre los diferentes clientes, desde los cuales deberá ser trasladado a alguna planta, reparado o desechado allí, si correspondiera, y luego tal vez incrementado para satisfacer la dinámica de la demanda. La demanda de pallets es satisfecha entonces desde las distintas plantas de distribución o por remisiones directas desde los fabricantes en el caso en que sea material nuevo que adquiere la empresa. Los pallets se transportan y entregan teniendo en cuenta los criterios de celeridad y cuidado, distancia geográfica y costos de envío. En ciertas oportunidades se producen “cuellos de botella” durante los cuales la alta demanda no puede ser cumplida satisfaciendo los criterios apuntados. La operatoria se resuelve actualmente en forma empírica basada en la idoneidad de los operadores más que en adecuados cálculos que optimicen su costo global y atenúen o eliminen los “cuellos de botella”. Por estas razones se ha pensado en diseñar un modelo del sistema que permita considerar la interrelación entre la demanda, el stock y los costos. De tal forma, se espera contar con una herramienta de simulación que, variando dinámicamente condiciones de demanda, tamaño de stock y costos, permita evaluar las decisiones óptimas en cuanto a transporte e inventario.

2. MATERIALES Y MÉTODOS

2.1. Datos disponibles

Se cuenta con datos de la operatoria durante un año volcados sobre una planilla Excel. Corresponden a 3148 instancias que representan la misma cantidad de envíos de pallets desde las plantas de distribución hacia los clientes en el período comprendido entre el 10 de marzo de 2017 al 31 de marzo de 2018. La primera columna es para la variable cualitativa que establece si el material es nuevo o usado. La segunda contiene la fecha de carga y la tercera la de descarga. La cuarta es el número interno de envío, la quinta establece el tipo de pallets que contiene el envío, la sexta la cantidad de pallets en el envío. Séptima y octava columna están reservadas para la planta de origen y el cliente de destino respectivamente. La novena corresponde al transportista que realiza el viaje y la décima es la variable que establece si el despacho se canceló, se rechazó o fue efectivizado. La siguiente columna es para observaciones y luego hay dos columnas más donde se consignan costo total del envío y costo por cada pallet. La planilla es acompañada por otras que establecen el número de viajes por fecha, los viajes de recupero de pallets realizados y una de costos por unidad.

2.2. Ideas de Modelado

La idea principal es integrar el modelo del sistema con tres bloques correspondientes a demanda, stock y costos. A través del proceso de explotación de los datos se busca establecer los patrones que rigen sus comportamientos, las variables que puedan resultar de interés para la decisión y la escala temporal con la que resulte adecuado trabajar. También se verá en esa etapa que hipótesis deben adoptarse para cubrir faltas de información, cómo debe ser el registro de la operatoria a futuro para alimentar nuevas simulaciones sin necesidad de nuevas fases de explotación y, en general la granularidad con la que considerar cada variable.

La secuencia de modelado requerirá conocer la Demanda de los clientes, determinar el Stock disponible para cubrirla y finalmente evaluar los Costos sumando los de transporte a los de reposición y/o arreglo del material. Se tratará de considerar en primer término un esquema que englobe estos aspectos relacionalmente con hipótesis lo más generales posibles para luego ir particularizando en las situaciones específicas que la operatoria conlleva. Debe observarse que en este sistema el stock total con que cuenta la empresa es itinerante pudiendo encontrarse cada pallet en una planta propia, en un cliente o en tránsito lo que requerirá establecer un stock disponible a intervalos fijos de tiempo para satisfacer una demanda que también deberá considerarse

en adecuados intervalos temporales. Sobre esto habrá que establecer los cálculos de costos y los parámetros y variables a tener en cuenta para resolver las situaciones de faltantes.

2.3 Curado de la Base de Datos

Se comenzó por dar nombres a las variables de la base de datos (planilla BD) que representen adecuadamente su contenido. Allí donde fue necesario se modificó el tipo de dato para hacerlo numérico o categórico según las necesidades funcionales y el software a utilizar. Se estudiaron los datos faltantes y las inconsistencias provenientes de la utilización de distintos nombres para un mismo caso de algunas variables. Esto se hizo particularmente necesario en las variables nominadas como Origen y Destino donde existían plantas y clientes singulares que figuraban bajo distintas nominaciones. En la variable Destino además se consideró como uno el caso del cliente que tuviera depósitos receptores distintos, próximos geográficamente, pues estas diferencias no tenían incidencia alguna en los costos. En la variable Origen se observó que algunos correspondían a Fabricantes que enviaban directamente los pallets adquiridos por la empresa a los clientes y los demás a plantas de la empresa, lo que se tuvo en cuenta en el análisis como se detallará más adelante. Además, al considerar los pares Origen-Destino se observó que algunos orígenes correspondían a clientes que enviaban pallets a las plantas, es decir que se habían registrado de esa forma algunos retornos, lo que requirió tipificar cada instancia con una variable Tipo de Destino, A o B. Esto sin duda representa un desorden de registro que debe solucionarse a futuro. La variable Tipo de Pallets contiene los casos Arlog, muy mayoritario, Europeo, Perimetral y unas pocas instancias Sin Identificar. Se decidió no considerarla en el armado inicial del modelo porque los costos de transporte eran los mismos y porque el stock y la demanda son en realidad para cada tipo por lo cual, una vez armado el esquema, podría fácilmente particionarse adecuadamente replicándolo para cada caso. Por último, se consideró necesario crear la variable Mes a partir de la variable Fecha de Carga para poder así cumplir con la idea de tomar el mes como la unidad temporal si así se lo decidiera. Del resto de las planillas disponibles se consideró muy útil aquella que contiene los retornos de pallets. En ella la variable Origen indica el cliente que envía los pallets o a veces el depósito del transporte desde donde salen, el Destino es la variable que indica la planta de la empresa a la que los pallets retornan. Hay que anotar aquí que los pallets no siempre vuelven a la planta de la que salieron lo que requerirá una compensación interna de la empresa que se analizará más adelante. Los costos totales por cada origen-destino, la cantidad anual de pallets y el número de viajes realizados al año figuran en sendas variables. Se nombró adecuadamente las variables, se les asignó su tipo adecuado, se estudiaron faltantes y outliers, y se observó especialmente que los nombres de destinos y orígenes fueran consistentes con los establecidos en la base de datos. Por otra parte, se notó que no existe discriminación por tipo de pallet en estos datos lo que abonó aún más la idea de considerarlos todos en una única clase hasta lograr una registración adecuada. La planilla de costos también adjuntada se refiere al costo de cada pallet y no a su transporte. Consigna su costo nuevo, los costos de reparación, de inspección y de tratamiento térmico, pero tampoco consigna estas cifras discriminando por tipo de pallet. Por supuesto en todo este trabajo se tuvo en cuenta la perspectiva de modelado que se consideró inicialmente y las posibles transformaciones a realizar para explotar los datos.

2.4. Explotación de datos

Disponer los datos para que respondan a la idea de modelo concebida, requiere distintas transformaciones de formatos y variables basadas en decisiones tomadas en consulta con quienes efectivamente llevan adelante la operatoria de la empresa. No cualquier simplificación ni agrupación de variables, realizada en forma general y sin conocimiento específico, resulta adecuada. Sin embargo, se logró llevar a cabo algunas que permitieron compactar y hacer más ágiles los cálculos y la visión de los resultados. Por otra parte, las variables tomadas en cuenta o diseñadas deben poder expresar las relaciones entre los distintos bloques del modelo. Ambos aspectos de presentación y contenido de las variables resultan entonces el primer paso necesario para la explotación de los datos. Como consecuencia de esto los datos se transformaron y dispusieron en tres bloques según se detalla.

2.4.1. Demanda

Para procesar los datos se comenzó ordenando las instancias según la fecha carga. A continuación, se analizaron los orígenes desde los cuales salen los pallets. Se observó que había dos tipos de orígenes. Uno que correspondía a Fabricantes que enviaban directamente los pallets nuevos adquiridos por la empresa a los clientes, y el otro que se refería a las Plantas (o Paleteras) de distribución de la empresa desde donde pallets ya usados, recuperados y eventualmente reparados eran enviados nuevamente a clientes. Luego del curado de los datos se obtuvo entonces la distribución de la Tabla 1 que muestra el aporte de cada planta (paletera), en cantidad de transportes realizados, a la satisfacción anual de la demanda de los destinos y el aporte realizado directamente desde los fabricantes de pallets nuevos.

Tabla 1 *Frecuencias absolutas de envíos mediante camión*

Origen	Total	Porcentaje
Fabricante 1 Colonia Caroya	219	6,96
Fabricante Campana	133	4,23
Fabricante Federación	146	4,64
Fabricante Misiones	205	6,52
Fabricante Rosario	176	5,59
Paletera Cba. Capital	204	6,48
Paletera Chaco	254	8,07
Paletera La Plata	46	1,46
Paletera Mar Del Plata	37	1,18
Paletera Mendoza	136	4,32
Paletera Santa Fe	24	0,76
Paletera Tigre	255	8,11
Paletera Tucumán	242	7,69
Planta AMBA Norte	1069	33,98
Total	3146	100,00

Se plantea además un problema que debe resolverse. Al considerar que los datos cubren un lapso temporal de aproximadamente un año se observa que la demanda de los clientes medida en cantidad de entregas mediante camión anuales tiene una gran dispersión por lo cual hay que establecer adecuadamente la unidad temporal a considerar. Si se tiene en cuenta que los datos de retorno de pallets aportados en cuanto a cantidades y viajes consideran el total en 13 meses, de marzo 2017 a marzo 2018, parece una buena idea establecer inicialmente el mes como la unidad temporal. Esto resulta coherente con el tiempo que, en términos habituales, los pallets permanecen en poder de los clientes, aproximadamente unos dos meses. Cabe aclarar además que se resuelve tomar la fecha de carga, que es la fecha en la cual la empresa dispone los pallets para envío a efecto de establecer la demanda. Mes a mes hay entonces una demanda de los clientes sobre cada planta de la empresa que se transformará en la función de extracción del stock. Esto lleva a integrar la demanda de los clientes que se realiza sobre una misma planta. En realidad, a efecto de evaluar el stock mes a mes en cada planta es necesario elaborar un poco más estos datos observando el mes que corresponde a la fecha de carga de cada transporte y la cantidad de pallets transportada en cada oportunidad. Así puede establecerse la cantidad que cada fabricante y planta (o paletera) debe enviar por mes hacia los clientes.

Entonces, una vez determinados los transportes mensuales hay que calcular la cantidad de pallets que llevan en total, considerando además como ya se dijo, todos los tipos de pallets en conjunto. La Tabla 2 muestra los resultados de haber realizado este trabajo.

Tabla 2 Demanda mensual sobre Fabricantes y Paleteras. Frecuencias absolutas en pallets

Origen	M1P	M2P	…	M12P
Fabricante 1 Colonia Caroya	0	18348	…	10274
Fabricante Campana	4800	6624	…	3200
Fabricante Federación	400	6352	…	5900
Fabricante Misiones	11900	7260	…	10700
Fabricante Rosario	7400	7400	…	5600
Paletera Cba. Capital	2400	1690	…	0
Paletera Chaco	9414	7200	…	9538
Paletera La Plata	2320	3200	…	400
Paletera Mar Del Plata	780	750	…	250
Paletera Mendoza	400	400	…	1990
Paletera Santa Fe	0	0	…	0
Paletera Tigre	10070	18802	…	5000
Paletera Tucumán	7560	9576	…	3456
Planta AMBA Norte	37820	21700	…	3369

Esta tabla será insumo de la simulación a efectuarse y en adelante recibirá el nombre de Tabla Input.Demanda Adicionalmente se considera solo una parte de esta tabla en la cual se tiene solo la demanda efectuada a los fabricantes mes a mes. Esta nueva Tabla Input.Demanda.Fabricantes será útil a efecto de cubrir según la distribución mensual de demandas los faltantes de stock que se produzcan. A continuación, se traspuso la Tabla.Input.Demanda de modo tal que cada fabricante o paletera resultara una variable para el trabajo de simulación estadística ulterior. A tal fin, para cada una de estas variables, también se calculó su máximo y su mínimo lo que quedó registrado en la Tabla Input.min.Max.Demanda.

2.4.2 Stock

El stock disponible mes a mes se integra con los pallets que ya están acumulados en las distintas plantas como excedente no comercializado durante ese lapso, los que retornan a ellas desde los distintos clientes y los que se compran nuevos para cubrir ampliaciones de demanda y roturas, pérdidas etc. que se hubieran producido. Es decir, cada planta debe contar con un perfil disponible de stock anual detallado mes a mes. Como ya se apuntó hay registro de transportes que regresan pallets hacia las paleteras para inspección, reparación o descarte, y nuevo envío. Ese flujo esta detallado, por imperio de un registro de datos desordenado, en dos lugares. Por un lado, se encuentra en la base de datos expresado como viajes que se realizan desde los clientes hacia las plantas (o paleteras). En la expresión de esos datos han tenido que corregirse distintas inconsistencias nominativas. El otro lugar donde se encuentran datos similares es en la planilla de Retornos. Allí nuevamente se hizo necesario resolver inconsistencias. En esta planilla además del origen y destino, hay una variable que detalla los costos totales del transporte para cada par origen-destino, la cantidad de pallets totales retornados durante el año, la cantidad de viajes, y las cantidades de pallets promedio por viaje y por mes. Para evaluar la existencia de stock mes a mes en cada paletera deberá entonces consolidarse la información de los pallets existentes en cada planta (o paletera) con los aportes de los recuperos indicados en la base de datos más los retornos indicados en planilla separada. Tipificados con distinto nombre según su lugar de registro, los recuperos y los retornos involucran siempre el regreso de pallets a las plantas o paleteras para formar parte del stock. Se denominó recuperos mensuales a los retornos anotados en el caso B de la variable Tipo de Destino que se asocia con el tránsito interno de la empresa, producido desde un cliente hacia una planta o paletera, o como viajes entre plantas y/o paleteras. Con el auxilio de la variable Mes puede establecerse la cantidad de viajes de este tipo efectuados por mes a los destinos donde los pallets

serán acopiados para su posterior reenvío a clientes. Se construye entonces la Tabla 3 que informa los recuperos en cantidad de pallets por mes.

