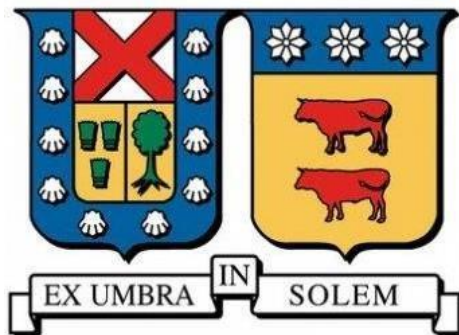




UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE INDUSTRIAS

**DISEÑO DE UN SISTEMA DE APOYO A LA TOMA DE DECISIONES PARA LA
ESTIMACIÓN DE DEMANDA ACADÉMICA Y LA ASIGNACIÓN DE HORARIOS A
ASIGNATURAS MEDIANTE PROGRAMACIÓN LINEAL ENTERA MIXTA EN LA
CARRERA DE INGENIERÍA CIVIL INDUSTRIAL DE LA UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA**

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL INDUSTRIAL

AUTOR

RUBÉN ALEJANDRO RIQUELME MAUREIRA

PROFESOR GUÍA

ESTEBAN SALGADO

SANTIAGO, SEPTIEMBRE, 2026



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Diseño de un sistema de apoyo a la toma de decisiones para la estimación de demanda académica y la asignación de horarios a asignaturas mediante programación lineal entera mixta en la carrera de ingeniería civil industrial de la universidad técnica federico Santa María.

Nombre del candidato(a): Rubén Alejandro Riquelme Maureira

Carrera / Grado: Ingeniería civil industrial

Campus: Vitacura

Departamento: Industrias

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Esteban Salgado Valenzuela, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 03-09-2026

Firma:

Estudiante o Candidato(a):

Fecha: 03-09-2026 **Firma:**

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Índice

1	Resumen Ejecutivo	9
2	Glosario.....	11
3	Introducción	12
4	Objetivos.....	15
4.1	Objetivo General	15
4.2	Objetivos Específicos.....	15
5	Marco Teórico.....	16
5.1	Planificación universitaria	16
5.2	Estimación de la demanda académica.....	19
5.3	Grafos de conflictos	21
5.4	University Timetabling Problem.....	22
5.4.1	Definición y estructura	22
5.4.2	Casos y metodologías empleadas.....	23
5.5	Modelos de Optimización.....	25
5.5.1	Origen, evolución y fundamentos de la optimización.....	25
5.5.2	Modelos de Programación Lineal Entera Mixta.....	27
5.5.3	Estructura base	29
5.6	Indicadores de eficiencia y eficacia	30
6	Metodología	33
6.1	Enfoque y alcance	33
6.2	Recolección de datos.....	33

6.3	Modelos de estimación de demanda	35
6.4	Elaboración de grafos de conflictos	39
6.5	Formulación del MILP	40
6.5.1	Conjuntos	40
6.5.2	Parámetros	41
6.5.3	Variables	42
6.5.4	Expresión auxiliar de ocupación	42
6.5.5	Función objetivo	42
6.5.6	Restricciones	43
6.6	Métricas de desempeño	44
7	Resultados y Discusión	47
7.1	Preparación y caracterización de los datos	47
7.2	Resultados de la estimación de la demanda académica	48
7.2.1	Comportamiento histórico de la demanda	48
7.2.2	Desempeño de los modelos predictivos	50
7.2.3	Selección del modelo	54
7.3	Resultados de los grafos de conflicto	57
7.4	Resultados del modelo MILP	62
7.4.1	Supuestos del modelo	62
7.4.2	Resultados computacionales	63
7.4.3	Asignación de bloques horarios	65
7.5	Evaluación del modelo obtenido	67
8	Conclusiones y Recomendaciones	71
8.1	Conclusiones	71
8.2	Recomendaciones	73
9	Referencias	75

10	Anexos	78
10.1	Anexo A.....	78
10.2	Anexo B	81

Índice de figuras

Figura 1: Diagrama de flujo de la universidad de La Guajira (manual).	17
Figura 2: Diagrama de flujo de la universidad de Pamplona (proceso automatizado).	18
Figura 3: Cálculo de promedio y desviación.	36
Figura 4: Cálculo de regresión lineal.	36
Figura 5: Cálculo de árbol de decisión.	37
Figura 6: Cálculo de Random Forest.	38
Figura 7: Cálculo de Gradient Boosting.	39
Figura 8: Gráfico que muestra los 15 ramos con mayor demanda.	49
Figura 9: Gráfico que muestra un extracto de la demanda de los ramos por semestre.	49
Figura 10: Gráfico que muestra los 15 ramos con mayor desviación estándar.	50
Figura 11: Árbol de decisión.....	52
Figura 12: Indicadores de los modelos.	55
Figura 13: Grafo de conflictos completo para el plan 6006.....	58
Figura 14: Grafo de conflictos segmentado para el plan 6006.	59
Figura 15: Grafo de conflictos completo para el plan 6008.....	60
Figura 16: Grafo de conflictos segmentado para el plan 6008.	61
Figura 17: Horario para el semestre 3 del plan 6006.....	66
Figura 18: Horario para el semestre 8 del plan 6006.....	67
Figura 19: Importación de las bases de datos.	78

Figura 20: Preparación de las bases de datos.....	78
Figura 21: Creación de la base de datos de la demanda con RF.....	79
Figura 22: Creación del grafo de conflictos general del plan 6006.	79
Figura 23: Creación del grafo de conflictos general del plan 6008.	80
Figura 24: Creación del grafo de conflictos segmentado por semestre del plan 6006 y 6008.	80

Índice de tablas

Tabla 1: Conjuntos del modelo.	40
Tabla 2: Parámetros del modelo.	41
Tabla 3: Variables del modelo. Elaboración propia.....	42
Tabla 4: Cantidad de filas y columnas de cada base de datos inicialmente.....	47
Tabla 5: Cantidad de filas y columnas de cada base de datos posterior a la limpieza.....	48
Tabla 6: Cantidad de filas y columnas de cada base de datos elaboradas.	48
Tabla 7: Indicadores para el modelo de regresión lineal.	51
Tabla 8: Indicadores para el modelo de árbol de decisión.....	52
Tabla 9: Indicadores para el modelo de Random Forest.....	53
Tabla 10: Indicadores para el modelo de Gradient Boosting.....	54
Tabla 11: Indicadores para todos los modelos.....	55
Tabla 12: Predicciones obtenidas con RF.....	56
Tabla 13: Resumen del modelo.....	63
Tabla 14: Cruce entre mallas y predicción de RF.....	64
Tabla 15: Ramos que no deben tener conflictos por semestre.....	65
Tabla 16: Validación de conflictos entre asignaturas.	68
Tabla 17: Promedio de bloques libres.....	69
Tabla 18: Porcentaje de bloques utilizados.....	69

1 Resumen Ejecutivo

La planificación de horarios académicos es un proceso importante dentro de la gestión universitaria, ya que debe considerar factores como los planes de estudio, la demanda de estudiantes y las distintas restricciones académicas y operativas. A partir de esta problemática, la presente memoria desarrolla un sistema de apoyo a la toma de decisiones para la carrera de Ingeniería Civil Industrial del Campus Vitacura de la Universidad Técnica Federico Santa María. El sistema integra la estimación de la demanda académica con la asignación de bloques horarios, considerando las asignaturas impartidas por el Departamento de Industrias de los planes de estudio 6006 y 6008.

Para estimar la demanda de estudiantes por asignatura se utilizaron antecedentes históricos proporcionados por la universidad, los cuales fueron previamente limpiados y estructurados. Luego se evaluaron distintos modelos de predicción: promedio simple, regresión lineal, árbol de decisión, Random Forest y Gradient Boosting. Al comparar sus resultados mediante los indicadores MAE, RMSE y R^2 , Random Forest presentó el mejor desempeño, con un MAE de 3,552, un RMSE de 7,434 y un R^2 de 0,957. Por esta razón, este modelo fue utilizado para estimar la demanda académica y posteriormente determinar la cantidad de paralelos necesarios para cada asignatura.

Con la demanda estimada y los conflictos identificados entre asignaturas de un mismo semestre, se formuló un modelo de programación lineal entera mixta (MILP) para realizar la asignación de bloques horarios. El modelo considera distintos criterios, entre ellos la selección de patrones horarios convenientes, la reducción del uso de bloques tardíos, la reducción de coincidencia entre paralelos y la distribución de las clases durante la semana. Para el escenario analizado se consideraron 28 asignaturas y se generaron 82 combinaciones de asignatura-paralelo. Como resultado, el modelo obtuvo una solución factible en un tiempo de cinco minutos.

Para evaluar la calidad del horario generado se utilizaron tres indicadores. En primer lugar, se analizaron 299 pares de asignaturas y no se registraron conflictos entre asignaturas. Además, el promedio de bloques ociosos (ventanas) se mantuvo entre 0 y 2 bloques por jornada, dependiendo del semestre. Por último, la tasa de utilización general de los bloques disponibles alcanzó un 71,79 %. Estos resultados permiten concluir que la herramienta desarrollada logra estimar la demanda, determinar la cantidad de paralelos requeridos y generar horarios factibles, con pocos bloques libres entre asignaturas y una distribución adecuada durante la semana. De esta forma, el sistema puede ser utilizado como una herramienta de apoyo para la jefatura de carrera en el proceso de planificación académica.

2 Glosario

- MILP o PLEM: Modelo de programación lineal entera mixta.
- UTP: University Timetabling Problem.
- UCTP: University Course Timetabling Problem.
- Simulated Annealing: Corresponde a un algoritmo de optimización heurística inspirado en el proceso de enfriamiento de metales.
- PL: Programación lineal.
- Machine learning: Corresponde a una rama de la inteligencia artificial que permite a las máquinas aprender de manera automática en base a datos.
- Redes neuronales: Se trata de un subgrupo de machine learning inspirado en la estructura y funcionamiento del cerebro humano.
- Greedy: Es un algoritmo heurístico que permite resolver problemas de optimización mediante la toma de decisiones paso a paso.

3 Introducción

Dentro de toda universidad existe una problemática asociada a la asignación de horarios a asignaturas y a que esta se realice de manera óptima. Para lograrlo, las diversas casas de estudio deben considerar una gran variedad de factores, tales como la disponibilidad de docentes, la demanda académica, posibles conflictos horarios, la malla curricular de cada carrera y las asignaturas que deben impartirse por semestre, entre otros. Por ello, la asignación de bloques horarios a las diversas asignaturas se posiciona como un tema relevante dentro de la gestión universitaria. Sumado a lo anterior, en los últimos años la demanda por matrículas ha ido en aumento, asociada a diversos factores de carácter político, económico y a las mayores oportunidades de acceso a la educación superior. Un dato relevante es que para el año 2025 las matrículas en educación superior aumentaron en un 5 % respecto al año 2024 (Subsecretaría de Educación Superior, 2025), consolidando así un crecimiento sostenido desde el término de la pandemia. Todo esto se debe considerar al momento de generar una estrategia para la asignación de horarios académicos.

Actualmente, la Universidad Técnica Federico Santa María cuenta con un modelo para la creación de horarios en sus distintos campus. Sin embargo, este se desarrolla de manera centralizada, sin considerar las particularidades de cada sede. Esto quiere decir que desde Casa Central se envía a cada campus el horario en el que se debe impartir cada asignatura y la cantidad de paralelos de cada una, basándose en un informe histórico. Esto genera conflictos, ya que no se consideran las condiciones específicas de cada campus ni las necesidades particulares de los departamentos académicos.

En el caso del campus Vitacura, se identifican diversas problemáticas asociadas a este sistema. En primer lugar, la asignación de horarios basada en datos históricos impide una planificación óptima, privilegiando la repetición de patrones anteriores por sobre una evaluación actual de las necesidades existentes. Esto se evidencia al momento de afrontar la toma de ramos

por parte de los alumnos, ya que existen asignaturas con demanda superior (o inferior) a la esperada. En consecuencia, hay casos en los cuales estudiantes quedan sin poder tomar asignaturas que deben cursar por malla. Sumado a esto, se genera un alto volumen de solicitudes por parte de la comunidad estudiantil para poder cursar dichas asignaturas. Por otro lado, también se generan inconvenientes por parte del cuerpo docente, debido a que se deben adaptar al plan impuesto por Casa Central y no al revés. Finalmente, el modelo presenta un alto nivel de rigidez, dificultando la adaptación ante cambios imprevistos. Esto genera procesos burocráticos y lentos ante solicitudes de modificación.

A partir de lo anterior, se identifica una necesidad concreta en el campus Vitacura, así como una oportunidad de mejora en el proceso de asignación de horarios a las diversas asignaturas. La implementación de una solución permitiría mejorar la calidad del servicio académico, optimizar el uso de la infraestructura disponible y, potencialmente, reducir costos operacionales. En este contexto, surge la oportunidad de desarrollar un modelo de optimización que permita generar una asignación de bloques más eficiente, considerando las restricciones y recursos disponibles, y con potencial de ser aplicado en el entorno real de la universidad, específicamente en el campus ubicado en Vitacura.

El alcance de la presente memoria comprende el desarrollo de un sistema de apoyo a la toma de decisiones orientado a facilitar la planificación académica de la carrera de Ingeniería Civil Industrial del Campus Vitacura de la Universidad Técnica Federico Santa María. La herramienta permitirá estimar la demanda académica de las asignaturas y apoyar al jefe de carrera en la asignación de bloques horarios para los distintos cursos y paralelos correspondientes tanto al plan de estudios antiguo (6006), como al plan de estudios vigente (6008). Para ello, se considerarán las principales restricciones académicas y operativas asociadas al proceso, con el propósito de generar una propuesta de horarios factible y optimizada. No obstante, los resultados

obtenidos constituirán una recomendación para apoyar la planificación, manteniéndose la decisión final bajo responsabilidad de la jefatura de carrera.

En función de lo anterior, surgen las siguientes interrogantes: ¿Qué nivel de precisión puede alcanzar un modelo de estimación de demanda académica para determinar la cantidad de estudiantes esperados en las asignaturas de la carrera de Ingeniería Civil Industrial del Campus Vitacura? ¿En qué medida un modelo de programación lineal entera mixta permite generar una asignación factible y optimizada de bloques horarios, considerando las restricciones académicas y operativas del proceso? y, por último, ¿Qué beneficios cuantitativos y cualitativos puede aportar el sistema de apoyo a la toma de decisiones propuesto al proceso actual de planificación académica?

4 Objetivos

4.1 Objetivo General

Desarrollar un sistema de apoyo a la toma de decisiones que integre la estimación de la demanda académica y la optimización de la asignación de bloques horarios mediante programación lineal entera mixta, para mejorar la planificación de las asignaturas de la carrera de Ingeniería Civil Industrial del Campus Vitacura de la Universidad Técnica Federico Santa María.

4.2 Objetivos Específicos

Levantar, limpiar y estructurar la información necesaria para el desarrollo del sistema, considerando datos de asignaturas, docentes, estudiantes y antecedentes académicos históricos.

Desarrollar y evaluar un modelo de estimación de demanda académica para determinar la cantidad esperada de estudiantes por asignatura en los planes de estudio 6006 y 6008.

Formular e implementar un modelo de optimización basado en programación lineal entera mixta que represente el problema de asignación de horarios bajo las restricciones del sistema.

Evaluar el sistema propuesto mediante el cumplimiento de restricciones y la calidad de los horarios generados.

5 Marco Teórico

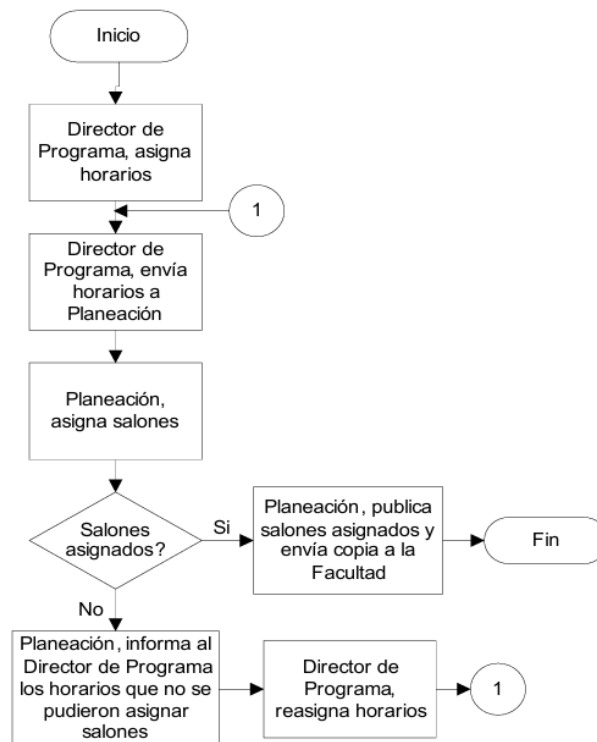
5.1 Planificación universitaria

La planificación académica universitaria implica coordinar diversos factores para organizar adecuadamente las actividades de cada periodo. Según Barrón Tobías (2018), este proceso comprende el análisis del sistema, la definición de objetivos, la identificación de los recursos disponibles, la programación de las actividades y la evaluación de los resultados. En este contexto, la elaboración de horarios no consiste únicamente en asignar bloques a las asignaturas, sino también en compatibilizar la disponibilidad docente, la carga académica, la cantidad de estudiantes, los grupos y la infraestructura existente. Sin embargo, muchas instituciones de educación superior aún realizan esta tarea de forma manual, lo que puede generar incompatibilidades, ineficiencias y una elevada carga de trabajo para los responsables. Además, debido a que las condiciones varían en cada periodo académico, las planificaciones anteriores no siempre responden adecuadamente a los nuevos requerimientos de docentes y estudiantes, lo que puede afectar su nivel de satisfacción.