Tabla 3. Recuperos por mes. Frecuencia absoluta en pallets

Destino	M1P	M2P	…	M12P
Paletera Cba Capital	0	0	…	1990
Paletera Chaco	0	0	…	0
Paletera La Plata	2400	0	…	0
Paletera Mar Del Plata	1080	360	…	0
Paletera Mendoza 1	400	0	…	0
Paletera Santa Fe	0	1000	…	500
Paletera Tigre	800	0	…	800
Paletera Tucuman	0	0	…	1560
Planta AMBA Norte	0	480	…	0

El tema de los retornos resulta un poco más complejo ya que no están desglosados por mes ni por cantidad de pallets. La solución propuesta a este problema fue considerar, en consulta con la empresa, el momento habitual en que retornaban los pallets alquilados. Se consideró así entonces, que los retornos se producían dos meses después del envío hacia el cliente desde una planta. Sin embargo, no necesariamente los pallets que llegan a un cliente desde una paletera o planta regresan al mismo lugar de modo que se apeló al criterio empírico de cercanía y al origen, en cada caso, para decidir en cual fecha y planta los retornos se podían suponer efectivizados. Hay que destacar aquí que el problema surge, no de la imposibilidad de realizar un mejor registro de toda esta operatoria de carácter mensual, sino del hábito que más o menos idóneamente se ha establecido para producir los retornos y registrarlos. Perfectamente se podría a partir de los requerimientos de modelado optimizador organizar los viajes de retorno y tomar estas estadísticas en forma más detallada y óptima. Esta es justamente una de las causas por las que se planteó la necesidad del modelado a efecto de minimizar costos, ya que la contabilización del real stock mensual de pallets en cada planta resulta, en los hechos, aproximada en forma grosera. Al adoptar criterios heurísticos para realizar el modelado y aconsejar luego un registro estadístico distinto y ajustado a la necesidad del modelo, se logra despejar el grado de incertidumbre sobre este aspecto crucial de la operatoria logística. Hay que considerar además que los pallets regresados son siempre menos que los enviados pues hay pérdidas, robos y destrucciones en un porcentaje que en la operatoria real actual se estima, una vez más, con aproximación grosera. Esta estimación deberá contemplarse al realizar la simulación a través, por ejemplo, de un parámetro fijado por el usuario que establezca un porcentaje de pérdida en la operatoria. La Tabla 4 muestra un ejemplo de cómo se dispone luego de todo el trabajo relatado la información de retornos.

Tabla 4. Retornos por mes. Frecuencias absolutas en pallets y viajes

Origen	Destino	Pallets	Viajes	M1P		M12P
Cliente 1 - Villa Mercedes.	Paletera Cba Capital	3600	9	1600	…	0
Cliente 10 - Santa Fe	Paletera Santa Fe	800	3	0	…	0
Cliente 11 - Pilar	Planta AMBA Norte	2520	6	0	…	0
Cliente 12 – Cipolletti.	Paletera La Plata	1100	5	0	…	0
Cliente 13 – Rosario	Paletera Chaco	405	1	0		....

Con la información de los recuperos, mes a mes, extraídos de la base de datos, y con la tabla de retornos por mes realizada integrando los datos existentes con hipótesis razonables de operación, se está en condiciones de establecer el stock real existente, mes a mes, para cada paletera o planta. Para esto se comienza por sumar las cantidades retornadas y recuperadas de pallets y disponerlas según la Tabla 5.

Tabla 5 Input Recuperación. Frecuencias absolutas en pallets

Destino	M1P	M2P …	M12P
Paletera Cba Capital	17224	14200 …	11462
Paletera Chaco	5188	6150 …	4405
Paletera La Plata	2722	0 …	800
Paletera Mar Del Plata	1330	1360 …	1250
Paletera Mendoza 1	2100	2000 …	1560
Paletera Santa Fe	2800	8200 …	4100
Paletera Tigre	800	43200 …	800
Paletera Tucuman	6828	0 …	1560
Planta AMBA Norte	49690	18850 …	52940

La Tabla 5 será otro insumo de la simulación bajo la denominación Input.Recuperación A efecto de la simulación resulta útil trasponer esta tabla para considera como variable cada origen y hallar sus máximos y mínimos de demanda. Esto se vuelca en la tabla Input.min.Max.Recuperación Para determinar el stock real mes a mes la información de recuperos y retornos deberá integrarse con la demanda sobre las plantas que efectúan los clientes. Esta se obtendrá de la tabla denominada Input.Demanda. Puede verse en la Tabla 6 como quedan estructurados estos datos.

Tabla 6. Input.Demanda. Frecuencias absolutas en pallets

Origen	M1P	M2P	…	M12P
Paletera Cba Capital	2400	1690	…	0
Paletera Chaco	9414	7200	…	9538
Paletera La Plata	1600	3200	…	400
Paletera Mar Del Plata	780	750	…	250
Paletera Mendoza	400	400	…	0
Paletera Santa Fe	0	0	…	0
Paletera Tigre	5670	18802	…	5000
Paletera Tucuman	7560	9576	…	3456
Planta AMBA Norte	34260	21340	…	33692

Se puede establecer que si hay faltantes de stock, deben cubrirse con adquisición de más pallets a los fabricantes, restando al stock disponible las cantidades de pallets demandadas a cada planta o paletera. Con todo, esta es una cuenta estática y no muestra adecuadamente la dinámica de la operatoria pues, en el caso de existir un sobrante de pallets en una planta un determinado mes, este será naturalmente destinado a cubrir la demanda del mes faltante por lo cual hace falta reflejar esta dinámica en el modelo. De este modo se cubren los faltantes de stock con pedidos a fabricantes y la simulación permitirá establecer a que fabricantes pedir desde que paleteras o plantas. Mes a mes se dispondrá de esta información y podrán compensarse anualmente los desequilibrios que ocurran sin afectar la satisfacción de la demanda.

2.4.3 Costos

Los costos tienen básicamente dos componentes: el costo de los pallets y el costo del transporte. La empresa no ha informado sobre costos de almacenamiento ni restricciones de espacio para realizarlo en cada paletera, aunque el modelo podría eventualmente incorporarlos con modificaciones sencillas. El costo de los pallets nuevos se obtiene de la planilla Costos Pallets donde no hay discriminación por tipo de pallets. Para obtener el costo total mensual por compra de pallets nuevos habrá que multiplicar la cantidad de pallets mensual que entrega cada fabricante por lo que se denominará Input.Costo.Pallet en la entrada a la simulación y que no es otra cosa que el costo por pallet nuevo. Los costos de transporte se relacionan con la distancia a recorrer entre paleteras o plantas o fabricantes y clientes. En una primera instancia se ha tenido en cuenta que la provisión de los clientes desde plantas, paleteras o fabricantes se realiza siguiendo criterios de cercanía. Esto hace posible establecer un costo promedio por pallet transportado desde cada origen a los clientes que abastezca. Tal información se obtiene de la base de datos aportada y es expuesta en la Tabla 7.

Tabla 7 Costo de transporte por pallet

Fabricante 1 Colonia Caroy..	31,23
Fabricante Campana	22,39
Fabricante Federación	11,76
Fabricante Misiones	35,63
Fabricante Rosario	9,77
Paletera Cba Capital	27,24
Paletera Chaco	37,76
Paletera La Plata	20,69
Paletera Mar Del Plata	13,23
Paletera Mendoza	43,52
Paletera Santa Fe	20,45
Paletera Tigre	11,63
Paletera Tucumán	20,73
Planta AMBA Norte	16,53

La Tabla 7 será entonces la entrada Input.Costo.Transporte en el modelo de simulación. Para establecer el costo de transporte por cada pallet que se reintegra a una planta o paletera se pueden usar los mismos costos pues si bien no siempre los pallets regresan al mismo lugar desde donde fueron enviados la alternativa de variarlo no implica en general costos distintos pues siempre se continúa empleando el criterio de cercanía. Vale aclarar que este criterio fijado originariamente en forma empírica se puede validar de acuerdo a los costos de transporte por pallets estableciendo una asociación entre cada origen destino y costo por pallet. Los pallets que reingresan al stock son inspeccionados lo que tiene un costo por pallet que se extrae de la planilla Costos Pallets. Este costo entrará en la simulación como la cantidad Input.Costo.Inspección que deberá aplicarse a cada pallet reingresante. En forma inicial el modelo aplicará un porcentaje de roturas a reparar que implicará un aumento a sumar a los costos del transporte. Esté número será otra de las entradas de la simulación denominada Input.Porcentaje.Roturas y se aplicará al total de pallets retornados multiplicando la cantidad obtenida por el costo de reparación por pallet, también obtenido de Costos Pallets y cuya entrada en la simulación de denominará Input.Costo.Rotura. Todos los costos calculados deberán integrarse mes a mes para obtener el costo de operación total.

2.5 Modelado de simulación

La información obtenida por explotación de los datos ingresará como distintos Inputs, sean parámetros o tablas de datos, a la simulación. El objetivo es simular el comportamiento del sistema durante el siguiente período con distintos valores de los parámetros a efecto de minimizar costos y hacer una previsión económica y financiera de los mismos. La Figura 1 muestra un esquema general del modelo.

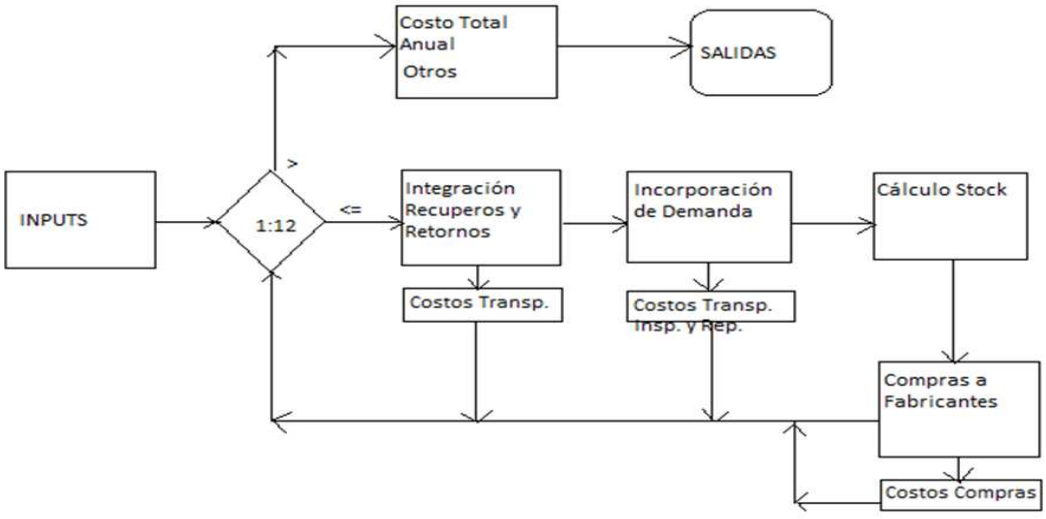


Figura 1. Esquema general del modelo

2.5.1 Entradas y Salidas

Las entradas al modelo de simulación son de dos tipos. El primer grupo está integrado por tablas extraídas de la explotación de datos realizada anteriormente y/o del registro estadístico que con igual formato pueda realizarse en adelante. El segundo grupo está constituido por los parámetros que caracterizan la operatoria.
Tablas de entrada: según lo ya visto las tablas de entrada serán las denominadas Input.Demanda, Input.Demanda.Fabricantes, Input.Demanda.Traspuesta, Input.min.Max.Demanda, Input.Recuperación, Input.Recuperación.Traspuesta, Input.min.Max.Rescuperación e Input.Costo.Transporte.
Parámetros de entrada: serán los valores de P (proporción de extravío, robo, destrucción) 0≤P≤1, R (proporción de rotura 0≤R≤1) Input.Costo.Pallet, Input.Costo.Inspección, e Input.Costo.Rotura. Todos estos estos parámetros son los que hacen dinámico al modelo

2.5.3 Incremento del tiempo

El modelo procederá a incremento de tiempo fijo. Es decir; las cantidades se actualizarán en forma mensual

2.5.4 Integración de Recuperos y Reintegros

Este bloque consistirá en la incorporación de la planilla Input.Recuperación al modelo.

2.5.5 Integración de la Demanda

Este bloque consistirá en la incorporación de la planilla Input.Demanda al modelo.

2.5.6 Cálculo de Stock

A efecto de establecer el stock es importante fijar una proporción de perdida, rotura total etc a través del parámetro P que puede tomar valores entre 0 y 1. Hay que aclarar que este coeficiente representa en realidad la proporción de pallets que no se recuperan por lo que su valor máximo es 1 cuando se lo hace con todos y 0 si no se recupera ninguno. Se multiplica cada columna de la planilla recuperación por este parámetro y se la denota Input.Recuperación.P El stock del mes de enero del primer año se integra con los recuperos más los retornos aportados por Input.Recuperación.P Cuando se resta a esa columna la demanda de enero, primera columna de la planilla Input.Demandas la cuenta puede dar positiva o negativa. Se constituyen así los residuos en falta o en exceso a agregar al stock disponible el mes siguiente. Si el residuo del mes de enero resulta positivo indicará sobrante y si es negativo faltante. Finalmente, entonces la primera columna del stock será la obtenida como se indicó y corresponderá al mes de enero del primer año. Ahora, para calcular el stock real (residuos) de febrero se suman a la primera columna la recuperación de febrero más los residuos del mes anterior y se restan las demandas del mes actual. Una vez más esta cuenta podrá dar positiva o negativa y constituye los residuos de febrero. Así se continua hasta el mes de diciembre. En cada mes se guarda entonces el residuo. La planilla completa con demandas, recuperaciones y residuos se guarda como Output.Stock.

2.5.7 Cálculos de Costos

2.5.7.1 Costos de transporte de recuperación

El material reingresado cada mes fue transportado desde los clientes a las paleteras. Los costos por pallets de esta operación se obtienen de la tabla Input.Costo.Transporte que permiten calcular el costo de reingreso a cada paletera de los pallets devueltos multiplicando su cantidad por el costo señalado. Hay que observar aquí que el cálculo de pérdidas, faltantes, así como la necesidad de reparaciones del material se efectúa una vez transportado de regreso a las paleteras, motivo por el cual debe trabajarse con la planilla Input.Recupe-

ración original sin multiplicar por el parámetro P. Se construye una planilla de costos de transporte por recuperación que mes a mes va señalando estos costos. Luego será integrada al costo total mensual. Se utiliza el nombre Output.Costo.Transporte.Recuperación.

2.5.7.2 Costo de inspección

Utilizando las entrada Input.Recuperación y el costo de inspección por pallet que figura en Input.Costos.Pallets se calcula el costo de inspección multiplicando cada cantidad de Input.Recuperación por el costo respectivo. Se guarda en la planilla Output.Costo.Inspección

2.5.7.3 Costos de rotura y reparación

Sobre la planilla Input.Recuperación se calcula la cantidad de pallets rotos a reparar aplicando a cada cantidad por mes y paletera el coeficiente de rotura que es la entrada Input.Porcentaje.Rotura.R. De nuevo hay que aclarar que este coeficiente representa en realidad la proporción de pallets que no se rompen por lo que su valor máximo es 1 cuando no hay pallets rotos y 0 si se rompieron todos. A las cantidades así obtenidas se las multiplica por el parámetro Input.Costo.Rotura. Se guarda en Output.Costo.Reparación

2.5.7.4 Costos de Compra a Fabricantes

Se opera con la fila suma de la tabla Input.Demanda. multiplicando cada cantidad solicitada a cada Fabricante por el parámetro Input.Costo.Pallet. La salida de guarda en Output.Costo.Compras.

2.5.7.5 Costo de transporte demanda a Paleteras Con la entrada Input.Demanda se procede multiplicando cada cantidad de demanda sobre paleteras por el costo de transporte cargado en Input.Costo.Transporte. Esto se guarda en Output.Costo.Transporte.Demanda.Paleteras

2.5.7.6 Costo de transporte demanda a Fabricantes

Se opera con la fila suma de la tabla Input.Demanda. multiplicando cada cantidad solicitada a cada Fabricante por el costo de transporte cargado en Input.Costo.Transporte. Esto se guarda en Output.Costo.Transporte.Demanda.Fabricantes

2.5.7.7 Costos totales

Las cantidades que figuran el cada una de las salidas de costos Output.Costo.Transporte.Recuperación, Output.Costo.Inspección, Output.Costo.Reparación, Output.Costo.Compras, y Output.Costo.Transporte.Demanda.Paleteras y Output.Costo.Transporte.Demanda.Fabricantes se suman para cada mes obteniéndose la salida Output.Costos.Totales. Sobre esta se calcula el costo total anual.

2.6 Simulación de período anual

El procedimiento puede repetirse en adelante hasta simular todos los años que se desee. Cada año las entradas paramétricas pueden variarse pues el modelo da esa posibilidad. Lo mismo ocurrirá con la tabla de costos de transporte. Además, para preservar la distribución de la aleatoriedad presente en la operatoria las tablas correspondientes a recuperación y demanda pueden simularse. Con la información de mínimos y máximos de demanda sobre cada origen, obtenida de la Tabla Input.min.Max.Demanda, se genera una nueva demanda por origen colocando una observación uniforme entre el mínimo y máximo apuntado que corresponde a la demanda para un mes. Esto se repite los 12 meses para cada origen obteniéndose una nueva Tabla Input.Demanda sobre la hipótesis de que la distribución de la demanda por fabricante tiene un comportamiento aleatoriamente uniforme. Este supuesto puede reemplazarse por cualquier otro, incluido uno que siga la distribución empírica que se obtiene de los datos originales. En forma similar se procede a la simulación de la tabla Input.Recuperación a partir de los datos de mínimos y máximos obtenidos de la Tabla Input.min.Max.Recuperación Todo el programa que vincula las distintas planillas y realiza los cálculos y la simulación, fue realizado en lenguaje C. En cada corrida pueden seleccionarse todos los parámetros de entrada

y con pequeñas modificaciones simular distintos comportamientos estocásticos de la demanda y la recuperación.

3. RESULTADOS

Se han realizado hasta aquí distintas pruebas. La corrida 0, que se realizó para comprobar el funcionamiento adecuado del programa, se introdujeron todos los datos y se supuso que no había pérdidas ni reparaciones a efectuar. Esto equivale al valor 1 de los parámetros P y R. Así se calculó el costo anual. Luego se realizaron corridas del programa simulando con distintos valores de parámetros P y R, para lo cual se adoptó el criterio de simulación basado en la aleatoriedad uniforme señalada pero solo para aquellos meses en los cuales los datos originales fueran distintos de cero. Si había cantidades nulas de demandas o recuperaciones se suponía que estas provenían de comportamientos estacionales que convenía mantener en la simulación. Las distintas salidas que se obtuvieron y que se muestran en la Tabla 8

Tabla 8 Simulaciones

Corrida Nº	Coeficiente P	Coeficiente R	Costos Totales	Stock Residual
0	1	1	349187946,79	332199
1	0,9	0,8	465862600,11	425526
2	0,80	0,70	420579739,77	281775
3	0,75	0,65	441645356,02	116033

Se observa que el costo en la corrida 0 es más bajo por cuanto se ha supuesto que no hay roturas ni pérdidas. El residuo de stock acumulado muestra que los pallets en poder de la empresa al cabo de un año cubren completamente la demanda y que los ocasionales cuellos de botella deben estudiarse pormenorizadamente allí donde ocurran. En cada una de las corridas siguientes se ha tomado la misma demanda inicial y la misma recuperación y se han variado los parámetros correspondientes a las pérdidas y roturas. Como era de esperar hay un sensible aumento de los costos cuando hay pérdidas y roturas. Sin embargo, el crecimiento del residuo de stock podría mostrar que resulta necesario ajustar con mayor precisión la representación aleatoria de la demanda y la recuperación utilizando las distribuciones empíricas observadas para las mismas en la fase de explotación de datos, descartando expresiones analíticas más sencillas como la distribución uniforme utilizada hasta aquí. Se observa que, si bien los costos se mantienen dentro del mismo orden de magnitud, la reducción del residuo de stock es coherente con los valores decrecientes de los parámetros P y R que representan un crecimiento de la pérdida y la rotura.

Para analizar los llamados cuellos de botella y tomar decisiones es útil observar los residuos de stock por paletera. La Figura 2 muestra los residuos en dos corridas. Se ve que, aunque existen faltantes de stock en algunas paleteras el total de stock disponible en el año es suficiente para cubrir toda la demanda. Por lo tanto, resolver los faltantes en algunas paleteras requiere establecer un mecanismo de compensación interna adecuado que mensualmente vaya cubriendo faltantes allí donde los hay.

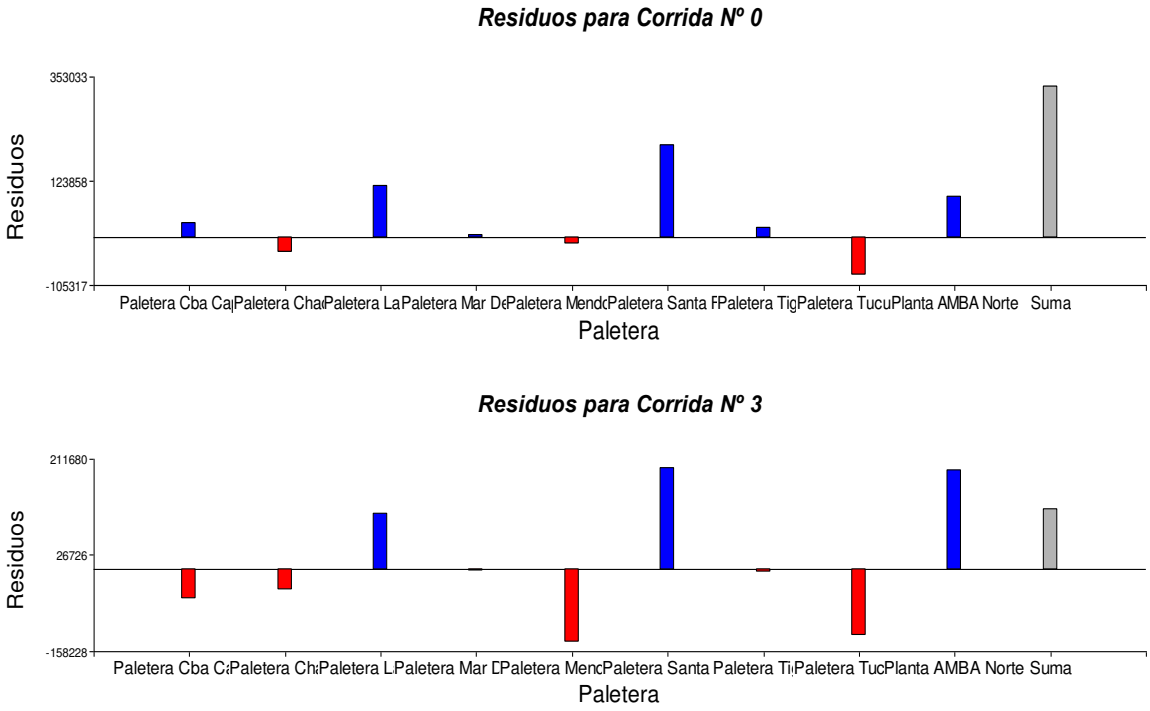


Figura 2. Análisis de faltantes de stock

4. CONCLUSIONES

El modelo desarrollado permite en primer término recomendar, a partir de la explotación de los datos realizada una forma de registro de operaciones que permita evaluar, programar y controlar, en forma más automatizada, la actividad de la empresa en función de satisfacer la demanda con mínimo costo. Tal cual muestra el grado de avance hasta aquí, se requiere ajustar el perfil estadístico de la demanda y de la recuperación de pallets para validar los parámetros y calibrar el modelo de acuerdo a la operatoria real observada. La estructura elegida facilita incorporar la variación de todos los costos reemplazando valores en sus planillas de entradas denominadas con el prefijo Input. De igual manera pueden considerarse diferentes situaciones de pérdida o necesidad de reparación de material. La capacidad de simular la operatoria para siguientes períodos permite predecir comportamientos e identificar problemas para tomar decisiones adecuadas. En particular la posibilidad de observar mensualmente el stock en cada paletera según el esquema de demandas y recuperación de material en cada lapso temporal facilita las decisiones en pos de cubrir los “cuellos de botella” que dejan insatisfecha la demanda aun cuando se dispone de stock en forma global.

La continuidad del trabajo se orienta a considerar el registro completo de la actividad por tipo de pallets, cosa que las estadísticas tomadas hasta aquí en realidad no permiten, para facilitar un análisis más fino de la demanda, del stock y de los costos. Por otra parte, con datos obtenidos con esa precisión cobraría mayor sentido obtener la simulación sobre el comportamiento observado de la demanda y la recuperación. También sería de mayor relieve, en ese caso, programar una forma automática de compensación de los faltantes de stock, allí donde los hubiere, que tuviera un mínimo costo. Estas son entonces las direcciones en las que debe proseguir el desarrollo del modelo.

5. BIBLIOGRAFÍA

Ballou, R. (2004) *Logística: Administración de la cadena de suministro.* Pearson Education.

Browersox, D. J; Closs, D.J; Cooper M.B. (2007) *Administración y logística en la cadena de suministros*. McGraw-Hill

Cruz Herradón, A. M; de Prado Morante, S.R; y Meseguer Galán, P. (2020) *Gestión logística y comercial.* MacMillian Education.

Di Rienzo J.A., Casanoves F., Balzarini M.G., Gonzalez L., Tablada M., Robledo C.W. (2018) *InfoStat versión 2018*. Grupo InfoStat, FCA, Universidad Nacional de Córdoba, Argentina. URL http://www.infostat.com.ar

Endres, D.; Schindelin, J. (2003) *A new metric for probability distributions*. IEEE Transactions on Information Theory. Vol. 49 N° 7 pp1858-1860.

Everitt, B. Landau, S. Leese, M.(2011) *Cluster analysis*. Wiley.

Gaffney,C. (2019) *Definition of Logistics Costs: Small Business-Chron*. https://smallbusiness.chron.com/

Gómez Aparicio, J. M. (2013) *Gestión logística y comercial*. McGraw-Hill.

Gómez M, R y Correa E, A. (2011) *Análisis del transporte y distribución de materiales de construcción utilizando simulación discreta en 3D.* Boletín de Ciencias de la Tierra. Universidad Nacional de Colombia. rbct@unalmed.edu.co

Guerrero Salas, H. (2017*) Inventarios. Manejo y Control*. Ecoe Ediciones.