Esta problemática no se limita a una institución en particular, sino que también se observa en otras universidades, donde la elaboración manual de horarios genera conflictos académicos, reprocesos y dificultades para responder oportunamente a las necesidades de cada periodo. Un ejemplo de ello se presenta en el programa de Ingeniería Industrial de la Facultad de Ingeniería de la Universidad de La Guajira, Colombia. Según Mejía Caballero y Paternina Arboleda (2010), los directores de programa son los encargados de elaborar semestralmente y de manera manual los horarios de clases, labor que puede demandar varios días de trabajo y no garantiza una asignación libre de conflictos. Entre los principales inconvenientes se encuentran la programación de dos asignaturas para un mismo docente en un mismo horario y la superposición de cursos que impide a los estudiantes inscribir todas las asignaturas que requieren, además de las molestias y reprocesos asociados a la corrección de estos cruces.

Asimismo, los responsables no cuentan con información previamente planificada sobre la cantidad de estudiantes por asignatura, lo que dificulta anticipar el número de paralelos necesarios y puede obligar a abrir grupos adicionales, asignar nuevos docentes y reprogramar los horarios. A continuación, se muestra un diagrama correspondiente al proceso que utiliza la Universidad de La Guajira.

Figura 1: Diagrama de flujo de la universidad de La Guajira (manual).

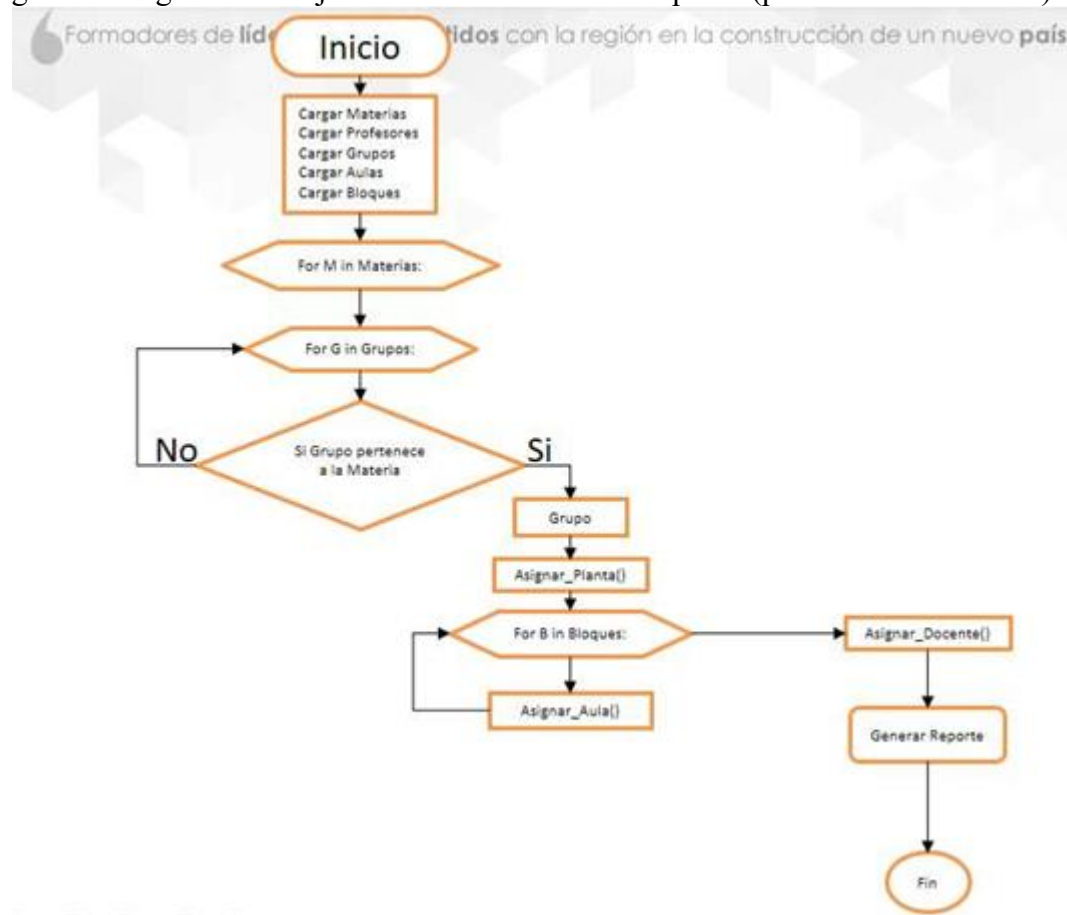


Nota: Figura extraída de Mejía Caballero y Paternina Arboleda (2010).

Frente a este tipo de dificultades, distintas universidades han optado por incorporar herramientas tecnológicas que permitan sistematizar la información académica y automatizar parcialmente la elaboración de horarios. En esta línea, Páez Amaya (2019) aborda el caso de la Universidad de Pamplona (Colombia), donde se desarrolla un algoritmo que organiza los horarios a partir de la información de asignaturas, grupos, docentes, aulas y bloques disponibles. El sistema considera restricciones como la disponibilidad de los profesores y salas, la carga máxima de cada docente y la necesidad de evitar cruces entre asignaturas de un mismo semestre. A partir de estos datos, el algoritmo asigna los horarios y recursos disponibles, priorizando

aquellas opciones que cumplen con las condiciones establecidas. Finalmente, genera un reporte que puede ser descargado en formato CSV con las asignaturas, docentes, aulas, días y bloques asignados, lo que permite revisar el resultado obtenido. Para automatizar este proceso, la herramienta fue desarrollada en Python, mediante el framework Django (interfaz para interacción del usuario) y una base de datos MySQL, permitiendo que la información sea ingresada y consultada a través de una plataforma web. A continuación, se muestra un diagrama correspondiente al proceso de funcionamiento del algoritmo.

Figura 2: Diagrama de flujo de la universidad de Pamplona (proceso automatizado).



Nota: Figura extraída de Páez Amaya (2019).

Cabe mencionar que, a diferencia del enfoque de la presente memoria, en este caso se utiliza un algoritmo basado en reglas y no un modelo de programación lineal entera mixta (MILP). Esto demuestra que una misma problemática puede abordarse mediante diferentes metodologías, dependiendo de las características y necesidades de cada institución.

5.2 Estimación de la demanda académica

Para toda universidad existe la necesidad de poder conocer o por lo menos estimar la demanda futura relacionada a matrículas y cupos en las asignaturas. Conocer esta información permite tomar decisiones informadas sobre la asignación de recursos, dotación necesaria de docentes, infraestructura y respaldo financiero a utilizar (Pauta Riera et al., 2025). Según Ríos-Parra (2024), uno de los principales problemas radica en que los cursos deben publicarse antes de la inscripción de los estudiantes. En este contexto, adquieren relevancia las metodologías de pronóstico, ya que permiten apoyar la toma de decisiones a partir de datos.

En estudios como el de Ayasi et al. (2023), se identifican distintas alternativas para abordar la predicción de inscripción. En primer lugar, se menciona la metodología utilizada por la Arab American University in Palestine (AAUP), basada en un modelo manual de carácter estadístico, donde los departamentos académicos pronostican la inscripción a partir de datos históricos y establecen un margen de $\pm 10\%$ respecto del promedio de inscripción del mismo curso en el mismo semestre. Sin embargo, este enfoque presenta limitaciones, especialmente en cursos con mayor incertidumbre, como asignaturas electivas o de formación general, donde pueden existir variaciones inesperadas en el número de estudiantes. Frente a ello, los autores plantean el uso de técnicas más robustas, como machine learning y redes neuronales, las cuales permiten identificar patrones y relaciones a partir de datos académicos históricos. Dentro de los algoritmos evaluados se incluyen regresión logística (LR), descenso gradiente estocástico (SGD), K vecinos más cercanos (KNN), árbol de decisión (CART), Gradient Boosting (GB), clasificador Bagging (BG), máquina de soporte vectorial (SVM), Random Forest (RF) y red neuronal perceptrón multicapa (MLP), entre otros. Para comprender mejor las diferencias entre estos algoritmos, a continuación, se describen aquellos que presentan mayor utilidad para la estimación de la demanda académica. En particular, se consideran la regresión lineal, el árbol de decisión, Random Forest y Gradient Boosting, debido a que permiten analizar los datos desde

distintos enfoques y comparar su capacidad para predecir la inscripción de estudiantes en una asignatura.

- **Regresión lineal:** Corresponde a un modelo estadístico que busca representar la relación existente entre una variable que se desea predecir y una o más variables explicativas. Este modelo permite identificar cómo los cambios en las variables explicativas se relacionan con variaciones en la demanda estimada.
- **Árbol de decisión:** Es un modelo que organiza los datos mediante una serie de preguntas o condiciones. A partir de estas divisiones, agrupa los casos que presentan características similares y entrega una predicción.
- **Random Forest:** Se basa en un modelo que combina varios árboles de decisión para realizar una predicción. Cada árbol analiza los datos de forma distinta y, al final, se promedian sus resultados. Esto permite obtener estimaciones más estables y reducir los errores que podría presentar un solo árbol.
- **Gradient Boosting:** Se trata de un modelo que construye varios árboles de decisión de manera consecutiva. Cada nuevo árbol intenta corregir los errores cometidos por los anteriores, mejorando progresivamente la predicción de la demanda académica.

Para la presente memoria, se consideran estos modelos, debido a que permiten estimar directamente la demanda académica a partir de variables históricas. La regresión lineal permite identificar tendencias generales y analizar la influencia de cada variable, mientras que los modelos basados en árboles pueden reconocer relaciones no lineales y patrones más complejos. De esta manera, su comparación permite evaluar tanto modelos simples e interpretables como alternativas con una mayor capacidad predictiva. Sumado a lo anterior, la estimación de parámetros como la demanda académica permite transformar información incierta en valores

aproximados y útiles para la planificación. Esto contribuye a una toma de decisiones más eficiente, facilita la definición de requerimientos de capacidad, reduce posibles problemas operacionales y permite alimentar el modelo de optimización para la asignación de bloques horarios.

5.3 Grafos de conflictos

En la planificación académica, los grafos de conflicto permiten representar los problemas que existen entre las asignaturas que deben ser incorporadas en un horario. Según Ogunkan et al. (2024), cada asignatura se representa mediante un vértice, mientras que las aristas conectan aquellas que no pueden desarrollarse en un mismo bloque horario. Estos conflictos pueden generarse cuando dos asignaturas comparten el mismo grupo de alumnos, son dictadas por un mismo docente o necesitan utilizar la misma infraestructura. De esta manera, el grafo permite identificar de forma clara cuáles son las asignaturas que no deben ser programadas simultáneamente. Para su construcción, se utiliza la información académica disponible y se incorpora una conexión entre cada par de cursos que presenta alguna condición de incompatibilidad.

La principal utilidad de esta representación en la elaboración de horarios es que permite organizar las asignaturas considerando los conflictos existentes entre ellas. Para ello, se utilizan colores dentro del grafo, lo cual consiste en asignar un color a cada vértice, evitando que dos vértices conectados tengan el mismo color. En este tipo de problemas, cada color puede representar un bloque horario. Por esta razón, las asignaturas que presentan una incompatibilidad deben ser asignadas a bloques distintos, mientras que aquellas que no tienen conflictos pueden ser realizadas en un mismo horario.

En el estudio desarrollado por Ogunkan et al. (2024), esta metodología permite transformar el grafo de asignaturas en un horario sin superposiciones, agrupando en un mismo

bloque únicamente los cursos que pueden realizarse de manera simultánea. Una vez definidos los bloques horarios, se realiza la asignación de salas considerando la cantidad de estudiantes inscritos y la capacidad disponible de cada espacio.

En el contexto de la presente memoria, los grafos de conflicto resultan especialmente relevantes, ya que permiten representar las incompatibilidades entre asignaturas pertenecientes a un mismo semestre de la malla académica. De esta manera, es posible evitar que dichas asignaturas sean asignadas en un mismo bloque horario, reduciendo las superposiciones que podrían afectar a los estudiantes que deben cursarlas simultáneamente.

5.4 University Timetabling Problem

5.4.1 Definición y estructura

Como se establece en Mallari et al. (2023), los modelos UTP (University Timetabling Problem) corresponden a casos de optimización combinatoria NP-completos. Esto significa que son problemas de alta complejidad computacional, por lo que se utiliza MILP para su resolución. El objetivo de estos modelos es determinar la solución óptima para la asignación de recursos en ámbitos académicos (horarios). Es importante señalar que dentro de la estructura de estos modelos se consideran restricciones duras (obligatorias) y blandas (deseables). Sumado a esto, la función objetivo busca minimizar los conflictos y/o maximizar la calidad del horario.

Dentro de los UTP existen tres grandes subgrupos, los cuales son programación escolar (school timetabling), programación de cursos (course timetabling) y programación de exámenes (examination timetabling). En el contexto universitario, el course timetabling (UCTP) resulta sumamente relevante, ya que implica asignar cursos a horarios, profesores y salas, respetando restricciones como la no superposición de recursos y la capacidad de las salas (Mallari et al., 2023). Sumado a esto, su objetivo principal es obtener horarios de alta calidad, lo cual se relaciona con el cumplimiento de restricciones blandas y la reducción de penalizaciones asociadas a su incumplimiento (Babaei et al., 2015). Además, modelos más recientes incorporan

criterios adicionales como la carga de trabajo de los docentes, el uso eficiente de las salas y la reducción de costos operacionales.

5.4.2 Casos y metodologías empleadas

La literatura identifica distintos escenarios del problema de timetabling universitario, los cuales varían según el nivel de información disponible, la cantidad de recursos, las restricciones institucionales y el tamaño del problema. Dependiendo de estas condiciones, se han utilizado distintos métodos de solución, tales como programación entera, MILP, heurísticas, metaheurísticas o enfoques híbridos. A continuación, se presentan diversos escenarios:

- Información completa del sistema: A partir de lo señalado por Gu et al. (2025), se puede identificar un escenario con toda la información necesaria, en la medida en que la formulación del modelo requiere contar previamente con datos sobre cursos, salas, capacidades, horarios, docentes y restricciones institucionales. A partir de esto, este escenario puede resolverse mediante programación entera (ILP), programación lineal entera mixta (MILP) o ILP/MILP con enfoque heurístico, ya que, al tener la información se plantean las variables de decisión, la función objetivo y las restricciones pertinentes. En cuanto a las herramientas de resolución, se señalan 10 alternativas. Tales como CPLEX, Gurobi, Lingo, OpenSolver, GLPK, AIMMS, GAMS, XPRESS, AMPL y Google OR-Tools CP-SAT. Asimismo, mencionan bibliotecas como PuLP en Python y linprog en MATLAB.
- Restricciones propias del establecimiento: En este escenario se establece que cada institución cuenta con condiciones específicas. Considerando esto, en el texto Chen et al. (2021), se señala que no existe una formulación única aplicable a todos los casos, ya que es necesario adaptarse a las condiciones particulares de

cada universidad. Así, cada establecimiento se enfrenta a un problema de timetabling distinto, en el cual se consideran las políticas institucionales, sistema educativo, entre otros. Si bien se indica que es necesario adaptarse, se establecen alternativas para llegar a la solución. Si se desea abordar de manera exacta, se puede formular un modelo matemático mediante ILP o MILP. Por otro lado, si el problema presenta grandes dimensiones o una alta complejidad, se puede resolver mediante heurística o metaheurística para obtener una solución aproximada o factible.

- Disponibilidad limitada de salas: El tercer escenario corresponde a una problemática originada en capacidad insuficiente de salas. Esto provoca que el lecture timetabling sea aún más complejo. Según lo señalado por Adewumi et al. (2009), se establece que el origen de este caso se debe a un aumento en la cantidad de estudiantes, creación de nuevos cursos, insuficiencia de salas/laboratorios, entre otros. Frente a esto, proponen un modelo metaheurístico basado en algoritmos genéticos, dado que permite obtener un resultado factible ante la ineficacia de métodos exactos. Para ello es necesario contar con cierta información como la capacidad de las distintas salas, número de estudiantes inscritos por curso, asignación de docentes, entre otros.
- Crecimiento de demanda académica: En este escenario el sistema es llevado al límite por consecuencia de un aumento en la demanda por cupos. Esto conlleva a un crecimiento exponencial en el tiempo computacional empleado para obtener un resultado factible, es decir, a mayor demanda mayor será el tiempo necesario. Es por esto por lo que en el texto Bashab et al. (2023), se señala que la mejor alternativa para encontrar una solución es abordar la problemática mediante

heurística, metaheurística o algoritmos de aproximación. Si bien no entregan necesariamente una respuesta óptima, es suficientemente buena y se obtiene en un tiempo razonable.