Han, J; Komber, M; Pei, J. (2011) *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

Hillier, F; Lieberman,G. (2010) *Introducción a la Investigación de Operaciones*. McGraw-Hill.

Jiménez Avello, A. (2013) *Simulación de procesos y aplicaciones*. Dextra.

Johnson, D. (2000) *Métodos multivariados aplicados al análisis de datos*. Thomson editores.

Marino, N. (2013) *Global facts.* Inbound logistics- Latam N°97. V7.2013 pp. 16-19

Mba Skool. Logistics Costs. (2019) https://www.mbaskool.com/business-concepts/operations-logistics-supply-chain-terms/14446-logistics-costs.html

Mora García, L. A. (2011) *Gestión logística en centros de distribución y almacenes y bodegas.* Ecoe Ediciones.

Paz, H. R. (2008) *Canales de distribución: Gestión comercial y logística*. Lectorum Ugerman.

Torres, M.M. (2014) *Gestión de stock*. Días de Santos.

Weil, M. I.T. (2013) *Toolkit*. Inbound logistics- Latam N°97. V7 pp. 12-18

Yong Shi, Yingjie Tian, Gang Kou, Yi Peng, Jianping Li (2011) *Optimization based data mining: theory and aplications*. Springer https://doi.org/10.1007/978-0-85729-504-0


XV Congreso Internacional de Ingeniería Industrial y Afines
CERTIFICADO
Cristóbal Raúl Santa María
Ha participado como AUTOR/A del trabajo "Explotación de datos y simulación sobre un sistema de stock y transporte", que lleva por código CO22-G02 y ha sido presentado en el XV COINI 2022 que se desarrolló entre el 7 y el 12 de noviembre de 2022. La Asociación Argentina de Carreras de Ingeniería Industrial y Afines - AACINI extiende el presente en la ciudad de Mar del Plata, a los 12 días del mes de noviembre de 2022.
Mg. Ing. Antonio Morcela
Presidente COINI 2022
Esp. Miguel Ángel Rissetto
Presidente AACINI
COINI 2022
aacini
Departamento de Ingeniería Industrial
Facultad de Ingeniería
Universidad Nacional de Mar del Plata

Programas desarrollados para la simulación.

1er etapa: proyecto "ParaStockYSumaResiduos.cbp"

main.h

```
#ifndef MAIN_H_
#define MAIN_H_

#include <stdio.h>
#include <stdlib.h>

#include "residuos.h"


#define    ARCH_RECUP     "Input.Recuperacion.csv"
#define    ARCH_DEMAN     "Input.Demanda.csv"
#define    ARCH_PSTOK     "Para.Stock.csv"
#define    ARCH_RESID     "Suma.Residuos.csv"

float recuperarParametro(int cArg, char *argumento);


#endif // MAIN_H_
```

main.c

```
#include "main.h"

int main(int cArg, char **vArg)
{
   int retVal = 1;
   float   param = recuperarParametro(cArg, *(vArg + 1));

   fprintf(stdout,
         "Procesando los archivos:\n\"%s\"\n\"%s\".\n",
         ARCH_RECUP, ARCH_DEMAN);
   retVal = procesarArchivos(ARCH_RECUP, ARCH_DEMAN, ARCH_PSTOK,
ARCH_RESID,
                        param);
   if(retVal == 0)
      fprintf(stderr, "ERROR - procesando archivos.\n");
   else
```

```
    fprintf(stdout,
            "Fin de programa fue exitoso.\n"
            "Se generaron %d filas en el archivo \"%s\".\n",
            retVal, ARCH_PSTOK);
  return !retVal;
}


float recuperarParametro(int cArg, char *argumento)
{
  float   param = argumento && *argumento ?
                  (int)(atof(argumento) * 100) / 100. : 1.0;



  return param;
}
```

funciones.h // es común a varios programas

```
#ifndef FUNCIONES_H_
#define FUNCIONES_H_

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

int abrirLosArchivos(FILE **fp1, const char *arch1, const char *modo1,
                FILE **fp2, const char *arch2, const char *modo2,
                FILE **fp3, const char *arch3, const char *modo3);

int contarCamposYVerifLineaTexto(const char *linea, int cant, const char *msj);

int msjError(const char *msj);

#endif // FUNCIONES_H_
```

funciones.c // es común a varios programas

```
#include "funciones.h"


int abrirLosArchivos(FILE **fp1, const char *arch1, const char *modo1,
                FILE **fp2, const char *arch2, const char *modo2,
```

```
                FILE **fp3, const char *arch3, const char *modo3)
{
   char    patNam[100],
         *pat;

   pat = getenv("PAT_INP_1");
   sprintf(patNam, "%s%s", pat ? pat : ".\\", arch1);
   if(fp1)
      *fp1 = fopen(patNam, modo1);

   pat = getenv("PAT_INP_2");
   sprintf(patNam, "%s%s", pat ? pat : ".\\", arch2);
   if(fp2)
      *fp2 = fopen(patNam, modo2);

   pat = getenv("PAT_OUT");
   sprintf(patNam, "%s%s", pat ? pat : ".\\", arch3);
   if(fp3)
      *fp3 = fopen(patNam, modo3);

   if(fp1 && *fp1 == NULL)
      fprintf(stderr,
            "ERROR - abriendo archivo \"%s\" en modo \"%s\"\n",
            arch1, modo1);
   if(fp2 && *fp2 == NULL)
      fprintf(stderr,
            "ERROR - abriendo archivo \"%s\" en modo \"%s\"\n",
            arch2, modo2);
   if(fp3 && *fp3 == NULL)
      fprintf(stderr,
            "ERROR - abriendo archivo \"%s\" en modo \"%s\"\n",
            arch3, modo3);
   if((fp1 && *fp1 == NULL) || (fp2 && *fp2 == NULL) || (fp3 && *fp3 == NULL))
   {
      if(fp1 && *fp1)
         fclose(*fp1);
      if(fp2 && *fp2)
         fclose(*fp2);
      if(fp3 && *fp3)
         fclose(*fp3);
      return 0;
   }
   return 1;
}
```

```
int contarCamposYVerifLineaTexto(const char *linea, int cant, const char *msj)
{
   const char *aux = linea - 1;
   int        cantCampos = 0;

   if(strchr(linea, '\n') == NULL)
   {
      fprintf(stderr, "ERROR - falta fin de linea (%s).\n", msj);
      return 0;
   }
   while((aux = strpbrk(aux + 1, "\t\n")) != NULL)
      cantCampos++;
   if(cant != cantCampos)
   {
      fprintf(stderr, "ERROR - cantidad de campos (%s).\n", msj);
      return 0;
   }
   return 1;
}


int msjError(const char *msj)
{
   fprintf(stderr, "ERROR - %s.\n", msj);
   return 1;
}
```

lista.h // es común a varios programas

```
/* --------------------------------o---x---o--------------------------------
 *         listaD.h     LISTA DINÁMICA CIRCULAR
 * --------------------------------o---x---o-------------------------------- */

#ifndef LISTA_H_
#define LISTA_H_

#include <stdio.h>
#include <stdlib.h>
#include <string.h>


#define SIN_MEM     0
#define TODO_BIEN   1
#define CLA_DUP     2
```

```
typedef struct sNodo
{
   void         *info;
   unsigned       tamInfo;
   struct sNodo   *sig,
                  *ant;
} tNodo;
typedef tNodo *tLista;


void crearLista(tLista *p);

int  vaciarLista(tLista *p);

int  listaVacia(const tLista *p);

int  listaLlena(const tLista *p, unsigned cantBytes);

int  insertarAlFinal(tLista *p, const void *d, unsigned cantBytes);

int  insertarAlComienzo(tLista *p, const void *d, unsigned cantBytes);

int  mostrarDeIzqADer(const tLista *p,
                 void(*mostrar)(const void *, FILE *), FILE *fp);

int  mostrarDeDerAIzq(const tLista *p,
                 void(*mostrar)(const void *, FILE *), FILE *fp);

int  insertarEnOrden(tLista *p, const void *d, unsigned cantBytes,
                 int (*comparar)(const void *, const void *),
                 int (*acumular)(void **, unsigned *,
                           const void *, unsigned));

void ordenarLista(tLista *p, int (*comparar)(const void *, const void *));

int sacarPrimero(tLista *p, void *d, unsigned cantBytes);

int contarParesElimNoPares(tLista *p,
                     int (*comparar)(const void *, const void *),
                     void (*mostrar)(FILE *, const void *),
                     FILE *fpErrores);
#endif // LISTA_H_
```

lista.c // es común a varios programas

```
/* ------------------------------o---x---o------------------------------
```

```
 *        listaD.c     LISTA DINÁMICA CIRCULAR
 * ---------------------------------o---x---o------------------------------ */

#include "../50-listaDoblEnlaz/lista.h"

void crearLista(tLista *p)
{
   *p = NULL;
}


int  vaciarLista(tLista *p)
{
   int     cant = 0;
   tNodo  *act = *p;

   if(act)
   {
      while(act->ant)
         act = act->ant;
      while(act)
      {
         tNodo *aux = act->sig;

         free(act->info);
         free(act);
         act = aux;
         cant++;
      }
      *p = NULL;
   }
   return cant;
}


int  listaVacia(const tLista *p)
{
   return *p == NULL;
}


int  listaLlena(const tLista *p, unsigned cantBytes)
{
   tNodo  *nue = (tNodo *)malloc(sizeof(tNodo));
   void   *aux = malloc(cantBytes);
```

```
   free(nue);
   free(aux);
   return aux == NULL || nue == NULL;
}


int  insertarAlFinal(tLista *p, const void *d, unsigned cantBytes)
{
   tNodo  *act = *p,
          *nue;

   if(act)
      while(act->sig)
         act = act->sig;
   if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
     (nue->info = malloc(cantBytes)) == NULL)
   {
      free(nue);
      return 0;
   }
   memcpy(nue->info, d, cantBytes);
   nue->tamInfo = cantBytes;
   nue->sig = NULL;
   nue->ant = act;
   if(act)
      act->sig = nue;
   *p = nue;
   return 1;
}


int  insertarAlComienzo(tLista *p, const void *d, unsigned cantBytes)
{
   tNodo  *act = *p,
          *nue;

   if(act)
      while(act->ant)
         act = act->ant;
   if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
     (nue->info = malloc(cantBytes)) == NULL)
   {
      free(nue);
      return 0;
   }
   memcpy(nue->info, d, cantBytes);
```

```
   nue->tamInfo = cantBytes;
   nue->sig = act;
   nue->ant = NULL;
   if(act)
      act->ant = nue;
   *p = nue;
   return 1;
}


int  mostrarDeIzqADer(const tLista *p,
                      void(*mostrar)(const void *, FILE *), FILE *fp)
{
   tNodo  *act = *p;
   int     cant = 0;

   if(act)
   {
      mostrar(NULL, fp);
      while(act->ant)
         act = act->ant;
      while(act)
      {
         mostrar(act->info, fp);
         act = act->sig;
         cant++;
      }
   }
   return cant;
}


int  mostrarDeDerAIzq(const tLista *p,
                      void(*mostrar)(const void *, FILE *), FILE *fp)
{
   tNodo  *act = *p;
   int     cant = 0;

   if(act)
   {
      mostrar(NULL, fp);
      while(act->sig)
         act = act->sig;
      while(act)
      {
         mostrar(act->info, fp);
```

```
        act = act->ant;
        cant++;
      }
   }
   return cant;
}


int  insertarEnOrden(tLista *p, const void *d, unsigned cantBytes,
                int (*comparar)(const void *, const void *),
                int (*acumular)(void **, unsigned *,
                        const void *, unsigned))
{
   tNodo  *act = *p,
        *sig,
        *ant,
        *nue;

   if(act == NULL)
   {
      ant = NULL;
      sig = NULL;
   }
   else
   {
      int aux;
      while(act->sig && comparar(act->info, d) < 0)
         act = act->sig;
      while(act->ant && comparar(act->info, d) > 0)
         act = act->ant;
      aux = comparar(act->info, d);
      if(aux == 0)
      {
         *p = act;
         if(acumular)
            if(acumular(&act->info, &act->tamInfo, d, cantBytes) == 0)
               return SIN_MEM;
         return CLA_DUP;
      }
      if(aux < 0)
      {
         ant = act;
         sig = act->sig;
      }
      else
      {
```

```
            ant = act->ant;
            sig = act;
         }
      }
      if((nue = (tNodo *)malloc(sizeof(tNodo))) == NULL ||
        (nue->info = malloc(cantBytes)) == NULL)
      {
         free(nue);
         return SIN_MEM;
      }
      memcpy(nue->info, d, cantBytes);
      nue->tamInfo = cantBytes;
      nue->sig = sig;
      nue->ant = ant;
      if(sig)
         sig->ant = nue;
      if(ant)
         ant->sig = nue;
      *p = nue;
      return TODO_BIEN;
}


void ordenarLista(tLista *p, int (*comparar)(const void *, const void *))
{
      tNodo  *act = *p,
             *sup = NULL,
             *inf = NULL;
      int     marca = 1;

      if(act == NULL)
         return;
      while(act->ant)
         act = act->ant;
      while(marca)
      {
         marca = 0;
         while(act->sig != sup)
         {
            if(comparar(act->info, act->sig->info) > 0)
            {
               void      *inf = act->info;
               unsigned    tam = act->tamInfo;

               marca = 1;
               act->info = act->sig->info;
```

```
            act->sig->info = inf;
            act->tamInfo = act->sig->tamInfo;
            act->sig->tamInfo = tam;
         }
         act = act->sig;
      }
      sup = act;
      while(act->ant != inf)
      {
         if(comparar(act->info, act->ant->info) < 0)
         {
            void      *inf = act->info;
            unsigned    tam = act->tamInfo;

            marca = 1;
            act->info = act->ant->info;
            act->ant->info = inf;
            act->tamInfo = act->ant->tamInfo;
            act->ant->tamInfo = tam;
         }
         act = act->ant;
      }
      inf = act;
   }
}


int sacarPrimero(tLista *p, void *d, unsigned cantBytes)
{
   tNodo     *aux = *p;

   if(aux == NULL)
      return 0;
   while(aux->ant)
      aux = aux->ant;
   memcpy(d, aux->info, cantBytes <= aux->tamInfo ? cantBytes : aux->tamInfo);
   if(aux->sig)
      aux->sig->ant = NULL;
   *p = aux->sig;
   free(aux->info);
   free(aux);
   return 1;
}

/** elimina los nodos que no coincide una clave en el nodo inmediatamente a
 **     continuación, devolviendo cuántos quedan en la lista
```

```
 **/
int contarParesElimNoPares(tLista *p,
                           int (*comparar)(const void *, const void *),
                           void (*mostrar)(FILE *, const void *),
                           FILE *fpErrores)
{
   tNodo     *pri = *p,
             *sig;
   int        cantPares = 0;

   if(pri == NULL)
      return 0;
   while(pri->ant)
      pri = pri->ant;
   sig = pri->sig;
   while(sig)
   {
      if(comparar(pri->info, sig->info))
      {
         if(*p == pri)
         {
            if(sig)
               *p = sig;
            else
               *p = pri->ant;
         }
         mostrar(fpErrores, pri->info);
         sig->ant = pri->ant;
         if(pri->ant)
            pri->ant->sig = sig;
         free(pri);
         pri = sig;
         if(sig)
            sig = sig->sig;
      }
      else
      {
         pri = sig->sig;
         if(pri)
            sig = pri->sig;
         else
            sig = NULL;
         cantPares++;
      }
   }
   if(*p && (*p)->ant == NULL && (*p)->sig == NULL)
```

```
   {
      free(*p);
      *p = NULL;
   }
   return cantPares;
}
```