- Enfoque modular: En este último escenario se trabaja con la alternativa de reducir el problema a módulos menores. Es decir, en vez de asignar salas y horarios a toda la universidad, se enfoca en departamentos, campus o sedes. Según Adewumi et al. (2009), se pueden resolver estos problemas al plantear un modelo que se construye desde una estructura menor a una mayor (de abajo hacia arriba). Esto permite reducir la complejidad del problema y adaptarlo a diversos niveles de la institución.

5.5 Modelos de Optimización

5.5.1 Origen, evolución y fundamentos de la optimización

Los modelos de optimización son la respuesta a una vasta búsqueda por resolver problemas y así tomar decisiones ya sean con enfoques tácticos, estratégicos, entre otros. Sus orígenes se remontan a siglos anteriores con grandes pensadores griegos, pero los modelos de optimización actuales surgen durante el siglo XX. Destacando como uno de sus principales precursores a George Dantzig, quien durante el año 1947 planteó la metodología Simplex. Esta corresponde a un procedimiento iterativo de naturaleza algebraica que permite resolver problemas de programación lineal, sustentado en fundamentos geométricos (Vitoriano & Ramos, 2023). Con el pasar de los años, los procesos de optimización fueron evolucionando a la par de la informática, la cual permite plantear y resolver problemas de gran complejidad en el menor tiempo posible. Esto se debe a que cada vez, los computadores cuentan con mejores componentes para el análisis y desarrollo de dichas iteraciones. Un ejemplo corresponde a la creación de CPLEX 1.0 en 1988. Este correspondía a un motor de optimización que ocupaba

código de programación C para resolver un problema con metodología Simplex. Si se realiza una comparación se evidencia la evolución en la velocidad para obtener respuestas de estos problemas. Al comparar el modelo CPLEX 3.0 de 1997 con el CPLEX 7.0 de 2000, se observa una disminución de 28 veces en el tiempo de respuesta. Sumado a esto, si se contrasta la versión 1.0 de 1988 con la versión de los años 2000 se logra apreciar de manera abismal la velocidad del motor, ya que es 10.000 veces más rápido para llegar al resultado (Ramos et al., 2010).

En los problemas de optimización se busca encontrar el valor que deben tomar las variables para satisfacer la función objetivo y las respectivas restricciones (Ramos et al., 2010). Para profundizar en la estructura que caracteriza a dichos problemas, se establece lo siguiente:

- **Función objetivo:** Corresponde a la representación matemática de lo que se desea minimizar o maximizar según sea el caso. Ejemplos típicos son minimizar costos operacionales, encontrar el trayecto más corto entre dos puntos, entre otros.
- **Variables:** Establecen las decisiones que se pueden tomar para lograr la finalidad de la función objetivo.
- **Restricciones:** Representan los márgenes a los cuales está sujeto el problema, es decir, establece los límites que se debe cumplir para el correcto desempeño tanto de las variables como de la función objetivo.

Dentro de lo mencionado, es relevante señalar que los problemas de optimización se clasifican en 2 grupos dependiendo de la metodología empleada para su resolución.

- **Método clásico:** Son técnicas basadas en el cálculo diferencial, con la finalidad de encontrar máximos o mínimos para funciones exactas y derivables.
- **Método metaheurístico:** Técnicas empleadas para llegar a soluciones suficientemente buenas para problemas computacionalmente complejos. Por lo general se inspiran en fenómenos sencillos observados en la naturaleza.

Complementando la información anterior, la ingeniería juega un rol fundamental dentro de los problemas de optimización. Esto se debe principalmente a que funciona como un puente que permite conectar la teoría matemática con casos reales.

5.5.2 Modelos de Programación Lineal Entera Mixta

Para comprender este estilo de modelos, primero es importante establecer el caso más básico. Este corresponde a la programación lineal, la cual consiste en una técnica empleada para resolver casos de optimización en los que se requiere asignar recursos limitados entre diversas actividades competitivas. Su propósito es determinar el nivel de actividades que maximiza los beneficios o minimiza los costos, considerando restricciones establecidas (Guédez Fernández, 2011). Sumado a esto, se caracteriza principalmente por ser empleada en problemas lineales (las restricciones son proporciones directas) y encontrar la solución dentro de un “área factible”.

La estructura que compone a la PL es la misma que la señalada en el capítulo anterior para los problemas de optimización, es decir, cuenta con una función objetivo compuesta por variables y se está sujeto a restricciones. Dentro de este punto, resalta una metodología para su resolución:

- **Método Simplex:** Corresponde a un algoritmo iterativo que recorre los vértices de un polígono (denominado como conjunto factible) de extremo a extremo, asegurando que el valor de la función objetivo aumente o por lo menos, que se mantenga. Este proceso se repite hasta encontrar un vértice donde ninguna de las aristas lleve a una mejor solución. Es importante señalar que el polígono tiene aristas definidas, por lo que el proceso tiene una duración finita (Nemirovski, 2023).

Cabe mencionar que, si bien se logra llegar a soluciones óptimas en un gran número de casos, los problemas de programación lineal cuentan con ciertas limitaciones. En primer lugar,

una de las principales limitaciones de este estilo de problemas es que los casos en la vida real no siempre se pueden representar de manera lineal. En segundo lugar, se considera un modelo determinista, es decir, que todos los parámetros son conocidos. Esto no deja espacio para la incertidumbre. En último lugar, este estilo de casos presenta complicaciones para enfrentar problemas de gran envergadura, ya que conlleva un gran costo computacional (Guédez Fernández, 2011).

Teniendo dichas limitaciones en cuenta es que se da origen a la programación lineal entera mixta (MILP o PLEM). Según Bermúdez (2011), los modelos MILP se entienden como problemas de optimización con la misma estructura de los lineales, con la diferencia en el dominio de las variables. Esto lo indica el mismo nombre, ya que cuentan con variables que necesitan una respuesta afirmativa o negativa, lo que representa la parte entera. Sumado a esto, si se establece que algunas variables son enteras y otras continuas, se denomina un problema de programación lineal entera mixta. Aquí radica la principal diferencia con el problema anterior, ya que se introducen en el modelamiento variables de toma de decisión binarias.

En este sentido, la incorporación de este tipo de modelos permite abordar problemáticas reales en las cuales existen decisiones discretas y continuas de manera simultánea, como ocurre en sistemas complejos donde se busca coordinar distintos recursos. Un ejemplo de ello se observa en sistemas energéticos modernos, donde la integración de vehículos eléctricos y generación fotovoltaica genera desbalances entre la oferta y la demanda eléctrica, debido a que la producción y el consumo no se concentran en los mismos horarios (Fachrizal et al., 2020). Frente a este escenario, el uso de modelos de optimización, como la programación lineal entera mixta, permite coordinar estas decisiones con el objetivo de mejorar la eficiencia del sistema, reducir costos y optimizar el uso de los recursos disponibles.

Dentro de los modelos de programación lineal entera mixta, es posible identificar

subgrupos de problemas que presentan estructuras específicas, siendo uno de los más relevantes el problema de asignación. Según lo planteado en el texto, este tipo de problema se enfoca en la asignación óptima de recursos a distintas actividades o tareas, considerando un conjunto de restricciones que limitan dicha asignación (Bermúdez, 2011). Para ello, se utilizan variables binarias que representan la decisión de asignar o no un recurso a una tarea determinada, lo que permite modelar de manera precisa este tipo de situaciones. En este contexto, los problemas de asignación pueden ser formulados como modelos de programación lineal entera mixta, ya que combinan la estructura lineal del problema con variables de decisión discretas, permitiendo representar de forma adecuada este tipo de decisiones. Un ejemplo de ello es el problema de asignación de salas, en el cual se intenta ubicar un conjunto de individuos en espacios físicos considerando criterios de compatibilidad. Este tipo de problemas presenta una naturaleza combinatoria, lo que implica una gran cantidad de posibles configuraciones y, por tanto, una alta complejidad en su resolución. En este estilo de casos, se han utilizado diversas técnicas de optimización, como métodos basados en Simulated Annealing (algoritmo dentro de la categoría Monte Carlo), para encontrar soluciones eficientes dentro de un espacio de búsqueda amplio (Dwivedi, 2011).

5.5.3 Estructura base

Un modelo de Programación Lineal Entera Mixta se compone de una función objetivo, un conjunto de restricciones y variables de decisión que pueden ser continuas, enteras o binarias. La función objetivo representa el criterio que se desea maximizar o minimizar, mientras que las restricciones establecen las condiciones que debe cumplir la solución. En problemas de asignación de horarios, las variables binarias suelen indicar si una asignatura se dicta o no en un determinado bloque horario (Lai et al., 2026).

De manera general, un modelo MILP puede expresarse de la siguiente forma:

$$\text{mín o máx } Z = \sum_{j=1}^n c_j x_j + \sum_{k=1}^m d_k y_k \quad (1)$$

Sujeto a:

$$\sum_{j=1}^n a_{ij} x_j + \sum_{k=1}^m e_{ik} y_k \begin{cases} \leq \\ = \\ \geq \end{cases} b_i \quad \forall i = 1, \dots, f \quad (2)$$

$$x_j \in \mathbb{R}_{\geq 0} \quad \forall j = 1, \dots, n \quad (3)$$

$$y_k \in \mathbb{Z}_{\geq 0} \text{ o } y_k \in \{0,1\} \quad \forall k = 1, \dots, m \quad (4)$$

Donde:

- Z : Valor de la función objetivo a optimizar.
- x_j : Variables de decisión continuas.
- y_k : Variables de decisión enteras o binarias.
- c_j, d_k : Coeficientes de costo, beneficio o rendimiento asociados a cada variable.
- a_{ij}, e_{ik} : Coeficientes asociados a las variables en cada restricción.
- b_i : Límite o valor requerido por la restricción i .

Siendo n el número de variables continuas, m el número de variables enteras o binarias y f el número de restricciones del modelo.

5.6 Indicadores de eficiencia y eficacia

Autores como Ahmed (2023) señalan que dentro del enfoque tradicional del desarrollo de un proyecto es relevante evaluar el alcance, la calidad y el costo que conlleva. Con esto se permite cuantificar el modelo y verificar si se está cumpliendo su propósito. Aquí es donde aparecen dos conceptos fundamentales, los cuales son eficacia y eficiencia. Según el texto, para medir estas acciones se utilizan métricas de desempeño, diferentes en cada proyecto. Por definición, una métrica corresponde a un método de cálculo (cuantitativo) que permite evaluar, comparar y realizar el seguimiento del avance de un proyecto. Se señala que estos indicadores permiten a los directores de proyectos tomar acciones correctivas y así lograr evitar posibles

desvíos del camino hacia el objetivo.

En los problemas de asignación de horarios, no basta con comprobar que todas las asignaturas cuenten con un bloque horario, sino que también es necesario evaluar la calidad de la solución obtenida. Para ello, se pueden utilizar métricas que permitan medir el cumplimiento de las restricciones y comparar distintas alternativas de horario de manera objetiva. Lai et al. (2026), en un estudio aplicado a la programación de exámenes universitarios, evalúan la calidad de los horarios considerando diferentes criterios, entre ellos la cantidad de conflictos, la distribución temporal de las actividades y la utilización de los recursos con los que se dispone. Aunque dicho estudio se enfoca en exámenes, su planteamiento permite observar la importancia de establecer indicadores que representen cuantitativamente si la solución obtenida evidencia un buen desempeño o no.

A partir de lo anterior, en problemas de asignación de horario se puede evaluar la calidad de las soluciones obtenidas mediante métricas asociadas a la calidad del horario establecido. Para problemas de University Timetabling Problem (UTP), los indicadores de desempeño no son necesariamente fijos, ya que pueden variar en cada escenario en base al contexto de cada universidad, restricciones, entre otros. No obstante, es posible formular algunos indicadores aplicables en un gran número de casos, tales como:

- N° de superposiciones: Corresponde a la cantidad de pares de asignaturas incompatibles que son programados en un mismo bloque horario. El objetivo es minimizar el número de conflictos existentes, procurando que el valor de este indicador sea igual a cero.
- Promedio de bloques libres: Este indicador corresponde al promedio de bloques libres existentes entre clases de un mismo semestre, calculado como el total de bloques ociosos dividido por la cantidad de jornadas con clases. Su finalidad es

reducir las ventanas horarias y favorecer la elaboración de un horario más compacto para los estudiantes.

Cabe mencionar que no basta con obtener una solución factible, sino que en conjunto con las métricas de desempeño se logra validar si la propuesta realmente mejora la eficiencia y/o eficacia en la asignación de bloques horarios.

6 Metodología

En este capítulo se presenta la metodología y el enfoque utilizados para el desarrollo de la memoria. Se describe el proceso de obtención, preparación y estructuración de los datos, junto con las etapas consideradas para la estimación de la demanda académica y la posterior asignación de bloques horarios a las asignaturas de la carrera de Ingeniería Civil Industrial del Campus Vitacura de la Universidad Técnica Federico Santa María.

6.1 Enfoque y alcance

Como se señaló anteriormente, el estudio tiene como objetivo principal desarrollar una herramienta de apoyo a la toma de decisiones que facilite al jefe de carrera la planificación y organización de los bloques horarios de las asignaturas de Ingeniería Civil Industrial. Para ello, se busca generar información que permita anticipar la demanda académica y proponer una asignación de horarios acorde con las condiciones específicas del Campus Vitacura de la Universidad Técnica Federico Santa María.

El alcance del estudio se limita a las asignaturas impartidas por el Departamento de Industrias, debido a que son aquellas para las cuales se dispone de información. Por lo tanto, los resultados obtenidos deben entenderse como una propuesta de apoyo para este conjunto de asignaturas y no como una planificación completa de toda la oferta académica del campus.

6.2 Recolección de datos

Para llevar a cabo el estudio, se realizó una solicitud formal de información a la Universidad Técnica Federico Santa María, dirigida específicamente al jefe de carrera de Ingeniería Civil Industrial. Esta solicitud tuvo como finalidad obtener antecedentes históricos sobre las asignaturas impartidas por el Departamento de Industrias y conocer el procedimiento utilizado actualmente para la asignación de horarios. A continuación, se presentan las tres principales fuentes de información proporcionadas.

- Cantidad de estudiantes por asignatura: Corresponde a la principal base de datos utilizada en el estudio, ya que contiene información histórica sobre los cupos de las distintas asignaturas impartidas por el Departamento de Industrias. El archivo contiene información relacionada con el año, semestre, código y sigla de la asignatura, nombre, nivel y tipo de asignatura, código y nombre del departamento, campus o sede, jornada, modalidad, número de paralelos, promedio de cupos, cantidad de alumnos desinscritos, con retiro, inscritos, aprobados y reprobados, además del promedio simple obtenido.
- Estudiantes pendientes: Corresponde al listado de estudiantes que presentan asignaturas pendientes hasta el segundo semestre de 2025. La base de datos es de carácter anónimo, ya que a cada estudiante se le asigna un código identificador con el fin de resguardar su privacidad. El archivo contiene información relacionada con el código identificador del estudiante, carrera, campus, jornada, calidad académica, año de ingreso, nivel, sigla de la asignatura, plan de estudios y una sigla alternativa asociada a la asignatura.
- Inscripción de asignaturas: Corresponde a datos relacionados al año 2025 con respecto a los inscritos por asignatura y el desglose correspondiente en la cantidad de estudiantes con cada tipo de VTR. El archivo contiene información relacionada con la sede, periodo académico, sigla y nombre de la asignatura, número de paralelo, cupo disponible, nivel y tipo de asignatura, cantidad total de estudiantes inscritos y número de estudiantes clasificados en las categorías VTR1, VTR2 y VTR3.

Cabe señalar que, al considerar todas las asignaturas impartidas por el Departamento de Industrias, la base de datos incluye información correspondiente a programas de magíster,

jornada vespertina y otras modalidades. Por esta razón, para efectos del estudio se realiza una limpieza y filtrado de los datos mediante Python, considerando únicamente las asignaturas de pregrado impartidas en jornada diurna.

Por otro lado, las fuentes de información no se limitan a los antecedentes proporcionados por el jefe de carrera, ya que también se elaboraron archivos necesarios para el desarrollo de la memoria a partir de la plataforma SIGA. Mediante esta herramienta fue posible generar dos archivos de especial relevancia para la ejecución del estudio, los cuales se detallan a continuación.

- **Plan de estudios malla antigua:** Como su nombre lo indica, corresponde a la base de datos asociada al plan de estudios identificado con el código 6006. En ella se detallan las asignaturas que lo componen, la sigla correspondiente a cada una, al semestre al que pertenece y la cantidad de bloques horarios requeridos.
- **Plan de estudios malla nueva:** De igual manera, este archivo corresponde a la base de datos asociada al nuevo plan de estudios implementado por la carrera, identificado con el código 6008. Contiene información sobre las asignaturas que lo componen, la sigla correspondiente a cada una, el semestre al que pertenecen y la cantidad de bloques requeridos.

6.3 Modelos de estimación de demanda

La estimación de la demanda constituye uno de los aspectos relevantes del estudio. Para ello, se analiza la información disponible y se definen cinco alternativas metodológicas, con el propósito de determinar cuál se ajusta mejor a las características de los datos. A continuación, se presentan los métodos empleados y sus respectivos códigos desarrollados en Python.

- **Promedio simple y desviación estándar:** Se calcula el promedio de la demanda histórica de cada asignatura, diferenciando entre el primer y el segundo semestre

académico. Además, se determina la desviación estándar para analizar la variabilidad de los datos respecto de su promedio y evaluar el nivel de estabilidad de la demanda en cada asignatura.

Figura 3: Cálculo de promedio y desviación.

```
Promedio

Caso_prom = df1.groupby(["sigla", "semestre"]).agg(
    promedio_demanda=("alumnos_inscritos", "mean"),
    desviacion=("alumnos_inscritos", "std")).reset_index()
Caso_prom["desviacion"] = Caso_prom["desviacion"].fillna(0)
print('Filas, Columnas =', df2.shape)
Caso_prom.head()
```

Nota: Elaboración propia.

- **Regresión lineal:** Modelo base utilizado para analizar el comportamiento lineal de la demanda e identificar la relación existente entre las variables explicativas y la cantidad de estudiantes inscritos en cada asignatura.

Figura 4: Cálculo de regresión lineal.

```
Regresion Lineal

# Variables target y predictoras
y = df1["alumnos_inscritos"]
X = df1[["año", "semestre", "nro_paralelos", "promedio_cupos", "alumnos_reprobados", "alumnos_aprobados", "pendientes_2025"]]

# Train/Test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

# Modelo
modelo = LinearRegression()
modelo.fit(X_train, y_train)
y_pred = modelo.predict(X_test)

# Indicadores de eficiencia
mae_rl = mean_absolute_error(y_test, y_pred)
rmse_rl = mean_squared_error(y_test, y_pred)**0.5
r2_rl = r2_score(y_test, y_pred)

print("MAE:", mae_rl)
print("RMSE:", rmse_rl)
print("R2:", r2_rl)

# Coeficientes del modelo
coef = pd.DataFrame({
    "Variable": X.columns,
    "Coeficiente": modelo.coef_
})

print("\n Coeficientes del Modelo Lineal")
print(coef)
```

Nota: Elaboración propia.

- **Árbol de decisión:** Este modelo aporta un enfoque distinto a los métodos anteriores, ya que estima la demanda mediante una estructura jerárquica de reglas

de decisión. Para este escenario, se limita la profundidad del árbol a un máximo de cinco niveles, con el propósito de controlar su complejidad, reducir el riesgo de sobreajuste y evaluar la calidad predictiva de los resultados obtenidos.

Figura 5: Cálculo de árbol de decisión.

```
Arbol de decisión

# Variables predictoras y target
x = df1[["año", "semestre", "nro_paralelos", "promedio_cupos",
        "alumnos_reprobados", "alumnos_aprobados", "pendientes_2025"]]
y = df1["alumnos_inscritos"]

# Escalado
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train/Test split
seed = 125
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=seed)

# Entrenar árbol de decisión
tree_model = DecisionTreeRegressor(max_depth=5, random_state=seed)
tree_model.fit(X_train, y_train)

# Predicciones y métricas
y_pred = tree_model.predict(X_test)
mae_ad = mean_absolute_error(y_test, y_pred)
rmse_ad = mean_squared_error(y_test, y_pred)**0.5
r2_ad = r2_score(y_test, y_pred)

print("MAE:", mae_ad)
print("RMSE:", rmse_ad)
print("R2:", r2_ad)

# Reglas del árbol en texto
rules = export_text(tree_model, feature_names=list(X.columns))
print("\ Reglas del Árbol de Decisión\n")
print(rules)

# Graficar árbol
plt.figure(figsize=(25,12))
plot_tree(tree_model, feature_names=X.columns, filled=True, rounded=True, fontsize=10, proportion=True, precision=2, impurity=False)
plt.title("Árbol de Decisión - Predicción de Demanda Académica", fontsize=16)
plt.show()
```

Nota: Elaboración propia.

- **Random Forest:** Corresponde a un modelo que combina múltiples árboles de decisión para estimar la demanda académica, integrando sus resultados con el fin de obtener una predicción más estable y reducir el riesgo de sobreajuste. Para su aplicación, se utilizan 200 árboles, cada uno con una profundidad máxima de ocho niveles.

Figura 6: Cálculo de Random Forest.

```
Random Forest

# Variables predictoras y target
X = df1[["año", "semestre", "nro_parelos", "promedio_cupos", "alumnos_reprobados", "alumnos_aprobados", "pendientes_2025"]]
y = df1["alumnos_inscritos"]

# Escalado
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train/Test split
seed = 125
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=seed)

# Crear y entrenar (200 árboles y 8 niveles)
rf_model = RandomForestRegressor(n_estimators=200, max_depth=8, random_state=seed, n_jobs=-1)
rf_model.fit(X_train, y_train)

# Predicciones
y_pred = rf_model.predict(X_test)

# Evaluación
mae_rf = mean_absolute_error(y_test, y_pred)
rmse_rf = mean_squared_error(y_test, y_pred)**0.5
r2_rf = r2_score(y_test, y_pred)

print("MAE:", mae_rf)
print("RMSE:", rmse_rf)
print("R2:", r2_rf)

# Importancia de variables
feature_importances = pd.DataFrame({"Variable": X.columns,
                                     "Importancia": rf_model.feature_importances_
                                   }).sort_values(by="Importancia", ascending=False)

print("\ Importancia de Variables\n")
print(feature_importances)

# visualización
plt.figure(figsize=(10,6))
sns.barplot(x="Importancia", y="Variable", data=feature_importances, palette="viridis")
plt.title("Importancia de Variables - Random Forest", fontsize=14)
plt.show()
```

Nota: Elaboración propia.

- **Gradient Boosting:** A diferencia de Random Forest, este modelo construye los árboles de manera secuencial, de modo que cada nuevo árbol busca corregir los errores cometidos por los anteriores. Para su aplicación, se utilizan 200 árboles con una profundidad máxima de cinco niveles, con el propósito de estimar la demanda académica y posteriormente evaluar su desempeño predictivo.

Figura 7: Cálculo de Gradient Boosting.

```

Gradient Boosting

# Variables predictoras y target
X = df1[["año", "semestre", "nro_paralelos", "promedio_cupos", "alumnos_reprobados", "alumnos_aprobados", "pendientes_2025"]]
y = df1["alumnos_inscritos"]

# Escalado
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Train/Test split
seed = 42
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=seed
)

# Crear y entrenar (200 árboles y 5 niveles)
gb_model = GradientBoostingRegressor(
    n_estimators=200, learning_rate=0.1, max_depth=5, random_state=seed
)
gb_model.fit(X_train, y_train)

# Predicciones
y_pred = gb_model.predict(X_test)

# Evaluación
mae_gb = mean_absolute_error(y_test, y_pred)
rmse_gb = mean_squared_error(y_test, y_pred)**0.5
r2_gb = r2_score(y_test, y_pred)

print("MAE:", mae_gb)
print("RMSE:", rmse_gb)

print("R2:", r2_gb)

# Importancia de variables
feature_importances = pd.DataFrame({
    "Variable": X.columns,
    "Importancia": gb_model.feature_importances_
}).sort_values(by="Importancia", ascending=False)

print("\n--- Importancia de Variables - Gradient Boosting ---\n")
print(feature_importances)

# Visualización
plt.figure(figsize=(10,6))
plt.barh(feature_importances['Variable'], feature_importances['Importancia'], color='skyblue')
plt.xlabel("Importancia")
plt.title("Importancia de Variables - Gradient Boosting Regressor")
plt.gca().invert_yaxis()
plt.show()

```

Nota: Elaboración propia.

Por último, se comparan los resultados obtenidos por cada modelo mediante las métricas MAE, RMSE y R^2 , con el propósito de evaluar su desempeño y seleccionar la alternativa que presente la mejor capacidad predictiva.

6.4 Elaboración de grafos de conflictos

Este capítulo resulta relevante, ya que permite establecer una de las bases para la formulación del modelo de programación lineal entera mixta destinado a la asignación de bloques horarios a las asignaturas impartidas por el Departamento de Industrias. Para ello, se considera que las asignaturas pertenecientes a un mismo semestre no pueden ser programadas en un mismo bloque horario, debido a los conflictos que esta superposición podría generar para los

estudiantes.

Con el propósito de representar visualmente estas incompatibilidades, se desarrolla mediante Python una herramienta para la construcción de grafos de conflicto. Estos se elaboran de manera general para cada plan de estudios y, posteriormente, se desagregan por semestre, facilitando la identificación y el análisis de las asignaturas que no pueden coincidir en un mismo bloque horario. El código utilizado se presenta en los anexos.

6.5 Formulación del MILP

Tal como se ha señalado a lo largo del estudio, para desarrollar la asignación de horarios académicos se formula un modelo de programación lineal entera mixta. Este utiliza como principales fuentes de información las bases de datos descritas en el apartado 6.2, la estimación de la demanda académica obtenida mediante el modelo que presenta el mejor desempeño predictivo y las incompatibilidades identificadas a través del grafo de conflictos. A continuación, se presenta la formulación matemática del modelo, incluyendo la definición de sus conjuntos, parámetros, variables de decisión, función objetivo y restricciones.

6.5.1 Conjuntos

Tabla 1: Conjuntos del modelo.

Símbolo	Descripción
M	Conjunto de mallas. $m = \{6006, 6008\}$
S	Conjunto de semestres de la malla m
$R_{m,s}$	Conjunto de ramos del semestre s de la malla m con $r \in R$
R^{malla}	Conjunto de ramos presentes en la malla
R^{RF}	Conjunto de ramos con predicción de demanda
R	Conjunto de ramos. $R = R^{malla} \cap R^{RF}$
D	Conjunto de días
B	Conjunto de bloques horarios
P_r	Conjunto de patrones horarios factibles del ramo r
K_r	Conjunto de paralelos del ramo r

Nota: Elaboración propia.

6.5.2 Parámetros

Tabla 2: Parámetros del modelo.

Símbolo	Descripción
H_r	Carga horaria semanal del ramo r
Dem_r	Demanda estimada del ramo r
Cap^{min}	Cupo mínimo por paralelo
Cap^{max}	Cupo máximo por paralelo
N_r	Número de paralelos requeridos para el ramo r
$Cap_{r,k}$	Capacidad del paralelo k del ramo r
$BP_{d,b}$	Bloque protegido. Binaria
$U_{p,d,b}$	Uso del bloque en el patrón. Binaria
Pen_p	Penalización del patrón p
$Tarde_b$	Bloque tardío. Binaria
α	Ponderador de patrones horarios
β	Ponderador de bloques tardíos
γ	Ponderador de coincidencia entre paralelos
δ	Ponderador de balance global

Nota: Elaboración propia.

Es importante señalar cómo se obtiene el valor de N_r , correspondiente al número de paralelos, ya que inicialmente se dispone de una estimación de la demanda académica. Para determinar este valor, se utiliza la siguiente fórmula:

$$N_r = \max \left\{ 1, \left\lceil \frac{Dem_r}{Cap^{max}} \right\rceil \right\} \quad \forall r \in R$$

Posteriormente, es necesario definir la capacidad de cada paralelo, la cual se determina en función de la demanda estimada, de la capacidad mínima y máxima establecida.

$$Cap_{r,k} = \max \left\{ Cap^{min}, \min \left\{ Cap^{max}, \left\lceil \frac{Dem_r}{N_r} \right\rceil \right\} \right\} \quad \forall r \in R$$

$$Cap^{min} = 25$$

$$Cap^{max} = 45$$

Por último, se plantean dos restricciones que deben cumplirse para garantizar el correcto funcionamiento del modelo.

$$Cap^{min} \leq Cap_{r,k} \leq Cap^{max} \quad \forall r \in R, \forall k \in K_r$$

$$\sum_{k \in K_r} Cap_{r,k} \geq Dem_r \quad \forall r \in R$$

6.5.3 Variables

Tabla 3: Variables del modelo. Elaboración propia.

Símbolo	Descripción
$\mathbf{X}_{r,k,p}$	Asignación de patrón horario. Binaria
$\mathbf{ZP}_{r,k,l,d,b}$	Coincidencia entre paralelos del mismo ramo. Binaria
$\mathbf{L}$	Carga máxima global

Nota: Elaboración propia.

6.5.4 Expresión auxiliar de ocupación

Para simplificar la formulación de la función objetivo y de las restricciones, se define la siguiente expresión auxiliar:

$$Y_{r,k,d,b} = \sum_{p \in P_r} U_{p,d,b} X_{r,k,p} \quad \forall r \in R, \forall k \in K_r, \forall d \in D, \forall b \in B$$

$$Y_{r,k,d,b} \in \{0,1\} \quad \forall r \in R, \forall k \in K_r, \forall d \in D, \forall b \in B$$

6.5.5 Función objetivo

$$\min Z = a \sum_{r \in R} \sum_{k \in K_r} \sum_{p \in P_r} Pen_p X_{r,k,p} + \beta \sum_{r \in R} \sum_{k \in K_r} \sum_{d \in D} \sum_{b \in B} Cap_{r,k} Tarde_b Y_{r,k,d,b} + \gamma \sum_{r \in R} \sum_{\substack{k,l \in K_r \\ k < l}} \sum_{d \in D} \sum_{b \in B} ZP_{r,k,l,d,b} + \delta L$$

La función objetivo es una de las partes más importantes del modelo, ya que permite representar aquello que se busca optimizar. La ecuación anterior está compuesta por cuatro términos, cada uno de los cuales cumple una función relevante. A continuación, se presenta una breve explicación de cada uno de ellos.

- Primer término: Penaliza la selección de patrones con horarios menos convenientes. El ponderador α determina la importancia de este criterio.
- Segundo término: Penaliza la asignación de estudiantes a bloques tardíos. El ponderador β regula cuánto se prioriza evitar estos horarios.
- Tercer término: Penaliza la coincidencia horaria entre paralelos de una misma

asignatura. El ponderador γ define la relevancia de evitar estas coincidencias.

- Cuarto término: Reduce la máxima concentración de clases en un mismo día y bloque. El ponderador δ determina la importancia de distribuir las clases de manera equilibrada.

6.5.6 Restricciones

Complementando lo anterior, el modelo debe estar sujeto a restricciones específicas que aseguren su correcto funcionamiento y orienten la solución de acuerdo con la función objetivo. A continuación, se presentan las restricciones consideradas para la asignación de horarios académicos.

1. Se debe asignar solo 1 patrón a cada paralelo.

$$\sum_{p \in P_r} X_{r,k,p} = 1 \quad \forall r \in R, \forall k \in K_r$$

2. Dos asignaturas del mismo semestre y malla no pueden coincidir en el mismo bloque el mismo día.

$$Y_{r,k,d,b} + Y_{q,l,d,b} \leq 1$$

$$\forall m \in M, \forall s \in S, \forall r, q \in R_{m,s}, r < q, \forall k \in K_r, \forall l \in K_q, \forall d \in D, \forall b \in B$$

3. Identifica cuando dos paralelos de la misma asignatura se realizan simultáneamente (penalización en función objetivo).

$$ZP_{r,k,l,d,b} \geq Y_{r,k,d,b} + Y_{r,l,d,b} - 1$$

$$\forall r \in R, \forall k, l \in K_r : k < l, \forall d \in D, \forall b \in B$$

4. Balance global de la carga horaria (distribuir las clases en los 5 días).

$$\sum_{r \in R} \sum_{k \in K_r} Y_{r,k,d,b} \leq L \quad \forall d \in D, \forall b \in B$$

5. Cada patrón debe cumplir la cantidad de bloques que demanda la asignatura.

$$\sum_{d \in D} \sum_{b \in B} U_{p,d,b} = H_r \quad \forall r \in R, \forall p \in P_r$$

6. No se pueden realizar clases en bloques protegidos.

$$U_{p,d,b} \leq 1 - BP_{d,b} \quad \forall r \in R, \forall p \in P_r, \forall d \in D, \forall b \in B$$

7. Naturaleza de las variables

$$X_{r,k,p} \in \{0,1\} \quad \forall r \in R, \forall k \in K_r, \forall p \in P_r$$

$$ZP_{r,k,l,d,b} \in \{0,1\} \quad \forall r \in R, \forall k, l \in K_r : k < l, \forall d \in D, \forall b \in B$$

$$L \geq 0$$

Cabe mencionar que, para resolver el modelo propuesto, se desarrolla un código en el lenguaje de programación Python, en el cual se incorporan los elementos descritos anteriormente. Esto permite obtener una solución de manera más eficiente y dentro de un tiempo de ejecución razonable. El código utilizado se incluye en los anexos.