residuos.h

```
#ifndef RESIDUOS_H_
#define RESIDUOS_H_

#include <stdio.h>
#include <string.h>


#include "../01-funciones/funciones.h"
#include "../50-listaDoblEnlaz/lista.h"

#define FABRICA       "fabrica"
#define TITULOS_PSTOK  \
        "Destino\tDemanda M1P\tRecuperacion M1P\tResiduos M1P\tDemanda M2P"\
        "\tRecuperacion M2P\tResiduos M2P\tDemanda M3P\tRecuperacion M3P"  \
        "\tResiduos M3P\tDemanda M4P\tRecuperacion M4P\tResiduos M4P"     \
        "\tDemanda M5P\tRecuperacion M5P\tResiduos M5P\tDemanda M6P"      \
        "\tRecuperacion M6P\tResiduos M6P\tDemanda M7P\tRecuperacion M7P"  \
        "\tResiduos M7P\tDemanda M8P\tRecuperacion M8P\tResiduos M8P"     \
        "\tDemanda M9P\tRecuperacion M9P\tResiduos M9P\tDemanda M10P"     \
        "\tRecuparacion M10P\tResiduos M10P\tDemanda M11P"               \
        "\tRecuperacion M11P\tResiduos M11P\tDemanda M12P"               \
        "\tRecuperacion M12P\tResiduos M12P\n"

typedef struct
{
   char       palletera[51];
   int        esDema;
   int        cantPalle[12];
} tRecuDema;

int procesarArchivos(const char *recup, const char *deman, const char *pStok,
                     const char *resid, float param);



#endif // RESIDUOS_H_
```

residuos.c

```
#include "residuos.h"


int trozarRecuDema(const char *linea, tRecuDema *pRecuDema, float param)
{
   char   *aux = strchr(linea, '\t');
   int     cant = (int)(aux - linea),
           ciclo = 0,
           retErr;

   if(aux == NULL)
      return 0;
   if(cant >= sizeof(pRecuDema->palletera))
      cant = sizeof(pRecuDema->palletera) - 1;
   *pRecuDema->palletera = '\0';
   strncat(pRecuDema->palletera, linea, cant);
   retErr = strlen(pRecuDema->palletera);
   if(pRecuDema->esDema == 1)
      aux = strchr(aux + 1, '\t');
   while(aux && retErr && ciclo < 12)
   {
      retErr = sscanf(aux, "\t%d", &pRecuDema->cantPalle[ciclo]);
      pRecuDema->cantPalle[ciclo] *= param;
      ciclo++;
      aux = strchr(aux + 1, '\t');
   }
   return retErr != 0;
}


int compXPallDema(const void *d1, const void *d2)
{
   tRecuDema  *dd1 = (tRecuDema *)d1,
              *dd2 = (tRecuDema *)d2;
   int        cmp = strcasecmp(dd1->palletera, dd2->palletera);

   if(cmp)
      return cmp;
   return dd1->esDema - dd2->esDema;
}
```

```
int compararXPaletera(const void *d1, const void *d2)
{
   return strcasecmp(((tRecuDema *)d1)->palletera,
                ((tRecuDema *)d2)->palletera);
}


void mostrarRecuDema(FILE *fpErr, const void *d1)
{
   tRecuDema  *dd1 = (tRecuDema *)d1;
   int ciclo;

   fprintf(fpErr, "ERROR CON:\t\"%s\"\t\"%s\"",
         dd1->palletera, dd1->esDema == 1 ? "deman" : "recup");
   for(ciclo = 0; ciclo < 12; ciclo++)
      fprintf(fpErr, "\t%d", dd1->cantPalle[ciclo]);
   fprintf(fpErr, "\n");
}


int cargarRecuDema(tRecuDema *recuDema,
              tLista *lista, FILE *fpRecuDema, float param)
{
   char    linea[500];

   fgets(linea, sizeof(linea), fpRecuDema);
   while(fgets(linea, sizeof(linea), fpRecuDema))
   {
      if(recuDema->esDema == 0)
         if(!contarCamposYVerifLineaTexto(linea, 13, "en recupero"))
            continue;
      if(recuDema->esDema == 1)
         if(!contarCamposYVerifLineaTexto(linea, 14, "en demanda"))
            continue;
      if(!strncasecmp(linea, FABRICA, 7))
         continue;
      if(trozarRecuDema(linea, recuDema, param))
         insertarEnOrden(lista, recuDema, sizeof(*recuDema),
                     compXPallDema, NULL);
   }
   return 1;
}


int calcularResiduos(int *residuos, int *totFinal,
                tRecuDema *recup, tRecuDema *deman)
```

```
{
   int     ciclo;

   residuos[0] = recup->cantPalle[0] - deman->cantPalle[0];
   totFinal[0] += deman->cantPalle[0];
   totFinal[1] += recup->cantPalle[0];
   totFinal[2] += residuos[0];
   for(ciclo = 1; ciclo < 12; ciclo++)
   {
      residuos[ciclo] = recup->cantPalle[ciclo] -
                        deman->cantPalle[ciclo] + residuos[ciclo - 1];
      totFinal[ciclo * 3] += deman->cantPalle[ciclo];
      totFinal[ciclo * 3 + 1] += recup->cantPalle[ciclo];
      totFinal[ciclo * 3 + 2] += residuos[ciclo];
   }
   return ciclo;
}


int imprimirParaStock(FILE *fpPStok, FILE *fpResid, tLista *lista)
{
   tRecuDema   recuDema_1,
               recuDema_2;
   int         ciclo,
               cantPares,
               residuos[12] = {},
               totFinal[36] = {};

   /// contar pares de nodos
   cantPares = contarParesElimNoPares(lista, compararXPaletera,
                                      mostrarRecuDema, stderr);
   if(listaVacia(lista))
      return 0;
   fprintf(fpPStok, TITULOS_PSTOK);
   while(sacarPrimero(lista, &recuDema_1, sizeof(recuDema_1)) &&
         sacarPrimero(lista, &recuDema_2, sizeof(recuDema_2)))
   {
      calcularResiduos(residuos, totFinal, &recuDema_1, &recuDema_2);
      fprintf(fpPStok, "%s", recuDema_1.palletera);
      for(ciclo = 0; ciclo < 12; ciclo++)
         fprintf(fpPStok,
                 "\t%d\t%d\t%d",
                 recuDema_2.cantPalle[ciclo],
                    recuDema_1.cantPalle[ciclo],
                    residuos[ciclo]);
      fprintf(fpPStok, "\n");
```

```
   }
   fprintf(fpPStok, "Suma");
   for(ciclo = 0; ciclo < 36; ciclo++)
      fprintf(fpPStok, "\t%d", totFinal[ciclo]);
   fprintf(fpPStok, "\n");
   fprintf(fpResid, "Destino\tSuma\n");
   for(ciclo = 0; ciclo < 12; ciclo++)
      fprintf(fpResid,
            "Residuos M%dP\t%d\n",
            ciclo + 1, totFinal[ciclo * 3 + 2]);
   return cantPares;
}


/** \brief mediante  cargarRecuDema  carga en orden una lista doblemente
 *      enlazada los pallets recuperados y luego la demanda de pallets generando
 *      pares de (demanda, recup), (demanda, recup), y finalmente mediante
 *      imprimirParaStock genera las salidas previstas
 *
 *  \param  fpRecup     archivo de entrada (pallets recuperados)
 *  \param  fpDeman     archivo de entrada (pallets demanda)
 *  \param  fpPStok     archivo de salida (pallets para stock)
 *  \param  fpResid     archivo de salida (residuos)
 *  \param  lista       lista doblemete enlazada
 *  \param  param       coeficiente de merma para pallets recuperados
 *  \return     la cantidad de líneas generadas
 *
 */
int generarOutpParaStock(FILE *fpRecup, FILE *fpDeman, FILE *fpPStok,
                  FILE *fpResid, tLista *lista, float param)
{
   tRecuDema   recuDema;

   recuDema.esDema = 0;
   cargarRecuDema(&recuDema, lista, fpRecup, param);
   recuDema.esDema = 1;
   cargarRecuDema(&recuDema, lista, fpDeman, 1.);
   return imprimirParaStock(fpPStok, fpResid, lista);
}


/** \brief abre los archivos y devuelve cuántas filas (líneas de texto) se
 *      generan en los archivos de salida (sin contar encabezados ni totales).
 *
 *  \param  recup       nombre archivo de entrada (pallets recuperados)
 *  \param  deman       nombre archivo de entrada (pallets demanda)
```

```
 *  \param  pStok      nombre archivo de salida (pallets para stock)
 *  \param  resid      nombre archivo de salida (residuos)
 *  \return retVal     cantidad  de líneas generadas
 *
 */
int procesarArchivos(const char *recup, const char *deman, const char *pStok,
                const char *resid, float param)
{
   int     retVal = 0;
   FILE   *fpRecup,
          *fpDeman,
          *fpPStok,
          *fpResid;
   tLista lista;

   crearLista(&lista);
   if(!abrirLosArchivos(&fpRecup, recup, "rt",
                   &fpDeman, deman, "rt",
                   &fpPStok, pStok, "wt"))
      goto SALIDA;
   if(!abrirLosArchivos(NULL, NULL, NULL, NULL, NULL, NULL,
                   &fpResid, resid, "wt"))
      goto CERRAR;
   retVal = generarOutpParaStock(fpRecup, fpDeman, fpPStok, fpResid,
                            &lista, param);
   fclose(fpResid);
CERRAR :
   fclose(fpRecup);
   fclose(fpDeman);
   fclose(fpPStok);
SALIDA :
   return retVal;
}
```

2da etapa: proyecto "CostosTranspPaleterasYFabricYRecup.cbp"

main.h

```
#ifndef MAIN_H_
#define MAIN_H_

#include <stdio.h>
#include <stdlib.h>

#include "costos.h"
```

```
#define    ES_FABRICA     1

#define    ARCH_COSTO     "Input.Costos.Trans.Origen.Cliente.csv"

#define    ARCH_DEMAN     "Input.Demanda.csv"
#define    ARCH_TDPAL     "Output.Costos.Transporte.Demanda.Paleteras.csv"
#define    ARCH_TDFAB     "Output.Costos.Transporte.Demanda.Fabricantes.csv"

#define    ARCH_RECUP     "Input.Recuperacion.csv"
#define    ARCH_TRECU     "Output.Costo.Transporte.Recuperacion.csv"
//**********
#define    ARCH_C_PAL     "Input.Costos.Pallets.csv"
// con     ARCH_RECUP
#define    ARCH_C_INS     "Output.Costo.Inspeccion.csv"
#define    ARCH_C_REP     "Output.Costo.Reparacion.csv"
// con     ARCH_DEMAN
#define    ARCH_C_CPR     "Output.Costo.Compras.csv"

//
#define    ARCH_C_TOT     "Output.Costos.Totales.csv"
#define    ARCH_C_T_A     "Output.Costo.Total.Anual.csv"

double recuperarParametro(int cArg, char *argumento);


#endif // MAIN_H_
```

main.c

```
#include "main.h"

int main(int cArg, char **vArg)
{
   int       retVal = 1;
   double      param = recuperarParametro(cArg, *(vArg + 1));
   char       titFinal[7][19] = { "CostoTransRecup",
                                  "CostoInspeccion",
                                  "CostoReparacion",
                                  "CostoCompras",
                                  "CostoTransdemFab",
                                  "CostoTransDemPal",
                                  "CostoTotal" };
   double      totales[7][12] = { 0 };

   /** transporte demanda paletera **/
```

```
   fprintf(stdout,
         "Procesando los archivos:\n\"%s\"\n\"%s\".\n",
         ARCH_COSTO, ARCH_DEMAN);
   retVal = procesarArchivos(ARCH_COSTO, ARCH_DEMAN, ARCH_TDPAL, !ES_FA-
BRICA,
                     totales[5]);
   if(retVal == 0)                              /// ^^^^^^^^^^
      fprintf(stderr, "ERROR - procesando archivos.\n");
   else
      fprintf(stdout,
            "Procesamiento exitoso.\n"
            "Se generaron %d filas en el archivo \"%s\".\n",
            retVal, ARCH_TDPAL);

   /** transporte demanda fabricante **/
   fprintf(stdout,
         "Procesando los archivos:\n\"%s\"\n\"%s\".\n",
         ARCH_COSTO, ARCH_DEMAN);
   retVal = procesarArchivos(ARCH_COSTO, ARCH_DEMAN, ARCH_TDFAB, ES_FAB-
RICA,
                     totales[4]);
   if(retVal == 0)                              /// ^^^^^^^^^^
      fprintf(stderr, "ERROR - procesando archivos.\n");
   else
      fprintf(stdout,
            "Procesamiento exitoso.\n"
            "Se generaron %d filas en el archivo \"%s\".\n",
            retVal, ARCH_TDFAB);

   /** transporte recuperacion **/
   fprintf(stdout,
         "Procesando los archivos:\n\"%s\"\n\"%s\".\n",
         ARCH_COSTO, ARCH_RECUP);
   retVal = procesarArchivos(ARCH_COSTO, ARCH_RECUP, ARCH_TRECU, 2,
                     totales[0]);
   if(retVal == 0)                              /// ^^^^^^^^^^
      fprintf(stderr, "ERROR - procesando archivos.\n");
   else
      fprintf(stdout,
            "Procesamiento exitoso.\n"
            "Se generaron %d filas en el archivo \"%s\".\n",
            retVal, ARCH_TRECU);

   /** costo inspeccion **/
   fprintf(stdout,
         "Procesando los archivos:\n\"%s\"\n\"%s\".\n",
```

```
      ARCH_C_PAL, ARCH_RECUP);
retVal = procesarArchiDos(ARCH_C_PAL, ARCH_RECUP, ARCH_C_INS,
                  "INSPECCION", 1.0, 1,
                  totales[1]);
if(retVal == 0)     ///  ^^^^^^^^^^^^^^^^^
   fprintf(stderr, "ERROR - procesando archivos.\n");
else
   fprintf(stdout,
         "Procesamiento exitoso.\n"
         "Se generaron %d filas en el archivo \"%s\".\n",
         retVal, ARCH_C_INS);

/** costo recuperacion **/
fprintf(stdout,
      "Procesando los archivos:\n\"%s\"\n\"%s\".\n",
      ARCH_C_PAL, ARCH_RECUP);
retVal = procesarArchiDos(ARCH_C_PAL, ARCH_RECUP, ARCH_C_REP,
                  "REPARACION", param, 2,
                  totales[2]);
if(retVal == 0)     ///  ^^^^^^^^^^^^^^^^^^   acá se utiliza el parámetro
   fprintf(stderr, "ERROR - procesando archivos.\n");
else
   fprintf(stdout,
         "Procesamiento exitoso.\n"
         "Se generaron %d filas en el archivo \"%s\".\n",
         retVal, ARCH_C_REP);

/** costo compras **/
fprintf(stdout,
      "Procesando los archivos:\n\"%s\"\n\"%s\".\n",
      ARCH_C_PAL, ARCH_DEMAN);
retVal = procesarArchiDos(ARCH_C_PAL, ARCH_DEMAN, ARCH_C_CPR,
                  "NUEVO", 1.0, 3,
                  totales[3]);
if(retVal == 0)     ///  ^^^^^^^^^^^^^^^^^^   acá se utiliza el parámetro
   fprintf(stderr, "ERROR - procesando archivos.\n");
else
   fprintf(stdout,
         "Procesamiento exitoso.\n"
         "Se generaron %d filas en el archivo \"%s\".\n",
         retVal, ARCH_C_CPR);

/** costos total y total anual **/
fprintf(stdout,
      "generando los archivos:\n\"%s\"\n\"%s\".\n",
      ARCH_C_TOT, ARCH_C_T_A);
```

```
   retVal = procesarArchiTre(ARCH_C_TOT, ARCH_C_T_A, titFinal, totales);
   if(retVal == 0)
      fprintf(stderr, "ERROR - procesando archivos.\n");
   else
      fprintf(stdout,
            "Procesamiento exitoso.\n"
            "Se generaron %d filas en el archivo \"%s\".\n",
            retVal, ARCH_C_TOT);

   return !retVal;
}


double recuperarParametro(int cArg, char *argumento)
{
   double   param = argumento && *argumento ?
                  (int)(atof(argumento) * 100) / 100. : 1.0;

   if(param < 0.009 || param > 1.0)
      param = 1.0;
   return param;
}
```

costos.h

```
#ifndef RESIDUOS_H_
#define RESIDUOS_H_

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "../01-funciones/funciones.h"


#define FABRICA       "fabrica"
#define TITULOS_OUTP_PALE  \
         "Destino\tCosto Transp\tCosto Trans Demanda M1P\t"          \
         "Costo Trans Demanda M2P\tCosto Trans Demanda M3P\t"        \
         "Costo Trans Demanda M4P\tCosto Trans Demanda M5P\t"        \
         "Costo Trans Demanda M6P\tCosto Trans Demanda M7P\t"        \
         "Costo Trans Demanda M8P\tCosto Trans Demanda M9P\t"        \
         "Costo Trans Demanda M10P\tCosto Trans Demanda M11P\t"      \
         "Costo Trans Demanda M12P\n"

#define TITULOS_OUTP_FABR  \
```

```
        "Origen\tCosto Trans Fabricantes\tCosto Trans Fabricantes M1P\t"\
        "Costo Trans Fabricantes M2P\tCosto Trans Fabricantes M3P\t"   \
        "Costo Trans Fabricantes M4P\tCosto Trans Fabricantes M5P\t"   \
        "Costo Trans Fabricantes M6P\tCosto Trans Fabricantes M7P\t"   \
        "Costo Trans Fabricantes M8P\tCosto Trans Fabricantes M9P\t"   \
        "Costo Trans Fabricantes M10P\tCosto Trans Fabricantes M11P\t" \
        "Costo Trans Fabricantes M12P\n"

#define TITULOS_OUTP_RECU  \
        "Destino\tCostos Trans\tM1P\tM2P\tM3P\tM4P\tM5P\tM6P\tM7P\t"   \
        "M8P\tM9P\tM10P\tM11P\tM12P\tCostoRM1P\tCostoRM2P\tCostoRM3P\t" \
        "CostoRM4P\tCostoRM5P\tCostoRM6P\tCostoRM7P\tCostoRM8P\t"      \
        "CostoRM9P\tCostoRM10P\tCostoRM11P\tCostoRM12P\n"

#define TITULOS_INSPECCION \
        "Destino\tCostoInspM1P\tCostoInspM2P\tCostoInspM3P\t"         \
        "CostoInspM4P\tCostoInspM5P\tCostoInspM6P\tCostoInspM7P\t"     \
        "CostoInspM8P\tCostoInspM9P\tCostoInspM10P\tCostoinspM11P\t"   \
        "CostoInspM12P\n"

#define TITULOS_REPARACION \
        "Destino\tCostoRoturaM1P\tCostoRoturaM2P\tCostoRoturaM3P\t"    \
        "CostoRoturaM4P\tCostoRoturaM5P\tCostoRoturaM6P\t"            \
        "CostoRoturaM7P\tCostoRoturaM8P\tCostoRoturaM9P\t"            \
        "CostoRoturaM10P\tCostoRoturaM11P\tCostoRoturaM12P\n"

#define TITULOS_COMPRAS    \
        "Origen\tCostoCompraM1P\tCostoCompraM2P\tCostoCompraM3P\t"     \
        "CostoCompraM4P\tCostoCompraM5P\tCostoCompraM6P\t"            \
        "CostoCompraM7P\tCostoCompraM8P\tCostoCompraM9P\t"            \
        "CostoCompraM10P\tCostoCompraM11P\tCostoCompraM12P\n"

#define TITULOS_C_TOTAL    \
        "Costo\tM1P\tM2P\tM3P\tM4P\tM5P\tM6P\tM7P\tM8P\tM9P\tM10P\t"   \
        "M11P\tM12P\n"

#define TITULOS_C_TOT_A    \
        "Mes\tCostoMensual\n"



int procesarArchivos(const char *costo, const char *deman, const char *salid,
                int esFabri, double *total);

int procesarArchiDos(const char *costo, const char *ent_2, const char *salid,
                const char *queCosto, double param, int queTitulo,
```

```
                double *total);

int procesarArchiTre(const char *salTotal, const char *salTotAn,
                char titFinal[7][19], double totales[7][12]);
#endif // RESIDUOS_H_
```