6.6 Métricas de desempeño

Por último, se valida el desempeño del modelo mediante el uso de métricas. Para evaluar la calidad de la herramienta de apoyo a la elaboración de horarios, se establecen tres indicadores principales, los cuales se presentan a continuación:

- **Conflictos entre asignaturas:** Mediante este indicador se busca determinar si existen superposiciones entre asignaturas incompatibles dentro de la programación académica. Se consideran incompatibles aquellas asignaturas que, de acuerdo con el grafo de conflictos, no pueden desarrollarse simultáneamente. En este estudio, dicha incompatibilidad corresponde principalmente a las asignaturas pertenecientes a un mismo semestre del plan de estudios.

Cabe señalar que una de las restricciones obligatorias del modelo establece que las asignaturas correspondientes a un mismo semestre no pueden ser programadas en el mismo bloque horario. Por esta razón, se espera que el valor de este indicador sea igual a cero. Su cálculo se realiza principalmente para verificar que la solución obtenida cumple efectivamente con esta condición. En consecuencia,

el indicador permite validar la factibilidad del horario, aunque no necesariamente diferenciar la calidad entre distintas soluciones factibles.

- **Promedio de bloques ociosos (ventanas):** Este indicador mide la cantidad promedio de bloques libres existentes entre el primer y el último bloque con clases de cada día, para cada semestre del plan de estudios correspondiente. Para su cálculo, se utiliza la siguiente fórmula:

$$\text{Promedio de ventanas} = \frac{\text{Cantidad de bloques libres}}{\text{Cantidad de jornadas con clases}}$$

Es importante señalar que, para el cálculo de este indicador, no se consideran los bloques ubicados antes de la primera clase ni aquellos posteriores a la última clase de cada día. Un valor menor representa un horario más compacto y, en consecuencia, una menor cantidad de tiempos de espera para los estudiantes.

- **Porcentaje de bloques horarios utilizados:** Este indicador permite evaluar el nivel de ocupación de los bloques horarios disponibles. Para su cálculo, se compara la cantidad de bloques utilizados con el total de bloques disponibles en cada día. Se considera ocupado aquel bloque en el que se programa al menos una clase, independientemente de la cantidad de asignaturas que puedan desarrollarse simultáneamente sin presentar conflictos.

$$\% \text{ Utilización} = \frac{\text{Cantidad de bloques ocupados}}{\text{Cantidad de bloques disponibles}} \times 100$$

Con el propósito de facilitar su interpretación, el indicador se calcula de manera desagregada por día. Cabe señalar que un valor más elevado no implica necesariamente una mejor solución, ya que podría reflejar una concentración excesiva de las clases. Por esta razón, se utiliza como una métrica complementaria de distribución temporal y se analiza en conjunto con los dos indicadores anteriores.

Para la implementación y el cálculo de estas métricas se utiliza, al igual que en el modelo de asignación, un código desarrollado en Python. El detalle de su implementación se presenta en la sección de anexos.

7 Resultados y Discusión

7.1 Preparación y caracterización de los datos

En este apartado se presentan los resultados obtenidos tras la limpieza de los datos. Tal como se señala en la metodología, se consideran únicamente las asignaturas correspondientes a la carrera de Ingeniería Civil Industrial, específicamente aquellas pertenecientes al Departamento de Industrias.

Tabla 4: Cantidad de filas y columnas de cada base de datos inicialmente.

Base de datos	Filas	Columnas
Cantidad de estudiantes por asignatura	915	20
Estudiantes pendientes	4062	10
Inscripción de asignaturas	1332	12

Nota: Elaboración propia.

La cantidad de registros se reduce de manera considerable, debido a que se eliminan los ramos asociados a los planes de postgrado y vespertino. Asimismo, se retiran las columnas redundantes, lo que permite disponer de una base de datos más acotada y facilita su procesamiento posterior. Además, se incorporan (a la base de datos históricos) columnas relacionadas con la tasa de reprobación y la cantidad de asignaturas pendientes, obtenidas de forma anónima desde la segunda base de datos descrita en la metodología. A continuación, se adjunta una tabla con las filas y columnas de cada base de datos posterior a los procesos mencionados.

Tabla 5: Cantidad de filas y columnas de cada base de datos posterior a la limpieza.

Base de datos	Filas	Columnas
Cantidad de estudiantes por asignatura	423	17
Estudiantes pendientes	4062	10
Inscripción de asignaturas	1167	11

Nota: Elaboración propia.

Un resultado relevante de esta etapa es que la principal fuente de información para el desarrollo del estudio corresponde a la base de datos que contiene los antecedentes históricos de las asignaturas.

Por otro lado, se dispone de bases de datos elaboradas a partir de la malla curricular de cada plan de estudios. Estas presentan la siguiente cantidad de filas y columnas:

Tabla 6: Cantidad de filas y columnas de cada base de datos elaboradas.

Base de datos	Filas	Columnas
Malla antigua (6006)	59	5
Malla nueva (6008)	59	5

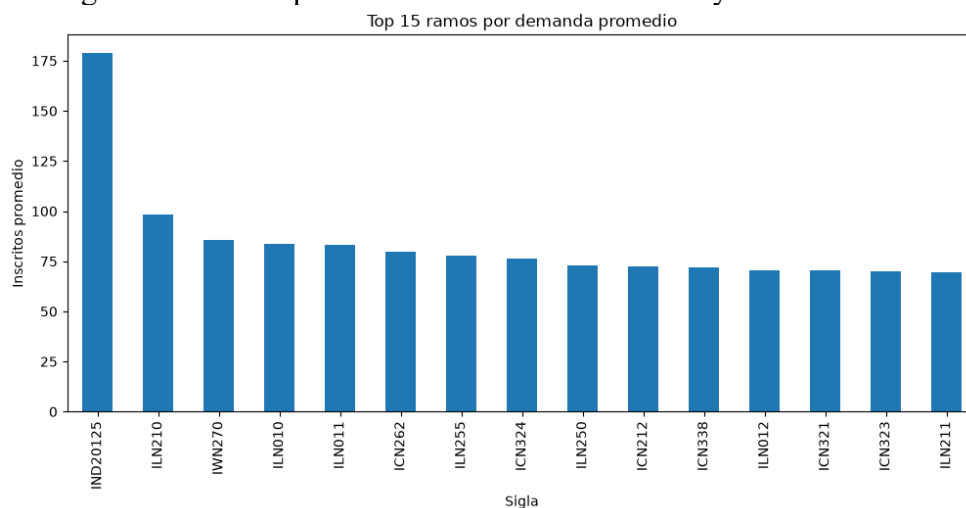
Nota: Elaboración propia.

7.2 Resultados de la estimación de la demanda académica

7.2.1 Comportamiento histórico de la demanda

Una vez finalizado el tratamiento de las bases de datos, se aplica el primer modelo de predicción, correspondiente al promedio de la demanda histórica por semestre, junto con su desviación estándar. Como resultado, se obtiene una base de datos compuesta por 89 filas y 4 columnas. A partir de esta información, se elaboran los siguientes gráficos.

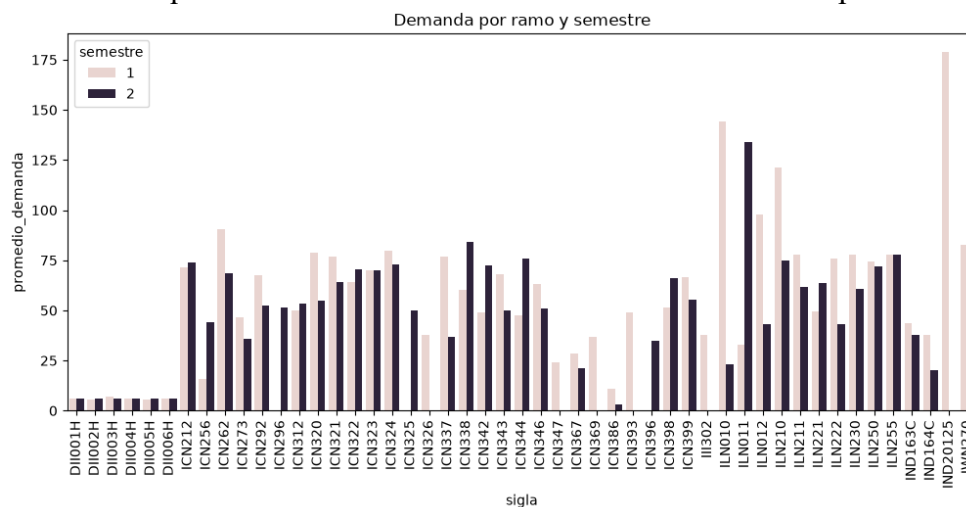
Figura 8: Gráfico que muestra los 15 ramos con mayor demanda.



Nota: Elaboración propia.

En esta figura se observan las asignaturas que presentan una mayor demanda estimada. En consecuencia, son aquellas para las cuales se debe considerar una mayor cantidad de paralelos, con el fin de cubrir los cupos proyectados.

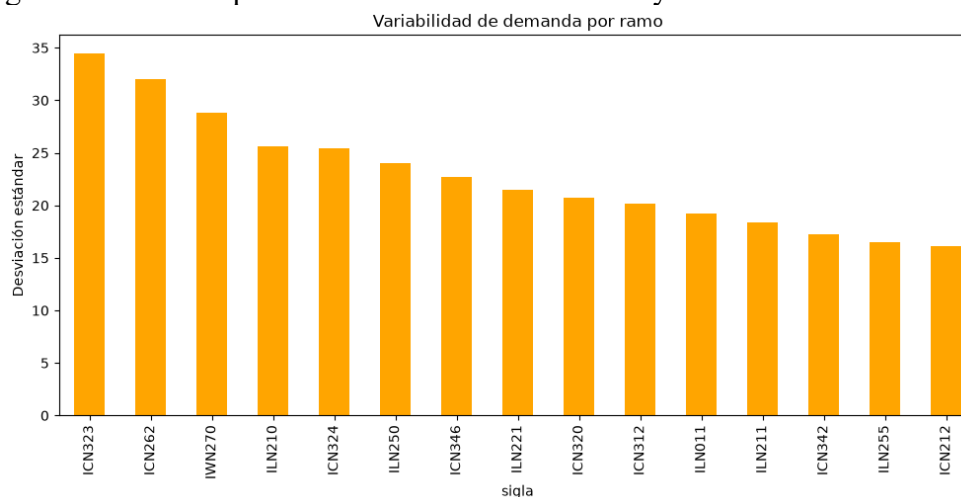
Figura 9: Gráfico que muestra un extracto de la demanda de los ramos por semestre.



Nota: Elaboración propia.

Este gráfico complementa el pronóstico presentado en la Figura 8. En él se observa que la demanda de algunas asignaturas varía entre semestres, lo que podría estar asociado a factores como estudiantes con asignaturas pendientes, alumnos con VTR2, entre otros. Además, existen asignaturas que presentan demanda únicamente en uno de los dos semestres académicos.

Figura 10: Gráfico que muestra los 15 ramos con mayor desviación estándar.



Nota: Elaboración propia.

A partir de este último gráfico, se observa que este método de pronóstico genera resultados con un alto nivel de variabilidad en algunas asignaturas, alcanzando, en el caso más extremo, una diferencia aproximada de 34 cupos respecto del promedio. Por esta razón, junto con la información obtenida en los gráficos anteriores, se descarta este modelo como alternativa para estimar la demanda. La principal justificación es que el método no logra representar adecuadamente el comportamiento de los datos ni incorporar posibles cambios futuros, como un aumento en la cantidad de estudiantes reprobados, entre otros factores.

7.2.2 Desempeño de los modelos predictivos

En este capítulo se implementan técnicas de pronóstico de demanda basadas en machine learning. Para ello, se consideran cuatro modelos, previamente descritos en la metodología. Para evaluar el desempeño de los modelos predictivos, se realiza una partición entrenamiento-prueba, destinando el 80 % de los datos al entrenamiento y el 20 % restante a la evaluación. Las métricas MAE, RMSE y R^2 se calculan sobre el conjunto de prueba. Los resultados obtenidos se presentan a continuación.

- Regresión lineal: En este modelo se busca comprender la relación lineal que tienen las variables existentes con respecto a la variable que se busca predecir, es

decir, la demanda académica de cada asignatura. Al aplicar el modelo se obtienen los siguientes indicadores.

Tabla 7: Indicadores para el modelo de regresión lineal.

Indicador	Valor
MAE	9,771
RMSE	16,034
R^2	0,748

Nota: Elaboración propia.

A partir de estos resultados, se puede establecer lo siguiente. En el caso del error absoluto medio (MAE), mientras menor sea su valor, mayor será la precisión de las predicciones. El valor obtenido indica que, en promedio, la demanda estimada se desvía en aproximadamente 10 estudiantes respecto de la demanda real. Sin embargo, este resultado debe compararse con los demás modelos y con la magnitud habitual de la demanda antes de determinar si el error es elevado.

Por su parte, la raíz del error cuadrático medio (RMSE) penaliza con mayor intensidad los errores de mayor magnitud. El valor de 16,034, superior al MAE, sugiere la presencia de algunas predicciones con desviaciones considerablemente mayores. Finalmente, el coeficiente de determinación (R^2) indica la proporción de la variabilidad de la demanda que es explicada por el modelo. En este caso, un valor de 0,748 significa que el modelo explica el 74,8 % de dicha variabilidad, lo que representa un ajuste relativamente alto, aunque todavía existe un 25,2 % que no es explicado por las variables incorporadas.

- **Árbol de decisión:** A diferencia del modelo anterior, este método busca predecir la demanda mediante una estructura en forma de árbol, en la cual se toman decisiones sucesivas a partir de las características de los datos. Al aplicar este

modelo, se obtienen los siguientes resultados.

Tabla 8: Indicadores para el modelo de árbol de decisión.

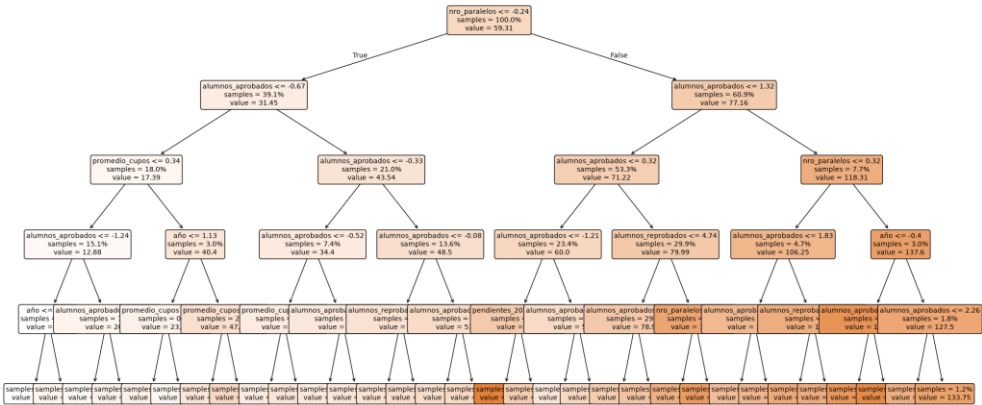
Indicador	Valor
MAE	5,634
RMSE	9,967
R ²	0,922

Nota: Elaboración propia.

A partir de los resultados presentados en la Tabla 8, se observa una mejora respecto del modelo anterior. El error absoluto medio disminuye a 5,634, lo que indica que las predicciones se desvían, en promedio, en aproximadamente 6 estudiantes respecto de la demanda real. Por lo tanto, el modelo presenta una mayor precisión. Asimismo, la raíz del error cuadrático medio se reduce a 9,967, lo que refleja una menor presencia de errores de gran magnitud. Finalmente, el coeficiente de determinación alcanza un valor de 0,922, lo que significa que el modelo explica el 92,2 % de la variabilidad de la demanda y presenta un mejor ajuste a los datos.

Figura 11: Árbol de decisión.

Árbol de Decisión - Predicción de Demanda Académica



Nota: Elaboración propia.

Para complementar el análisis anterior, se presenta el árbol de decisión obtenido a

partir del modelo.

- Random Forest: Este modelo utiliza una estrategia similar a la del árbol de decisión, pero combina múltiples árboles en lugar de emplear solo uno. Esto permite interpretar mejor los datos y generar una predicción más robusta. Los resultados obtenidos se presentan a continuación.

Tabla 9: Indicadores para el modelo de Random Forest.

Indicador	Valor
MAE	3,552
RMSE	7,434
R^2	0,957

Nota: Elaboración propia.

A partir de los resultados obtenidos, se observa que el modelo presenta una mayor precisión en la predicción de la demanda académica, dado que el MAE disminuye a 3,552. De igual manera, el RMSE alcanza un valor de 7,434, lo que evidencia una menor presencia de errores de gran magnitud. Finalmente, se obtiene un R^2 de 0,957, lo que indica que las estimaciones generadas siguen de manera muy cercana el comportamiento observado en los datos y superan el desempeño de los dos modelos anteriores.