costos.c

```
#include "costos.h"

/*  para  <Input.Recuperacion>  esFabri = 2 porque tiene una columna menos */
int imprimirOutp(const char *linea, double costo, FILE *fpSalid, double *total,
              int esFabri)
{
   char   *aux = strchr(linea, '\t');

   fprintf(fpSalid, "%.*s\t%.13lf", (int)(aux - linea), linea, costo);
   if(esFabri == 2)
   {
      char   *fin = strchr(linea, '\n');

      fprintf(fpSalid, "%.*s", (int)(fin - aux), aux);
      aux = (char *)linea;
   }
   while((aux = strchr(aux + 1, '\t')) != NULL)
   {
      int     cantPallet,
             posPunto,
             posSigno;
      double  precio;

      sscanf(aux, "%d", &cantPallet);
      precio = costo * cantPallet;
      fcvt(precio, 16, &posPunto, &posSigno);
      fprintf(fpSalid, "\t%.*lf", 15 - posPunto, precio);
      *total += precio;
      total++;
   }
   fprintf(fpSalid, "\n");
   return 1;
}


int buscarCosto(double *costo, FILE *fpCosto, const char *linea)
{
```

```
   char    lineaCosto[500],
         *aux;

   rewind(fpCosto);
   while(fgets(lineaCosto, sizeof(lineaCosto), fpCosto))
   {
      aux = strchr(lineaCosto, '\t');
      if(aux == NULL)
         return 0;
      if(strncasecmp(linea, lineaCosto, (size_t)(aux - lineaCosto)) == 0)
      {
         if(sscanf(aux + 1, "%lf", costo) == 1)
            return 1;
         return 0;
      }
   }
   return 0;
}

/*  para  <Input.Recuperacion>  esFabri = 2 no tiene efecto acá */
int generarOutp(FILE *fpCosto, FILE *fpDeman, FILE *fpSalid, int esFabri,
            double *total)
{
   char       linea[500];
   int        cant = 0,
              ciclo;
   double     costo;

   if(fgets(linea, sizeof(linea), fpDeman) == NULL)
      goto SALIDA;
   fprintf(fpSalid,
         esFabri == 1 ? TITULOS_OUTP_FABR :
            esFabri == 0 ? TITULOS_OUTP_PALE : TITULOS_OUTP_RECU);
   while(fgets(linea, sizeof(linea), fpDeman))
   {
      if(esFabri == 0 && strncasecmp(linea, FABRICA, 7) == 0)
         continue;
      if(esFabri == 1 && strncasecmp(linea, FABRICA, 7) != 0)
         continue;
      if(buscarCosto(&costo, fpCosto, linea))
      {
         imprimirOutp(linea, costo, fpSalid, total, esFabri);
         cant++;
      }
   }
   if(cant)
```

```
   {
      fprintf(fpSalid,
            "%s",
            esFabri != 2 ? "Suma\t" : "\t\t\t\t\t\t\t\t\t\t\t\t");
      for(ciclo = 0; ciclo < 12; ciclo++)
         fprintf(fpSalid, "\t%.2lf", total[ciclo]);
      fprintf(fpSalid, "\n");
   }
SALIDA :
   return cant;
}

/*  para  <Input.Demenda>
 **     solo paleteras      si esFabri = 0
 **     solo fabricantes    si esFabri = 1
 ** para  <Input.Recuperacion>
 **     son TODOS paleteras si esFabri = 2  - tiene una columna menos */
int procesarArchivos(const char *costo, const char *deman, const char *salid,
                     int esFabri, double *total)
{
   int     retVal = 0;
   FILE   *fpCosto,
          *fpDeman,
          *fpSalid;

   if(!abrirLosArchivos(&fpCosto, costo, "rt",
                        &fpDeman, deman, "rt",
                        &fpSalid, salid, "wt"))
      goto SALIDA;
   retVal = generarOutp(fpCosto, fpDeman, fpSalid, esFabri, total);
   fclose(fpCosto);
   fclose(fpDeman);
   fclose(fpSalid);
SALIDA :
   return retVal;
}

/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
int generarOutpDos(FILE *fpEntPp, FILE *fpSalid, double param, int queTitulo,
                   double *total)
{
   char    linea[500],
          *aux;
   int     cant = 0,
          cantPall,
          ciclo;
```

```
   if(!fgets(linea, sizeof(linea), fpEntPp))
      return cant;
   fprintf(fpSalid,
         "%s",
         queTitulo == 1 ? TITULOS_INSPECCION :
            queTitulo == 2 ? TITULOS_REPARACION : TITULOS_COMPRAS);
   while(fgets(linea, sizeof(linea), fpEntPp))
   {
      if(queTitulo == 3 && strncasecmp(linea, FABRICA, 2) != 0)
         continue;
      aux = strchr(linea, '\t');
      fprintf(fpSalid, "%.*s", (int)(aux - linea), linea);
      if(queTitulo == 3)
         aux = strchr(aux + 1, '\t');
      ciclo = 0;
      while(aux && *aux == '\t')
      {
         sscanf(aux, "%d", &cantPall);
         total[ciclo++] += cantPall * param;
         fprintf(fpSalid, "\t%ld", (long)(cantPall * param));
         aux = strchr(aux + 1, '\t');
      }
      fprintf(fpSalid, "\n");
      cant++;
   }
   if(cant)
   {
      fprintf(fpSalid,
            "%s",
            queTitulo == 1 ? "CostoInspeccion" :
               queTitulo == 2 ? "CostoReparacion" : "CostoCompra");
      for(ciclo = 0; ciclo < 12; ciclo++)
         fprintf(fpSalid, "\t%ld", (long)total[ciclo]);
      fprintf(fpSalid, "\n");
   }
   return cant;
}


int obtenerCoeficiente(FILE *fpCosto, const char *queCosto, double *coefi)
{
   char    linea[500],
         *aux;

   while(fgets(linea, sizeof(linea), fpCosto) && !strstr(linea, queCosto))
```

```
      ;       /** <== linea vacía **/
   if(!feof(fpCosto) && (aux = strchr(linea, '\t')) != NULL &&
     sscanf(aux, "%lf", coefi))
      return 1;
   return 0;
}

/* si   unoATres == 1   usa costo inspección  con  <Input.Costos.Pallets>
 */
int procesarArchiDos(const char *costo, const char *entPp, const char *salid,
                const char *queCosto, double param, int queTitulo,
                double *total)
{
   int     retVal = 0;
   double  coefi;
   FILE   *fpCosto,
          *fpEntPp,
          *fpSalid;

   if(!abrirLosArchivos(&fpCosto, costo, "rt",
                   &fpEntPp, entPp, "rt",
                   &fpSalid, salid, "wt"))
      goto SALIDA;
   if(obtenerCoeficiente(fpCosto, queCosto, &coefi))
      retVal = generarOutpDos(fpEntPp, fpSalid, param * coefi, queTitulo,
                         total);
   fclose(fpCosto);
   fclose(fpEntPp);
   fclose(fpSalid);
SALIDA :
   return retVal;
}

/**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**//**/
int generarOutpTre(FILE *fpSalTotal,
               char titFinal[7][19], double totales[7][12])
{
   int     fi,
           co;

   fprintf(fpSalTotal, "%s", TITULOS_C_TOTAL);
   for(fi = 0; fi < 6; fi++)
   {
      fprintf(fpSalTotal, "%s", titFinal[fi]);
      for(co = 0; co < 12; co++)
      {
```

```
            fprintf(fpSalTotal, "\t%.2lf", totales[fi][co]);
            totales[6][co] += totales[fi][co];
        }
        fprintf(fpSalTotal, "\n");
    }
    fprintf(fpSalTotal, "%s", titFinal[fi]);
    for(co = 0; co < 12; co++)
        fprintf(fpSalTotal, "\t%.2lf", totales[fi][co]);
    fprintf(fpSalTotal, "\n");
    return fi;
}


int generarOutpCua(FILE *fpSalTotAn, double *totales)
{
    static const char *meses[13] = { "Enero", "Febrero", "Marzo", "Abril",
                "Mayo", "Junio", "Julio", "Agosto", "Septiembre",
                "Octubre", "Noviembre", "Diciembre", "CostoAnual" };
    int         fi = 0;
    double      totAn = 0.0;

    fprintf(fpSalTotAn, "%s", TITULOS_C_TOT_A);
    while(fi < 12)
    {
        fprintf(fpSalTotAn, "%s\t%.2lf\n", meses[fi], *totales);
        totAn += *totales;
        totales++;
        fi++;
    }
    fprintf(fpSalTotAn, "%s\t%.2lf\n", meses[fi], totAn);
    return fi;
}


int procesarArchiTre(const char *salTotal, const char *salTotAn,
                char titFinal[7][19], double totales[7][12])
{
    int     retVal = 0;
    FILE    *fpSalTotal,
            *fpSalTotAn;

    if(!abrirLosArchivos(NULL, NULL, NULL,
                    NULL, NULL, NULL,
                    &fpSalTotal, salTotal, "wt"))
        goto SALIDA;
```

```
   if(!abrirLosArchivos(NULL, NULL, NULL,
                        NULL, NULL, NULL,
                        &fpSalTotAn, salTotAn, "wt"))
      goto SALIDA;
   retVal = generarOutpTre(fpSalTotal, titFinal, totales);
   if(generarOutpCua(fpSalTotAn, totales[6]) != 12)
      return 0;
   fclose(fpSalTotal);
   fclose(fpSalTotAn);
SALIDA :
   return retVal;
}
```

funciones.h

((ver antes))

funciones.c

((ver antes))

3ra etapa: proyecto "GenerarInput.cbp"

main.h

```
#ifndef MAIN_H_
#define MAIN_H_

#include <stdio.h>
#include <stdlib.h>

#include "dem_recu.h"


#define    ES_FABRICA     1

#define    ARCH_DEMAN     "Input.Demanda.csv"
#define    ARCH_mMDEM     "Input.min.Max.Demanda.csv"
#define    ARCH_DEM_N     "Input.DemandaN.csv"
#define    ARCH_mMD_N     "Input.min.Max.DemandaN.csv"

#define    ARCH_RECUP     "Input.Recuperacion.csv"
#define    ARCH_mMCOS     "Input.min.Max.Recuperacion.csv"
#define    ARCH_REC_N     "Input.RecuperacionN.csv"
#define    ARCH_mMC_N     "Input.min.Max.RecuperacionN.csv"
```

```
double recuperarParametro(int cArg, char *argumento);


#endif // MAIN_H_
```

main.c

```
#include "main.h"

int main(int cArg, char **vArg)
{
   int        retVal = 1,
              queSalida,
              semilla;
   char       *entorno = getenv("SRAND");

   semilla = entorno ? atoi(entorno) : 0;
   srand(semilla * time(NULL));
   queSalida = getenv("PAT_OUT") == NULL;

   /** demanda **/
   fprintf(stdout,
         "Procesando los archivos:\n\"%s\"\n\"%s\".\n",
         ARCH_DEMAN, ARCH_mMDEM);
   retVal = generarNuevo(ARCH_DEMAN, ARCH_mMDEM,
                  queSalida ? ARCH_DEM_N : ARCH_DEMAN, 14,
                  queSalida ? ARCH_mMD_N : ARCH_mMDEM, 4);
   if(retVal == 0)                        ///  ^^^^ cantCampos
      fprintf(stderr, "ERROR - procesando archivos.\n");
   else
      fprintf(stdout,
            "Procesamiento exitoso.\n"
            "Se generaron %d filas en el archivo \"%s\".\n",
            retVal, queSalida ? ARCH_DEM_N : ARCH_DEMAN);

   /** costos **/
   fprintf(stdout,
         "Procesando los archivos:\n\"%s\"\n\"%s\".\n",
         ARCH_RECUP, ARCH_mMCOS);
   retVal = generarNuevo(ARCH_RECUP, ARCH_mMCOS,
                  queSalida ? ARCH_REC_N : ARCH_RECUP, 13,
                  queSalida ? ARCH_mMC_N : ARCH_mMCOS, 3);
   if(retVal == 0)                        ///  ^^^^ cantCampos
      fprintf(stderr, "ERROR - procesando archivos.\n");
   else
```

```
    fprintf(stdout,
          "Procesamiento exitoso.\n"
          "Se generaron %d filas en el archivo \"%s\".\n",
          retVal, queSalida ? ARCH_REC_N : ARCH_RECUP);

  return !retVal;
}


double recuperarParametro(int cArg, char *argumento)
{
  double   param = argumento && *argumento ?
              (int)(atof(argumento) * 100) / 100. : 1.0;

  if(param < 0.009 || param > 1.0)
    param = 1.0;
  return param;
}
```

dem_recu.h

```
#ifndef RESIDUOS_H_
#define RESIDUOS_H_

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "../01-funciones/funciones.h"


#define    TIT_mM_3   "Variable\tMin\tMax"
#define    TIT_mM_4   "Variable\tn\tMin\tMax"



int generarNuevo(const char *entrad, const char *minMax, const char *salida,
          int cantCampos, const char *salida2, int cantCampos2);



#endif // RESIDUOS_H_
```

dem_recu.c

```
#include "dem_recu.h"


int grabarAzar(FILE *fpSalida, const char *aux, int min, int max,
           FILE *fpSalida2)
{
   int     nueMin = 0x7fffffff,
           nueMax = 0x00000000;

   while(aux && *aux == '\t')
   {
      int     valor = *(aux + 1) == '0' ?
                      0 :
                      rand() * rand() % (max + 1 - min) + min;
      if(valor < nueMin)
         nueMin = valor;
      if(valor > nueMax)
         nueMax = valor;
      fprintf(fpSalida, "\t%d", valor);
      aux = strchr(aux + 1, '\t');
   }
   fprintf(fpSalida2, "\t%d\t%d\n", nueMin, nueMax);
   return 1;
}


int buscarMinMax(FILE *fpMinMax, const char *clave, int tamClave,
             int *min, int *max, int cantCampos)
{
   char    linea[500],
         *aux;

   rewind(fpMinMax);
   while(fgets(linea, sizeof(linea), fpMinMax))
   {
      if(strncasecmp(linea, clave, tamClave) == 0)
      {
         aux = strchr(linea, '\t');
         if(cantCampos == 4)
            aux = strchr(aux + 1, '\t');
         sscanf(aux + 1, "%d\t%d", min, max);
         return 1;
      }
   }
   return 0;
}
```

```
int procesarEntradas(FILE *fpEntrad, FILE *fpMinMax, FILE *fpSalida,
                int cantCampos, FILE *fpSalida2, int cantCampos2)
{
   int     cant = 0,
           min,
           max;
   char    linea[500],
          *aux;

   if(fgets(linea, sizeof(linea), fpEntrad))
   {
      fprintf(fpSalida, "%s", linea);
      fprintf(fpSalida2, "%s\n", cantCampos2 == 4 ? TIT_mM_4 : TIT_mM_3);
   }
   while(fgets(linea, sizeof(linea), fpEntrad) &&
         (aux = strchr(linea, '\t')) != NULL)
   {
      if(!buscarMinMax(fpMinMax, linea, (int)(aux - linea), &min, &max,
                  cantCampos == 13 ? 3 : 4))
      {
         fprintf(stderr,
               "ERROR - min-MAX no encontrado para \"%.*s\".\n",
               (int)(aux - linea), linea);
         continue;
      }
      fprintf(fpSalida2,
            "%.*s%s",
            (int)(aux - linea), linea,
            cantCampos2 == 4 ? "\t12" : "");
      if(cantCampos == 14)
         aux = strchr(aux + 1, '\t');
      fprintf(fpSalida, "%.*s", (int)(aux - linea), linea);
      grabarAzar(fpSalida, aux, min, max, fpSalida2);
      fprintf(fpSalida, "\n");
      cant++;
   }
   return cant;
}


int generarNuevo(const char *entrad, const char *minMax, const char *salida,
             int cantCampos, const char *salida2, int cantCampos2)
{
   int     retVal = 0;
```

```
   FILE   *fpEntrad,
          *fpMinMax,
          *fpSalida,
          *fpSalida2;


   if(!abrirLosArchivos(&fpEntrad, entrad, "rt",
                        &fpMinMax, minMax, "rt",
                        &fpSalida, salida, "wt"))
      goto SALIDA;
   if(!abrirLosArchivos(NULL, NULL, NULL,
                        NULL, NULL, NULL,
                        &fpSalida2, salida2, "wt"))
   {
      fclose(fpEntrad);
      fclose(fpMinMax);
      fclose(fpSalida);
      goto SALIDA;
   }
   retVal = procesarEntradas(fpEntrad, fpMinMax, fpSalida, cantCampos,
                             fpSalida2, cantCampos2);
   fclose(fpEntrad);
   fclose(fpMinMax);
   fclose(fpSalida);
SALIDA :
   return retVal;
}
```

funciones.h

((ver antes))

funciones.c

((ver antes))