- Gradient Boosting: Este modelo continúa utilizando árboles de decisión como base, pero aplica una estrategia diferente. Cada nuevo árbol busca corregir los errores generados por el anterior, repitiendo este proceso de manera sucesiva para mejorar la predicción. Para este modelo, se obtienen los siguientes indicadores.

Tabla 10: Indicadores para el modelo de Gradient Boosting.

Indicador	Valor
MAE	4,673
RMSE	9,281
R^2	0,916

Nota: Elaboración propia.

Puesto que corresponde al último modelo empleado para estimar la demanda de las asignaturas de Ingeniería Civil Industrial, es posible contrastar sus resultados con los obtenidos anteriormente. Si bien presenta un MAE relativamente bajo, este sigue siendo superior al registrado por el modelo Random Forest. Además, el RMSE es cercano al obtenido con el árbol de decisión, lo que refleja una mayor presencia de errores de gran magnitud. Finalmente, el coeficiente de determinación alcanza un valor de 0,916, lo que indica que el modelo explica aproximadamente el 91,6 % de la variabilidad de la demanda. Aunque corresponde a un buen nivel de ajuste, se encuentra por debajo de los resultados obtenidos por Random Forest y por el árbol de decisión.

7.2.3 Selección del modelo

Posterior a la aplicación de los cuatro modelos de machine learning y a la obtención de sus respectivas métricas de desempeño, es posible realizar una comparación para determinar cuál interpreta de mejor manera el comportamiento de la demanda académica. Esto permite identificar el modelo que entrega estimaciones más precisas sobre la cantidad esperada de estudiantes por asignatura. Para facilitar este análisis, se presenta una tabla comparativa con los resultados obtenidos.

Tabla 11: Indicadores para todos los modelos.

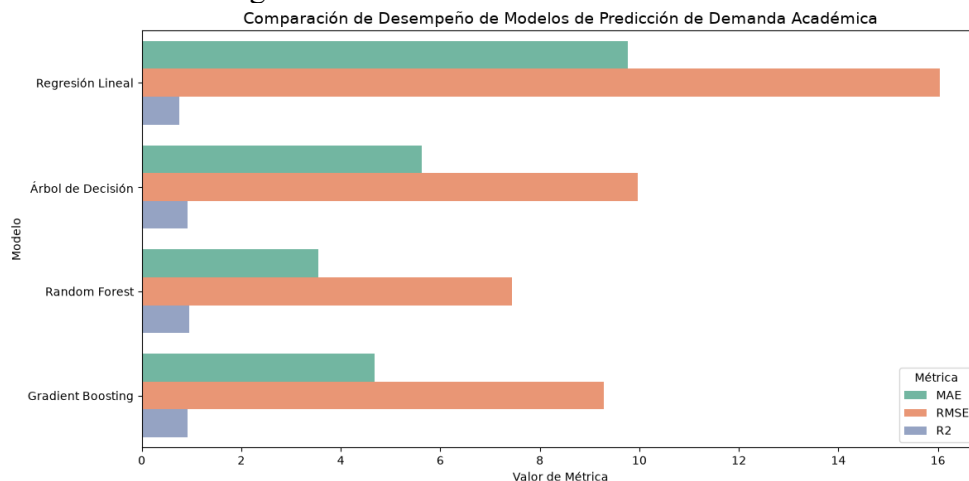
Indicador	RL	AD	RF	GB
MAE	9,771	5,634	3,552	4,673
RMSE	16,034	9,967	7,434	9,281
R ²	0,748	0,922	0,957	0,916

Nota: Elaboración propia.

Al observar la Tabla 11, se identifica que el modelo Random Forest (RF) presenta los mejores indicadores de desempeño. En primer lugar, obtiene el menor valor de MAE, lo que indica una mayor precisión en las estimaciones. En segundo lugar, también presenta el RMSE más bajo, lo que refleja una menor presencia de desviaciones de gran magnitud y, por lo tanto, predicciones más consistentes. Finalmente, alcanza el coeficiente de determinación más cercano a 1, lo que evidencia una mejor capacidad para representar el comportamiento de los datos. En consecuencia, al considerar las tres métricas en conjunto, se determina que Random Forest corresponde a la mejor alternativa para predecir la demanda de estudiantes en las asignaturas del Departamento de Industrias.

Para complementar este análisis, se presenta un gráfico que permite observar y comparar el desempeño de los distintos modelos.

Figura 12: Indicadores de los modelos.



Nota: Elaboración propia.

Teniendo en cuenta esta selección, se realiza la predicción obteniendo los siguientes resultados.

Tabla 12: Predicciones obtenidas con RF.

Sigla	Demanda Predicha
DII001H	15
DII002H	15
DII003H	15
DII004H	15
DII005H	15
DII006H	14
ICN212	97
ICN256	60
ICN262	120
ICN273	44
ICN292	86
ICN296	63
ICN312	82
ICN320	107
ICN321	91
ICN322	97
ICN323	137
ICN324	120
ICN325	51
ICN326	48
ICN337	89
ICN338	98
ICN342	91
ICN343	75
ICN344	89
ICN346	95
ICN347	39
ICN367	40
ICN369	46
ICN386	13
ICN393	55
ICN396	36
ICN398	75
ICN399	89
III302	38
ILN010	159
ILN011	153
ILN012	121
ILN210	132

ILN211	106
ILN221	105
ILN222	93
ILN230	103
ILN250	96
ILN255	107
IND163C	55
IND164C	67
IND20125	135
IWN270	139

Nota: Elaboración propia.

Es relevante señalar que las predicciones obtenidas corresponden, en su mayoría, a asignaturas pertenecientes a la malla antigua. Esto se debe principalmente a la información disponible proporcionada por la universidad y a la reciente implementación de la nueva malla curricular.

7.3 Resultados de los grafos de conflicto

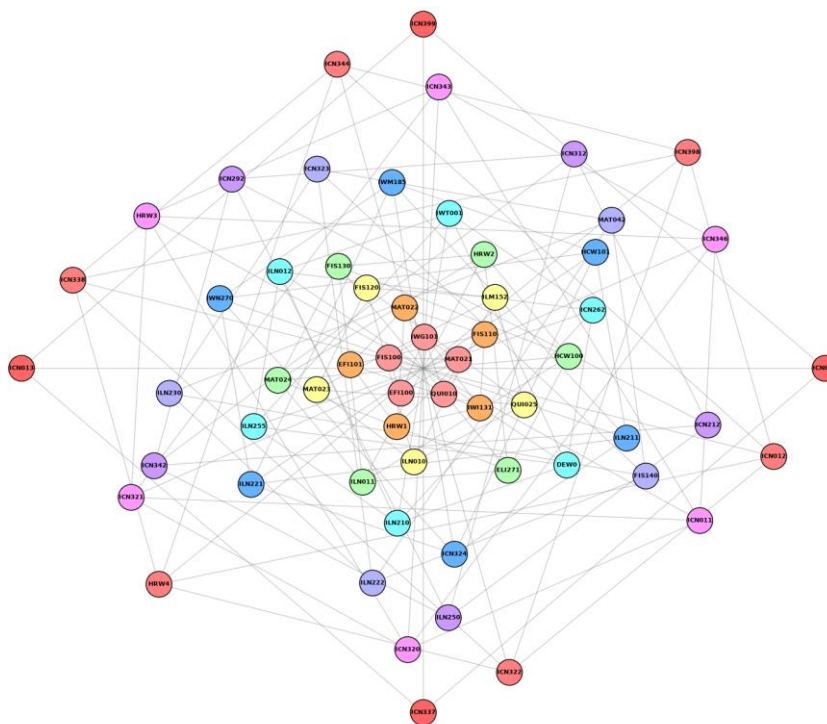
Para que el modelo se desarrolle sin inconvenientes, se construyen los grafos de conflicto, mediante los cuales se identifican los posibles cruces entre asignaturas de un mismo semestre. Es relevante señalar que se elaboran cuatro grafos de conflicto, agrupados según correspondan al plan de estudios antiguo o al plan nuevo.

- Plan de estudios 6006

Como se señaló anteriormente, el plan 6006 corresponde al plan de estudios antiguo. Para esta malla curricular se desarrollan dos grafos principales. El primero presenta, de manera conjunta y ordenada, la totalidad de los conflictos identificados. El segundo corresponde al mismo grafo, pero agrupado por semestre, lo que permite realizar un análisis más específico y facilita su interpretación visual. A continuación, se presentan los grafos desarrollados.

Figura 13: Grafo de conflictos completo para el plan 6006.

Grafo de Conflictos Completo - USM (Organizado por Semestre)

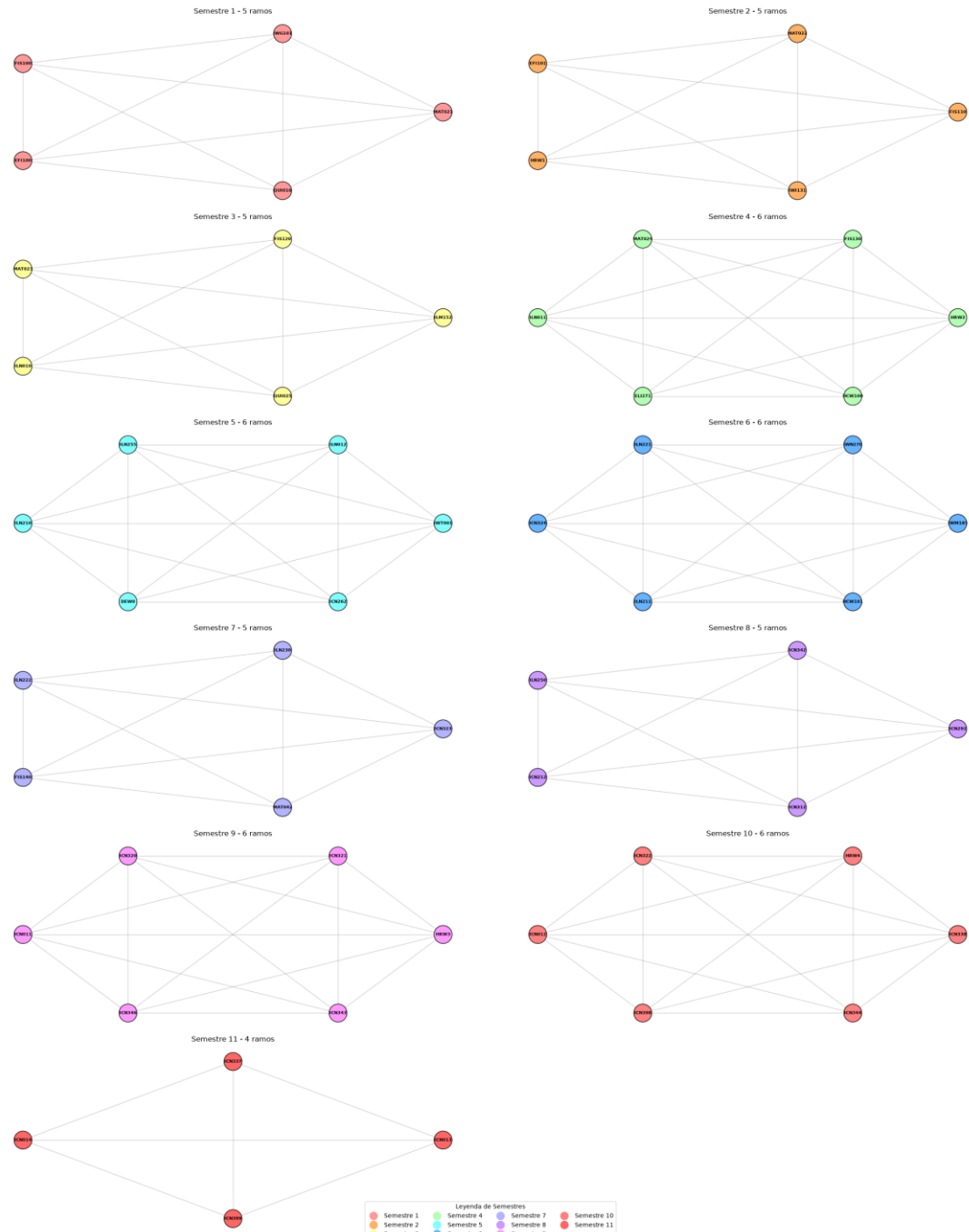


Nota: Elaboración propia.

Como se observa en la Figura 13, se presenta un grafo en el cual las aristas conectan las asignaturas, representadas por nodos, que presentan conflictos entre sí y que, por lo tanto, no deben ser asignadas en un mismo horario. Además, cada semestre se representa mediante un color distintivo, lo que facilita la identificación de las asignaturas correspondientes a un mismo nivel de la carrera. Por ejemplo, los nodos de color amarillo representan las asignaturas pertenecientes al tercer semestre, mientras que los nodos verdes corresponden al cuarto semestre, y así sucesivamente. Complementariamente, la disposición del grafo se organiza desde los semestres inferiores hacia los superiores, ubicando estos últimos en las zonas más externas. En consecuencia, mientras más alejado del centro se encuentra un nodo, mayor es el semestre al que pertenece según el plan de estudios.

Figura 14: Grafo de conflictos segmentado para el plan 6006.

Mini-Grafos de Conflictos - Malla Antigua



Nota: Elaboración propia.

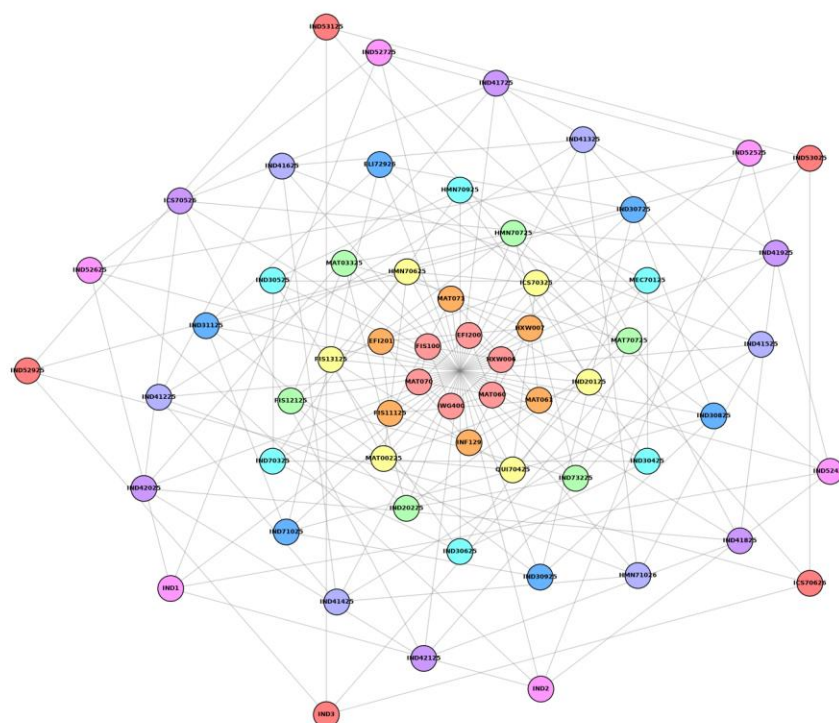
Para facilitar el análisis y la identificación de conflictos, se presenta el mismo grafo agrupado por semestre. Cabe mencionar que se mantienen los colores asignados a cada nivel y que su interpretación permanece sin cambios: las aristas representan los conflictos entre asignaturas y los nodos corresponden a los distintos ramos.

- Plan de estudios 6008

Para el nuevo plan de estudios se elaboran los mismos grafos, con el propósito de analizar ambos planes bajo los mismos criterios. A continuación, se presentan los resultados obtenidos.

Figura 15: Grafo de conflictos completo para el plan 6008.

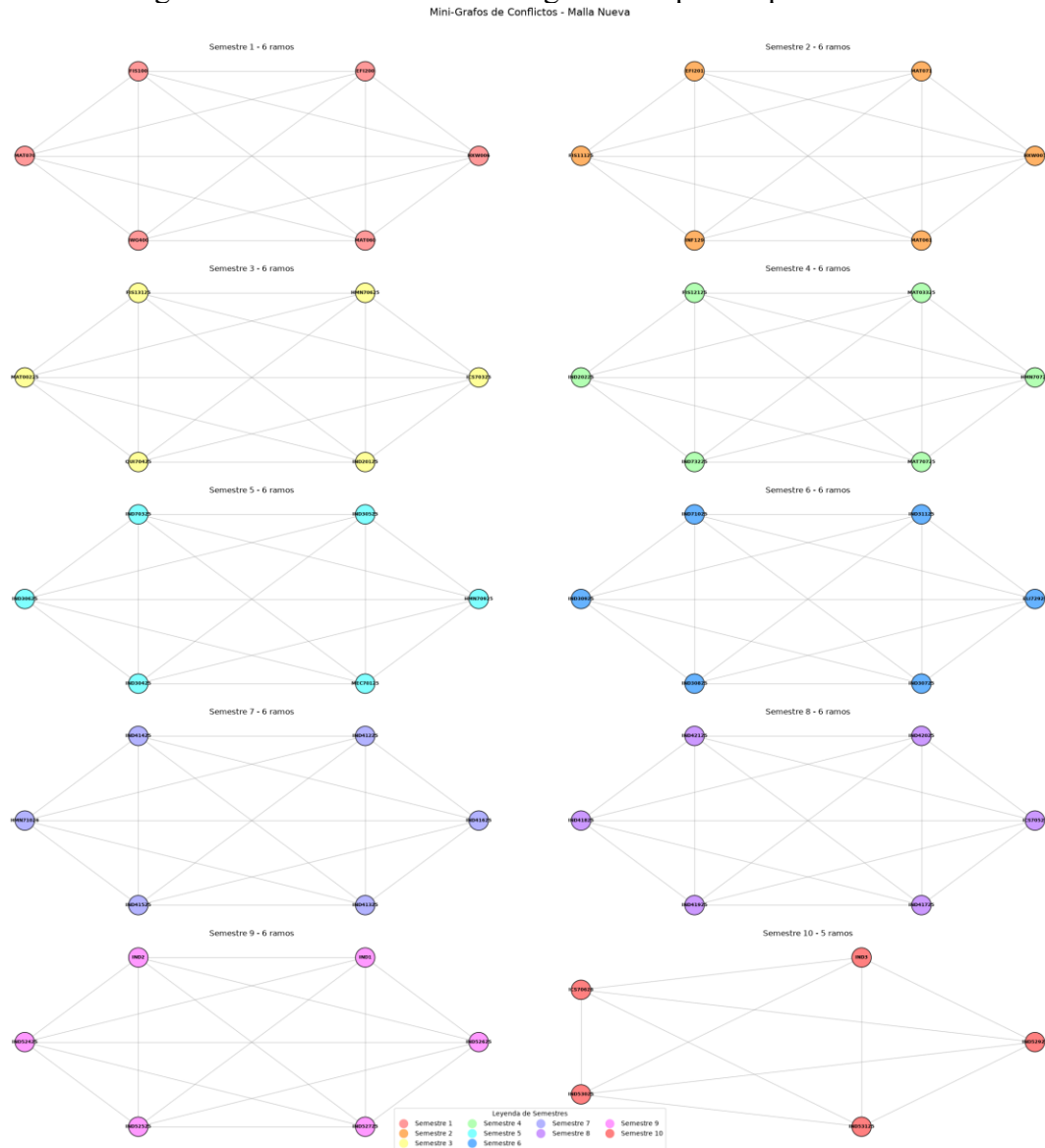
Grafo de Conflictos Completo - USM (Organizado por Semestre)



Nota: Elaboración propia.

Se mantiene el mismo criterio de análisis utilizado para el grafo del plan 6006. Los nodos representan las asignaturas pertenecientes al plan de estudios 6008, mientras que las aristas indican los conflictos existentes entre ellas, asociados a incompatibilidades horarias. Además, se asigna un color característico a cada semestre para facilitar su identificación.

Figura 16: Grafo de conflictos segmentado para el plan 6008.



Nota: Elaboración propia.

La Figura 16 corresponde al cuarto y último grafo desarrollado para representar los conflictos. Su estructura es la misma que la utilizada para la malla antigua en la Figura 14. En este caso, los conflictos se presentan segmentados por semestre, donde los nodos representan las asignaturas y las aristas corresponden a los conflictos entre ellas. Además, cada semestre se diferencia mediante un color específico para facilitar su identificación.

- Comparación de ambos planes

Es relevante señalar las principales diferencias identificadas entre los grafos de conflicto

desarrollados. Una de ellas corresponde a la cantidad de semestres que compone cada plan de estudios, ya que uno contempla 11 semestres y el otro 10. Además, dado que la cantidad de asignaturas se mantiene, en la nueva malla existen semestres que concentran un mayor número de conflictos como consecuencia de la reducción de un semestre respecto de la malla antigua.

7.4 Resultados del modelo MILP

7.4.1 Supuestos del modelo

Para el desarrollo adecuado del modelo de programación lineal entera mixta, se establecen ciertos supuestos que permiten obtener resultados consistentes con lo planteado. El principal corresponde a los valores asociados a los parámetros incorporados en el modelo. A continuación, se presentan los ponderadores utilizados en la función objetivo.

- $\alpha = 1 \rightarrow$ Se utiliza como valor de referencia para penalizar los patrones horarios menos convenientes.
- $\beta = 0,08 \rightarrow$ Se asigna un valor bajo a este ponderador, debido a que considera la cantidad de estudiantes asociados a una asignatura ubicada en un bloque tardío. De esta forma, se genera una penalización sin que este criterio domine la función objetivo.
- $\gamma = 15 \rightarrow$ Se establece un valor elevado, ya que se busca evitar que dos paralelos de una misma asignatura sean programados simultáneamente.
- $\delta = 0,2 \rightarrow$ Se asigna un valor moderado, debido a que se relaciona con la concentración de clases dentro del modelo y corresponde a un criterio complementario. Su objetivo es favorecer una distribución más equilibrada de las asignaturas a lo largo de la semana.

Estos parámetros corresponden a valores definidos de manera experimental, considerando el contexto y los objetivos del modelo. En caso de ser necesario, pueden ser modificados para futuras aplicaciones o escenarios. Además, se establece que las asignaturas

pueden ser programadas de lunes a viernes y se considera como un bloque protegido los bloques 7-8 de los miércoles.

7.4.2 Resultados computacionales

Tomando en consideración los supuestos establecidos, la predicción de la demanda y las condiciones específicas planteadas, se desarrolla un modelo MILP orientado a determinar la factibilidad de asignar horarios académicos. Para su implementación, se utiliza un código en Python que sigue una serie de etapas generales. En primer lugar, se incorpora la demanda académica y se calcula la cantidad de paralelos requerida para cubrir la demanda estimada junto con la capacidad de cada uno. Posteriormente, se generan patrones con posibles alternativas horarias para las asignaturas del Departamento de Industrias. Finalmente, esta información se integra con las restricciones definidas en el modelo y se obtiene como resultado una asignación tabulada en un archivo Excel. Los resultados obtenidos mediante este proceso se presentan a continuación.

Tabla 13: Resumen del modelo.

Indicador	Valor
Estado del modelo	Factible - límite de tiempo
Valor función objetivo	61,4
Carga máxima global L	7
Cantidad de ramos en R	28
Cantidad de ramo-paralelos	82
Cap mínimo	25
Cap máximo	45
Bloques protegidos	[('Miércoles', 7), ('Miércoles', 8)]
Bloques tardíos	[15, 16]
Máximo de patrones por ramo	250
alpha	1
beta	0,08
gamma	15
delta	0,2

Nota: Elaboración propia.

En este primer apartado entregado por el modelo se resumen los principales resultados obtenidos y se indica si existe o no una solución factible. En este caso, el modelo confirma que

se encuentra una solución factible. Además, se presenta el valor de la función objetivo, correspondiente a 61,4. Este resultado indica que el modelo logra minimizar la suma ponderada de los distintos términos que componen la función objetivo, respetando todas las restricciones establecidas. Por último, es relevante señalar que el modelo se resuelve mediante el solver CBC, implementado a través de la librería PuLP en Python, estableciendo un tiempo máximo de ejecución de 300 segundos (5 minutos). La ejecución finaliza al alcanzar el límite de tiempo establecido, por lo que no se certifica la optimalidad global de la solución encontrada. No obstante, la solución con valor 61,4 se obtiene aproximadamente a los 104 segundos de ejecución y no es mejorada durante los más de tres minutos restantes, lo que evidencia estabilidad en la mejor solución encontrada dentro del tiempo disponible.

Tabla 14: Cruce entre mallas y predicción de RF.

Indicador	Valor
Siglas únicas en las mallas	97
Siglas únicas con predicción RF	49
Siglas incluidas en R	28
Predicciones RF sin coincidencia en malla	21
Registros de malla sin predicción RF	69

Nota: Elaboración propia.

También se obtiene una tabla en la que se detalla el cruce entre las mallas antigua, nueva y el archivo que contiene las asignaturas para las cuales se realiza la predicción de la demanda. En esta etapa se evidencia que un número importante de asignaturas queda sin predicción y, por lo tanto, no es considerado en el modelo de asignación de horarios. Esto se debe a que solo se incluyen las asignaturas pertenecientes al Departamento de Industrias que no corresponden a alguna especialidad. En consecuencia, el modelo trabaja con 28 asignaturas que cumplen con estos criterios.

Tabla 15: Ramos que no deben tener conflictos por semestre.

Malla	Semestre	Ramos incluidos en R_m_s
6006	3	ILN010
6006	4	ILN011
6006	5	ICN262, ILN012, ILN210, ILN255
6006	6	ICN324, ILN211, ILN221, IWN270
6006	7	ICN323, ILN222, ILN230
6006	8	ICN212, ICN292, ICN312, ICN342, ILN250
6006	9	ICN320, ICN321, ICN343, ICN346
6006	10	ICN322, ICN338, ICN344, ICN398
6006	11	ICN337
6008	3	IND20125

Nota: Elaboración propia.

En la tabla adjunta se presentan las asignaturas que cumplen con las condiciones establecidas para ser consideradas dentro del modelo.

Posteriormente, el modelo también proporciona hojas de Excel en las cuales se tabulan los patrones horarios seleccionados en base a los paralelos de cada asignatura y las restricciones del modelo.

7.4.3 Asignación de bloques horarios

En cuanto a la asignación de bloques horarios, el modelo proporciona una tabla con la selección del patrón horario más conveniente y los datos asociados a dicha elección, como los paralelos correspondientes. Además, entrega una matriz compuesta por los días, el número de bloque y las asignaturas asignadas a cada uno. Para facilitar el análisis visual de los resultados por semestre, se desarrolla en Excel una matriz automatizada que permite visualizar las asignaturas correspondientes a cada nivel y verificar que no existan superposiciones horarias.

Figura 17: Horario para el semestre 3 del plan 6006.

Bloque	Lunes	Martes	Miércoles	Jueves	Viernes
1		ILN010-P1			
2		ILN010-P1			
3	ILN010-P3		ILN010-P2	ILN010-P4	
4	ILN010-P3		ILN010-P2	ILN010-P4	
5	ILN010-P4				ILN010-P3
6	ILN010-P4				ILN010-P3
7					
8					
9	ILN010-P2		ILN010-P1		
10	ILN010-P2		ILN010-P1		
11					
12					
13					
14					
15					
16					

Nota: Elaboración propia.

En la Figura 17 se presenta el horario generado por el modelo de programación lineal entera mixta para el tercer semestre del plan de estudios antiguo. En él se observan los paralelos necesarios para satisfacer la demanda académica y los bloques horarios seleccionados por el modelo. Es importante destacar que no se presentan superposiciones entre asignaturas y que las clases se distribuyen a lo largo de la semana, con el objetivo de facilitar la construcción de los horarios de los estudiantes. Además, se observa una preferencia por los bloques de la mañana. El modelo no prohíbe la utilización de horarios más tardíos, pero, cuando existen alternativas factibles, prioriza la asignación en bloques más tempranos.

Figura 18: Horario para el semestre 8 del plan 6006.

Bloque	Lunes	Martes	Miércoles	Jueves	Viernes
1	ILN250-P3	ICN312-P2	ICN292-P1	ICN312-P2	ICN312-P2
2	ILN250-P3	ICN312-P2	ICN292-P1	ICN312-P2	ICN312-P2
3	ICN312-P1	ICN342-P1	ICN292-P2	ICN342-P3	ICN312-P1
4	ICN312-P1	ICN342-P1	ICN292-P2	ICN342-P3	ICN312-P1
5	ICN342-P1	ICN212-P2	ICN312-P2	ICN312-P1	ICN292-P2
6	ICN342-P1	ICN212-P2	ICN312-P2	ICN312-P1	ICN292-P2
7	ICN342-P3	ICN342-P2		ICN342-P2	ICN212-P3
8	ICN342-P3	ICN342-P2		ICN342-P2	ICN212-P3
9	ILN250-P1 / ILN250-P2	ICN212-P1	ICN212-P1	ICN292-P1	ILN250-P1
10	ILN250-P1 / ILN250-P2	ICN212-P1	ICN212-P1	ICN292-P1	ILN250-P1
11	ICN212-P3	ICN312-P1	ILN250-P2 / ILN250-P3	ICN212-P2	
12	ICN212-P3	ICN312-P1	ILN250-P2 / ILN250-P3	ICN212-P2	
13					
14					
15					
16					

Nota: Elaboración propia.

Para complementar el análisis anterior, se presenta el horario diseñado para el octavo semestre del plan 6006. En este se observa cómo se asignan los bloques horarios a las distintas asignaturas sin generar conflictos. Además, en el bloque 9-10 del lunes se ubican dos paralelos de una misma asignatura. Esto refleja otro criterio del modelo, ya que se busca evitar la superposición de paralelos de una misma asignatura cuando es posible, aunque pueden coincidir en un mismo horario si la solución lo requiere. Esta situación no incumple las restricciones del modelo, aunque reduce las alternativas horarias disponibles para los estudiantes; por esta razón, se incorpora como un criterio penalizado y no como una prohibición absoluta.

7.5 Evaluación del modelo obtenido

Para finalizar el análisis del proyecto, se evalúan tres métricas diseñadas específicamente para el contexto de asignación de horarios del Departamento de Industrias de la carrera de Ingeniería Civil Industrial. Los resultados obtenidos se presentan a continuación.

- **Conflictos entre asignaturas:** Este indicador busca representar la cantidad de conflictos existentes dentro del modelo. Por definición, su valor debe ser igual a cero; sin embargo, se realiza esta validación con el fin de comprobar el cumplimiento de una de las

restricciones fundamentales del modelo.

Tabla 16: Validación de conflictos entre asignaturas.

Plan	Semestre	Asignaturas consideradas	Pares evaluados	Pares con conflicto	Cantidad de superposiciones	Resultado
6006	3	1	0	0	0	Sin superposiciones
6006	4	1	0	0	0	Sin superposiciones
6006	5	4	54	0	0	Sin superposiciones
6006	6	4	63	0	0	Sin superposiciones
6006	7	3	33	0	0	Sin superposiciones
6006	8	5	67	0	0	Sin superposiciones
6006	9	4	45	0	0	Sin superposiciones
6006	10	4	37	0	0	Sin superposiciones
6006	11	1	0	0	0	Sin superposiciones
6008	3	1	0	0	0	Sin superposiciones
TOTAL			299	0	0	Sin superposiciones

Nota: Elaboración propia.

Como se evidencia en la tabla adjunta, se realiza una verificación completa del horario y se obtiene el resultado esperado, es decir, la ausencia de conflictos.

- **Promedio de bloques ociosos:** Como se detalla en metodología, este indicador busca establecer un promedio de las ventanas existentes entre clases. Teniendo eso en cuenta, se obtienen los siguientes resultados.

Tabla 17: Promedio de bloques libres.

Plan	Semestre	Jornadas con clases	Bloques ociosos	Promedio
6006	3	5	4	0,8
6006	4	5	10	2
6006	5	5	2	0,4
6006	6	5	0	0
6006	7	5	10	2
6006	8	5	0	0
6006	9	5	4	0,8
6006	10	5	10	2
6006	11	3	6	2
6008	3	4	4	1

Nota: Elaboración propia.

En base a los valores obtenidos, se puede establecer que, en general, las ventanas entre asignaturas pertenecientes a un mismo semestre presentan valores reducidos. Esto indica que los estudiantes cuentan, en promedio, con menores tiempos de espera entre clases, lo que contribuye a una distribución más eficiente del horario. Además, se destaca que en los semestres 6 y 8 del plan 6006 no existen bloques ociosos. Por otra parte, en los demás semestres el promedio no supera los 2 bloques libres por jornada.

- **Porcentaje de bloques utilizados:** El objetivo de esta métrica es representar el porcentaje de utilización de los bloques disponibles, es decir, determinar cuántos de los bloques diarios se encuentran ocupados en el horario generado por el modelo. Los resultados obtenidos se muestran a continuación.

Tabla 18: Porcentaje de bloques utilizados.

Día	Bloques disponibles	Bloques ocupados	% de utilización
Lunes	16	12	75
Martes	16	12	75
Miércoles	14	10	71,43
Jueves	16	12	75
Viernes	16	10	62,5
Total	78	56	71,79

Nota: Elaboración propia.

A partir de los resultados se observa que, en general, existe una utilización similar de los

bloques horarios durante la semana. Los lunes, martes y jueves presentan la mayor ocupación, con un 75 %, mientras que el miércoles alcanza un 71,43 %. En cambio, el viernes registra el menor porcentaje de utilización, con un 62,5 %. Considerando el total de bloques disponibles, se obtiene una utilización global de 71,79 %, lo que refleja una distribución relativamente equilibrada de las clases durante la semana.