<u>4ta etapa</u>: proyecto "GenerarEntorno.cbp"

main.h

```
#ifndef MAIN_H_
#define MAIN_H_

#include <stdio.h>
#include <stdlib.h>

#include "generar.h"
```

```
#define PARAMETROS      "parametros.csv"

/**
double recuperarParametro(int cArg, char *argumento);
 **/

#endif // MAIN_H_
```

main.c

```
#include "main.h"

int main(int cArg, char **vArg)
{
   int        retVal = 1;

   if(!generarSimulacion(PARAMETROS))
      goto SALIDA;

SALIDA :
   return !retVal;
}
```

generar.h

```
#ifndef RESIDUOS_H_
#define RESIDUOS_H_

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <direct.h>

#define TEXTO_BASE      "Corridas:\t4\n"                          \
                        "Srand:\t1\n"                             \
                        "Param P\tParam R\n"                      \
                        "1.00\t1.00\n"                            \
                        "0.90\t0.80\n"                            \
                        "0.75\t0.65\n"                            \
                        "0.80\t0.70\n"

#define PROG_10         "ParaStockYSumaResiduos.exe"
#define PROG_11         "CostosTranspPaleterasYFabricYRecup.exe"
```

```
#define PROG_12        "GenerarInput.exe"


int generarSimulacion(const char *params);



#endif // RESIDUOS_H_
```

generar.c

```
#include "generar.h"


int mostrarModoUso(const char *params)
{
   FILE   *fpNue = fopen(params, "wt");

   fprintf(stderr,
         "Proceda a cargar el archivo de parametros \"%s\".\n"
         "Por ejemplo, con:\r\n",
         params);
   if(fpNue == NULL)
   {
      fprintf(stderr, "ERROR - no se pudo crear el archivo \"%s\".", params);
      return 0;
   }
   fprintf(stdout, "#####\r\n%s#####\r\n", TEXTO_BASE);
   fprintf(fpNue, "%s", TEXTO_BASE);
   fclose(fpNue);
   return 1;
}


int leerParametros(FILE *fpEnt, int *corridas, double **paramPR)
{
   char    linea[500],
         *aux;
   int     irepetible,
         ciclo = 0;
   double *valores;

   if(fgets(linea, sizeof(linea), fpEnt) &&
      strncasecmp(linea, "Corridas:", strlen("Corridas:")) == 0 &&
      (aux = strchr(linea, '\t')) != NULL)
```

```
    sscanf(aux, "%d", corridas);
  else
    return 0;
  if(*corridas > 99)
    *corridas = 99;

  if(fgets(linea, sizeof(linea), fpEnt) &&
    strncasecmp(linea, "SRand:", strlen("SRand:")) == 0 &&
    (aux = strchr(linea, '\t')) != NULL)
    sscanf(aux, "%d", &irepetible);
  else
    return 0;
  if(!fgets(linea, sizeof(linea), fpEnt) ||
    strcasecmp(linea, "Param P\tParam R\n"))
    return 0;
  valores = (double *)calloc(*corridas * 2, sizeof(double));
  while(fgets(linea, sizeof(linea), fpEnt)&&
      (aux = strchr(linea, '\t')) != NULL &&
      ciclo < *corridas * 2)
  {
    if(ciclo >= *corridas * 2)
      return 0;
    sscanf(linea, "%lf\t%lf", &valores[ciclo], &valores[ciclo + 1]);
    ciclo += 2;
  }
  *paramPR = valores;
  if(irepetible == 1)
    putenv("SRAND=1");
  return ciclo == *corridas * 2;
}


int eliminarAnterior()
{
  int    ciclo = 0;
  char   comando[500];

  while(ciclo < 100)
  {
    sprintf(comando, "RMDIR /S /Q .\\%02d-Output 2> nul", ciclo);
    system(comando);
    ciclo++;
  }
  while(--ciclo)
  {
    sprintf(comando, "RMDIR /S /Q .\\%02d-Input 2> nul", ciclo);
```

```
      system(comando);
   }
   return 1;
}


int crearDirectorios(int corridas)
{
   char    comando[100];
   int     ciclo;

   for(ciclo = 0; ciclo < corridas; ciclo++)
   {
      sprintf(comando, ".\\%02d-Input", ciclo + 1);
      _mkdir(comando);
      sprintf(comando, ".\\%02d-Output", ciclo);
      _mkdir(comando);
   }

   return 1;
}

int ejecutar(int ciclo, double paramP, double paramR)
{
   char    comando[100],
         *sRand = getenv("SRAND");

   if(sRand)
   {
      sprintf(comando, "SRAND=%d", ciclo);
      putenv(comando);
   }
   /** ParaStockYSumaResiduos.exe */
   sprintf(comando, "PAT_OUT=.\\%02d-Output\\", ciclo);
   putenv(comando);
   sprintf(comando, "PAT_INP_1=.\\%02d-Input\\", ciclo);
   putenv(comando);
   sprintf(comando, "PAT_INP_2=.\\%02d-Input\\", ciclo);
   putenv(comando);
   sprintf(comando, ".\\%s %lf", PROG_10, paramP);
   system(comando);
   /** CostosTranspPaleterasYFabricYRecup.exe */
   sprintf(comando, "PAT_INP_1=.\\00-Input\\");
   putenv(comando);
   sprintf(comando, ".\\%s", PROG_11);
   system(comando);
```

```
   /** GenerarInput.exe */
   sprintf(comando, "PAT_OUT=.\\%02d-Input\\", ciclo + 1);
   putenv(comando);
   sprintf(comando, "PAT_INP_1=.\\%02d-Input\\", ciclo);
   putenv(comando);
   sprintf(comando, "PAT_INP_2=.\\%02d-Input\\", ciclo);
   putenv(comando);
   sprintf(comando, ".\\%s %lf", PROG_12, paramR);
   system(comando);
   return 1;
}


int generarSimulacion(const char *params)
{
   int     retVal = 0,
          corridas,
          ciclo;
   double *paramPR;
   FILE   *fpEnt = fopen(params, "rt");

   if(fpEnt == NULL)
   {
      mostrarModoUso(params);
      goto SALIDA;
   }
   fprintf(stdout, "Leyendo el archivo \"%s\".\n", params);
   if(leerParametros(fpEnt, &corridas, &paramPR) == 0)
   {
      fprintf(stderr, "ERROR - archivo \"%s\" avcio.\n", params);
      return 0;
   }
   fclose(fpEnt);
   fprintf(stdout, "Eliminando la(s) salida(s) anterior(es).\n");
   eliminarAnterior();
   fprintf(stdout, "Creando el/los directorio(s) de salida.\n");
   crearDirectorios(corridas);
   fprintf(stdout, "Ejecutando la simulacion.\n");
   for(ciclo = 0; ciclo < corridas; ciclo++)
   {
      fprintf(stdout, "### Paso nro. %d ###\n", ciclo + 1);
      ejecutar(ciclo, paramPR[ciclo * 2], paramPR[ciclo * 2 + 1]);
      fprintf(stdout, "\n");
   }
SALIDA :
   system("PAUSE");
```

```
   return retVal;
}
```

5ta etapa: proyecto "GenerarParams.cbp"

main.h

```
#ifndef MAIN_H_
#define MAIN_H_

#include <stdio.h>
#include <stdlib.h>

#include "params.h"


#define PARAMETROS      "parametros.csv"

/**
double recuperarParametro(int cArg, char *argumento);
 **/

#endif // MAIN_H_
```

main.c

```
#include "main.h"

int main(int cArg, char **vArg)
{
   int        retVal = 1;

   if(!crearParams(PARAMETROS))
      goto SALIDA;

SALIDA :
   return !retVal;
}
```

params.h

```
#ifndef RESIDUOS_H_
#define RESIDUOS_H_
```

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <direct.h>

#define TEXTO_BASE     "Corridas:\t4\n"                          \
                       "Repetible:\t1\n"                         \
                       "Param P\tParam R\n"                      \
                       "0.8\t0.9\n"                              \
                       "0.8\t0.9\n"                              \
                       "0.8\t0.9\n"                              \
                       "0.8\t0.9\n"

#define PROG_10        "ParaStockYSumaResiduos.exe"
#define PROG_11        "CostosTranspPaleterasYFabricYRecup.exe"
#define PROG_12        "GenerarInput.exe"


int crearParams(const char *params);



#endif // RESIDUOS_H_
```

params.c

```
#include "params.h"


char menu(const char *msj, const char *opciones)
{
   char    opc;

   do
   {
      fprintf(stdout, "%s", msj);
      fflush(stdin);
      fscanf(stdin, "%c", &opc);
   } while(!strchr(opciones, opc));
   return opc;
}


int cargarParams(FILE *fpSal)
```

```
{
   int    corridas,
          ciclo;
   char   sRand;
   double paramP,
          paramR;

   fprintf(stdout, "Corridas: (1 a 98)\t");
   fflush(stdin);
   fscanf(stdin, "%d", &corridas);
   if(corridas < 1)
      corridas = 1;
   if(corridas > 98)
      corridas = 98;
   fprintf(fpSal, "Corridas:\t%d\n", corridas);
   sRand = menu("SRand (0/1):\t", "01");
   fprintf(fpSal, "SRand:\t%c\n" "Param P\tParam R\n", sRand);
   ciclo = 0;
   while(ciclo < corridas)
   {
      fprintf(stdout, "(%2d) Param P:\t", ciclo);
      fflush(stdin);
      fscanf(stdin, "%lf", &paramP);
      fprintf(stdout, "(%2d) Param R:\t", ciclo);
      fflush(stdin);
      fscanf(stdin, "%lf", &paramR);
      ciclo++;
      if(paramP < 0.01)
         paramP = 0.01;
      if(paramP > 1.00)
         paramP = 1.00;
      if(paramR < 0.01)
         paramR = 0.01;
      if(paramR > 1.00)
         paramR = 1.00;
      fprintf(fpSal, "%.2lf\t%.2lf\n", paramP, paramR);
      fprintf(stdout, "\n");
   }
   return ciclo;
}


int mostrarParams(FILE *fpEnt)
{
   char   linea[500];
   int    cant = 0;
```

```
   fprintf(stdout,
         "El archivo de parametros contiene:\n"
         "== ======= == ========== =========\n\n");
   while(fgets(linea, sizeof(linea), fpEnt))
   {
      fprintf(stdout, "%s", linea);
      cant++;
   }
   if(cant == 0)
      fprintf(stdout, "ERROR - archivo vacio.\n");
   fprintf(stdout, "\n==================================\n");
   return cant;
}


int crearParams(const char *params)
{
   int     retVal = 0;
   FILE   *fpParam = fopen(params, "rt");

   if(fpParam)
   {
      retVal = mostrarParams(fpParam);
      fclose(fpParam);
   }
   else
      fprintf(stdout, "El archivo de parametros no existe.\n");
   if(toupper(menu("Desea modificarlo/crearlo (S/N)", "SsNn")) == 'S')
   {
      if((fpParam = fopen(params, "wt")) == NULL)
         fprintf(stderr, "ERROR - creando el archivo \"%s\".\n", params);
      else
      {
         cargarParams(fpParam);
         fclose(fpParam);
         fpParam = fopen(params, "rt");
         if(fpParam)
         {
            retVal = mostrarParams(fpParam);
            fclose(fpParam);
         }
      }
   }
   system("PAUSE");
   return retVal;
```

```
}
```

NOTAS ADICIONALES:

Los fuentes (**lista.c** y **lista.h**) son de bibliotecas preexistentes a las que se les ha agregado alguna primitiva específica reutilizando algunas de las preexistentes. Es por ello su extensión. Corresponden a su uso académico en la materia Programación II de la carrera de Ingeniería en Informática de la UNLaM (materia de la que quien escribe fue Jefe de Cátedra hasta su jubilación). Se eligió la de lista doblemente enlazada que permite una gran flexibilidad con una pequeña sobrecarga de tiempos de ejecución, y permiten trabajar con el tipo de dato abstracto más próximo a este paradigma, con cualquier tipo de dato de la información con la que se esté tratando.

Además, en los programas puede haber algún comentario que haya quedado desactualizado durante el desarrollo y modificación en la etapa de prueba y modificación del código. Se ha tratado de desarrollar este software siguiendo el paradigma Cero Defecto, asegurando al máximo que no haya errores en tiempo de ejecución, y desarrollar tres módulos (programas ejecutables) independientes que permitan la repetibilidad o no de los resultados obtenidos, empleando la variable de entorno **SRAND** (ver uso de funciones de biblioteca **putenv** y **getenv**).

El cuarto módulo es el encargado de ejecutar los tres primeros fijándole su entorno de ejecución a través de las variables de entrono del Sistema Operativo, con el fin de obtener los resultados de las distintas corridas.

Finalmente, se ha utilizado el Lenguaje C, dado que sus tiempos de ejecución son ínfimos comparados con lenguajes que responden al Paradigma de Programación Orientada a Objetos.

- Anexo II: FPI-013: Evaluación de alumnos integrantes. (si corresponde)

Unidad Académica: Departamento de Ingeniería e Investigaciones Tecnológicas
Código:C240
Título del Proyecto: Minería de datos y simulación sobre un sistema de stock y transporte
Director del Proyecto: Cristóbal Raúl Santa María
Programa de acreditación: PROINCE X CyTMA2:…..
Fecha de inicio:01./01./2021
Fecha de finalización: 31/12./2022

1. Datos del alumno

Apellido y Nombre: Bazán Mariana Salomé

DNI: 38126907

Unidad Académica: DIIT

Carrera que cursa: Ingeniería Industrial

Período evaluado: 01/01/2021- 31/12/2022

2. Dictamen de evaluación de desempeño del alumno:
Colocar una cruz donde corresponda

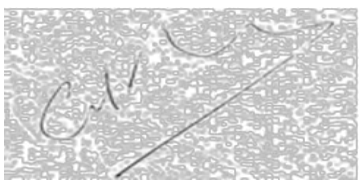
2.1 Satisfactorio: X
2.1 No satisfactorio:

Fundamentos del dictamen: La estudiante colaboró en la formulación del modelo de simulación participando en las reuniones al respecto y también se ocupó de la escritura de informes y la búsqueda de bibliografía con eficiencia. Culminó su carrera en 2022 participando ya recibida en el último tramo del trabajo y trasladándose a Sevilla, España para hacer una maestría en Gestión al haber obtenido una beca universitaria de posgrado allí. Por ese motivo luego siguió colaborando solo de forma virtual hasta terminar el trabajo.

3. Propuesta de continuidad en el proyecto (si corresponde según duración estimada)
Colocar una cruz donde corresponda

3.1 Continuar en el presente proyecto:
3.2 No continuar en el presente proyecto:

Fundamentos del dictamen:
……
……
……



San Justo, 10 de marzo de 2023 Cristóbal R. Santa María

Lugar y fecha Firma del Director Aclaración de firma