8 Conclusiones y Recomendaciones

8.1 Conclusiones

Para finalizar este proyecto, es importante presentar las conclusiones pertinentes respecto de lo propuesto inicialmente, su desarrollo y los resultados obtenidos. Para ello, en este capítulo se presenta una síntesis del proceso completo.

En cuanto a la preparación y calidad de los datos, fue necesario consolidar, limpiar y cruzar distintas fuentes de información antes de desarrollar los modelos propuestos. Esta etapa fue fundamental para establecer una base sólida tanto para la estimación de la demanda de estudiantes por asignatura del Departamento de Industrias como para la optimización de la asignación de horarios académicos. Cabe destacar que la principal fuente de información correspondió al archivo con los antecedentes históricos de demanda.

Por otro lado, en el proceso de estimación de la demanda, el promedio histórico resultó insuficiente, debido a la elevada desviación estándar observada entre periodos. Por esta razón, se recurrió a técnicas de machine learning con el objetivo de obtener un pronóstico más completo y preciso. Al comparar cuatro modelos con enfoques distintos, se seleccionó aquel que presentó los mejores indicadores de desempeño, es decir, menor MAE, menor RMSE y mayor R^2 . Este correspondió a Random Forest, ya que mostró un mejor ajuste a los datos y entregó estimaciones más precisas y confiables.

Respecto a la elaboración de los grafos de conflicto, estos permitieron representar de manera más clara las incompatibilidades horarias existentes entre las asignaturas y transformarlas en información útil para el desarrollo del modelo matemático. Además, permitieron visualizar de forma rápida y ordenada la composición de cada plan de estudios y los posibles conflictos presentes en cada semestre.

En relación con el desarrollo del modelo MILP, este corresponde a uno de los puntos centrales del proyecto. Fue posible formular e implementar un modelo capaz de integrar la

demanda académica estimada mediante Random Forest, calcular el número de paralelos, generar patrones horarios y ajustarse a las restricciones específicas del caso planteado. El modelo encontró una solución factible de acuerdo con la formulación establecida, con un tiempo de ejecución aproximado de cinco minutos. Este resultado es especialmente relevante, ya que, al tratarse de una herramienta de apoyo, permite al jefe de carrera contar con una alternativa viable en un tiempo acotado.

Como último punto, se destaca que no solo se encontró una solución factible, sino también un resultado con una buena calidad de horarios. Esto se evidencia en los indicadores obtenidos, ya que el modelo fue evaluado en su totalidad mediante la comparación de 299 pares de asignaturas, sin identificar ninguna superposición que representara un conflicto. Además, la cantidad de bloques libres dentro de los horarios generados varía entre 0 y 2, dependiendo del semestre, lo que indica una distribución más compacta. Finalmente, se alcanza una utilización global cercana al 72 %, lo que refleja una distribución equilibrada de las clases a lo largo de la semana. En conjunto, estos resultados indican que el horario proporcionado presenta una buena calidad, con una estructura compacta y una adecuada distribución temporal.

A modo de síntesis general, fue posible desarrollar un sistema de apoyo a la toma de decisiones que integra la predicción de la demanda académica con la optimización de la asignación de horarios mediante programación lineal entera mixta. Entre los principales beneficios del proyecto se encuentra la posibilidad de anticipar la demanda, determinar la cantidad de paralelos necesarios para cubrirla y generar una solución factible sin conflictos, considerando las restricciones específicas del caso estudiado. Todo esto se plantea como una herramienta de apoyo para el jefe de carrera, permitiéndole contar con alternativas viables y tomar decisiones basadas en datos.

8.2 Recomendaciones

Por otro lado, se pueden establecer ciertas recomendaciones orientadas a mejorar la eficiencia del modelo y ampliar sus funcionalidades.

- La primera recomendación sería ampliar la cobertura del modelo. Si bien el alcance de esta memoria se limita a asignaturas específicas del Departamento de Industrias, a futuro podría extenderse a asignaturas de otros departamentos que también forman parte de la carrera de Ingeniería Civil Industrial. Esto permitiría desarrollar un modelo más completo y reducir la dependencia de los horarios definidos por Casa Central, entregando al campus una mayor autonomía para elaborar su propia planificación académica.
- La segunda recomendación corresponde a ampliar la base de datos asociada al nuevo plan de estudios. Actualmente, la información disponible es limitada debido a su reciente implementación. A futuro, será posible incorporar una mayor cantidad de registros históricos al modelo de predicción de demanda, lo que permitiría mejorar sus estimaciones y, en consecuencia, ampliar el alcance del modelo de asignación de horarios.
- Por último, se recomienda considerar las preferencias horarias de los profesores antes de generar el horario. Esto permitiría desarrollar un modelo más ajustado a la disponibilidad y a las condiciones particulares de cada docente. Para ello, sería necesario incorporar un nuevo término en la función objetivo que represente dichas preferencias horarias.

$$Pref_{r,d,b} = \begin{cases} 0, & \text{si se prefiere} \\ 1, & \text{si son poco preferidos} \\ 2, & \text{si es altamente inconveniente} \end{cases}$$

- Además, se debe incorporar un parámetro (μ), encargado de ponderar la importancia de las preferencias horarias de los docentes dentro de la función objetivo. Una vez definido este parámetro, sería necesario añadir un nuevo término a la función objetivo con una estructura similar a la siguiente:

$$+\mu \sum_{r \in R} \sum_{k \in K_r} \sum_{d \in D} \sum_{b \in B} Y_{r,k,d,b} Pref_{r,d,b}$$

9 Referencias

1. Subsecretaría de Educación Superior. (2025). *Informe 2025: Matrícula en educación superior en Chile* (julio 2025). Gobierno de Chile.
2. Barrón Tobías, L. del C. (2018). *Problema de asignación de horarios: Caso de estudio en una institución de educación superior mexicana* [Tesis de maestría, Universidad Autónoma de Nuevo León].
3. Mejía Caballero, J. M., & Paternina Arboleda, C. (2010). Asignación de horarios de clases universitarias mediante algoritmos evolutivos. *Revista Educación en Ingeniería*, 5(9), 140–149.
4. Páez Amaya, N. (2019). *Algoritmo para la planificación de los horarios de clases en el programa de Ingeniería de Sistemas para la Universidad de Pamplona, Norte de Santander* [Trabajo de grado, Universidad de Pamplona].
5. Ramos, A., Sánchez, P., Ferrer, J. M., Barquín, J., & Linares, P. (2010). *Modelos matemáticos de optimización*. Universidad Pontificia Comillas.
6. Guédez Fernández, C. (2011). *Programación lineal e ingeniería industrial: Una aproximación al estado del arte*. **Ingeniería Industrial. Actualidad y Nuevas Tendencias**, 2(6), 61–78.
7. Nemirovski, A. (2023). *Introduction to linear optimization (ISyE 6661)* [Apuntes de curso]. Georgia Institute of Technology.
8. Vitoriano, B., & Ramos, A. (2023). *Programación matemática: Métodos de optimización*. Universidad Complutense de Madrid.
9. Bermúdez Colina, Y. (2011). *Aplicaciones de programación lineal, entera y mixta*. **Ingeniería Industrial. Actualidad y Nuevas Tendencias**, 2(7), 85–104. Universidad de

Carabobo.

10. Mallari, C. B., San Juan, J. L., & Li, R. (2023). *The university coursework timetabling problem: An optimization approach to synchronizing course calendars*. **Computers & Industrial Engineering**, **184**, 109561. <https://doi.org/10.1016/j.cie.2023.109561>
11. Dwivedi, H. V. (2011). *Analysis and implementation of room assignment problem and Cannon's algorithm on general purpose programmable graphical processing units with CUDA* (Tesis de maestría). Clemson University.
12. Gu, X., Krish, M., Sohail, S., Thakur, S., Sabrina, F., & Fan, Z. (2025). *From integer programming to machine learning: A technical review on solving university timetabling problems*. **Computation**, **13**(1), 10. <https://doi.org/10.3390/computation13010010>
13. Babaei, H., Karimpour, J., & Hadidi, A. (2015). *A survey of approaches for university course timetabling problem*. **Computers & Industrial Engineering**, **86**, 43–59. <https://doi.org/10.1016/j.cie.2014.11.010>
14. Chen, M. C., Sze, S. N., Goh, S. L., Sabar, N. R., & Kendall, G. (2021). *A survey of university course timetabling problem: Perspectives, trends and opportunities*. **IEEE Access**, **9**, 106515–106529. <https://doi.org/10.1109/ACCESS.2021.3100613>
15. Adewumi, A. O., Sawyerr, B. A., & Ali, M. M. (2009). *A heuristic solution to the university timetabling problem*. **Engineering Computations**, **26**(8), 972–984. <https://doi.org/10.1108/02644400910996853>
16. Bashab, A., Ibrahim, A. O., Hashem, I. A. T., Aggarwal, K., Mukhlif, F., Ghaleb, F. A., & Abdelmaboud, A. (2023). *Optimization techniques in university timetabling problem: Constraints, methodologies, benchmarks, and open issues*. **Computers, Materials & Continua**, **74**(3), 6461–6484. <https://doi.org/10.32604/cmc.2023.034051>

17. Ayasi, B., Saleh, M., García-Vico, Á. M., & Carmona, C. (2023). Predicting course enrollment with machine learning and neural networks: A comparative study of algorithms. In Ö. Bilen & E. Shaaban (Eds.), *Studies on social and education sciences 2023* (pp. 157–182). ISTES Organization.
18. Ríos Parra, S. F. (2024). *Predictive analysis of student enrollment in higher education courses using machine learning* [Tesis de maestría, Universidad de los Andes]. Repositorio Institucional Séneca.
19. Pauta Riera, J. L., Leyva Vazquez, M. Y., & Rumbaut Rangel, D. (2025). Pronóstico de la demanda estudiantil mediante modelos de regresión: Un estudio comparativo de la unidad académica de ingeniería, industria y construcción de la Universidad Católica de Cuenca. *Maestro y Sociedad*, 22(4), 3790–3798. <https://maestroysociedad.uo.edu.cu>
20. Ahmed, R. (2023). *Project performance measure and metrics*. En *Research Handbook on Project Performance* (pp. 11–22). Edward Elgar Publishing Limited.
21. Lai, X., Lu, X., & Jin, B. (2026). Data-driven university examination timetabling with consideration of student stress. *Computers & Operations Research*, 192, 107496. <https://doi.org/10.1016/j.cor.2026.107496>
22. Ogunkan, S. K., Idowu, P. O., Omidiora, E. O., & Oyeleye, C. A. (2024). First fit algorithm: A graph coloring approach to conflict-free university course timetabling. *Asian Journal of Research in Computer Science*, 17(5), 125–139. <https://doi.org/10.9734/AJRCOS/2024/v17i5443>

10 Anexos

10.1 Anexo A

En esta sección se incluirá el código en Python empleado para el desarrollo del proyecto general.

Figura 19: Importación de las bases de datos.

```
Importar Datafrom

#Pronostico de demanda (por simplicidad, solo se ocupan los datos hasta el semestre en curso 2026-1 del excel)
a = pd.read_excel(r"C:\Users\rique\Desktop\Datos demanda\Cantidad de Estudiantes por Asignatura (DIREST-061)_20260615_094930.xlsx")

#Complementar pronostico
b = pd.read_excel(r"C:\Users\rique\Desktop\Datos demanda\est_pendientes_2025-2-ANNON.xlsx")

#Cupos paralelos
c = pd.read_excel(r"C:\Users\rique\Desktop\Datos demanda\Inscripción Asignatura por Paralelo y VTR (DIREST-066) - Dashboard - Principal_MC_De

#ramos y requisitos correspondientes a la antigua malla 11 semestres
d = pd.read_excel(r"C:\Users\rique\Desktop\Datos demanda\plan_estudios_usm_malla antigua.xlsx")

#ramos y requisitos correspondientes a la nueva malla 10 semestres
e = pd.read_excel(r"C:\Users\rique\Desktop\Datos demanda\plan_estudios_usm_malla nueva.xlsx")

#Crear copias para trabajar
df1=a.copy()
df2=b.copy()
df3=c.copy()
df4=d.copy()
df5=e.copy()
```

Nota: Elaboración propia.

Figura 20: Preparación de las bases de datos.

```
#Borrar columnas con muchos datos nulos
df1 = df1.loc[:, df1.isnull().mean() < 0.6]

#Solo se trabajara con pregrado y electivos (diurnos)
df1 = df1[~df1['nivel_asignatura'].str.contains('postgrado', case=False, na=False)]
df1 = df1[~df1['jornada_asignatura_cursada'].str.contains('Vespertino', case=False, na=False)]

#eliminar columnas que no sirven
df1.drop(columns=['cod_departamento', 'departamento_asignatura', 'campus_sede_asignatura_cursada'], inplace=True)
print('Filas, Columnas =', df1.shape)
✓ 0.0s

#Borrar columnas con muchos datos nulos
df2 = df2.loc[:, df2.isnull().mean() < 0.6]
print('Filas, Columnas =', df2.shape)
✓ 0.0s

#Borrar columnas con muchos datos nulos
df3 = df3.loc[:, df3.isnull().mean() < 0.6]
df3 = df3[~df3['Nivel Asignatura'].str.contains('postgrado', case=False, na=False)]
df3.drop(columns=['Sede'], inplace=True)
print('Filas, Columnas =', df3.shape)
✓ 0.0s

#Crear columnas utiles para el analisis
df1.columns = df1.columns.str.lower().str.strip()
df2.columns = df2.columns.str.lower().str.strip()
df3.columns = df3.columns.str.lower().str.strip()

df1["tasa_reprobacion %"] = (df1["alumnos_reprobados"] / df1["alumnos_inscritos"])*100
pend = df2.groupby("sigla").size().reset_index(name="pendientes_2025")
df1 = df1.merge(pend, on="sigla", how="left")
df1["pendientes_2025"] = df1["pendientes_2025"].fillna(0)
df1.head()
```

Nota: Elaboración propia.

Figura 21: Creación de la base de datos de la demanda con RF.

```
# Ruta donde están tus archivos Excel
carpeta_archivos = r"C:\Users\rique\Desktop\Datos demanda"

X_total = df1[[
    "año",
    "semestre",
    "nro_paralelos",
    "promedio_cupos",
    "alumnos_reprobados",
    "alumnos_aprobados",
    "pendientes_2025"]]

X_total_scaled = scaler.transform(X_total)

df1["Demanda_Predicha"] = rf_model.predict(X_total_scaled)
df1["Demanda_Predicha"] = np.ceil(df1["Demanda_Predicha"]).astype(int)

demanda_milp = df1[["sigla", "Demanda_Predicha"]].copy()

demanda_milp = demanda_milp.rename(columns={
    "sigla": "Sigla"})

demanda_milp = (
    demanda_milp
    .groupby("Sigla", as_index=False)["Demanda_Predicha"]
    .max())

ruta_salida = os.path.join(
    carpeta_archivos,
    "demanda_predicha_random_forest.xlsx")

demanda_milp.to_excel(ruta_salida, index=False)

print("Archivo creado correctamente en:")
print(ruta_salida)
```

Nota: Elaboración propia.

Figura 22: Creación del grafo de conflictos general del plan 6006.

```
df4.columns = df4.columns.str.lower().str.strip()

# Colores
paleta_semestre = {
    1: "#ff9999", 2: "#ffb266", 3: "#ffff99", 4: "#b3ffb3", 5: "#80ffff",
    6: "#66b3ff", 7: "#b3b3ff", 8: "#cc99ff", 9: "#ff99ff", 10: "#ff8080", 11: "#ff6666"}
G = nx.Graph()
for idx, row in df4.iterrows():
    G.add_node(row['sigla'], nombre=row['asignatura'], semestre=row['semestre'])

# Crear aristas por semestre
ramos_por_semestre = df4.groupby('semestre')['sigla'].apply(list)
for semestre, ramos in ramos_por_semestre.items():
    for ramo_a, ramo_b in itertools.combinations(ramos, 2):
        G.add_edge(ramo_a, ramo_b)

coloreo = nx.coloring.greedy_color(G, strategy='largest_first')
df4['Bloque_Horario'] = df4['sigla'].map(coloreo)
shells = [list(df4[df4['semestre']==s]['sigla']) for s in sorted(df4['semestre'].unique())]
pos = nx.shell_layout(G, nlist=shells)
colores_nodos = [paleta_semestre[df4.loc[df4['sigla']==n, 'semestre'].values[0]] for n in G.nodes()]

plt.figure(figsize=(16,14))
nx.draw_networkx_nodes(G, pos, node_color=colores_nodos, node_size=900, edgecolors='black')
nx.draw_networkx_edges(G, pos, width=1.0, alpha=0.4, edge_color='gray')
nx.draw_networkx_labels(G, pos, labels={n:n for n in G.nodes()}, font_size=7, font_weight='bold')

plt.title("Grafo de Conflictos Completo - USM (Organizado por Semestre)", fontsize=16)
plt.axis('off')
plt.tight_layout()
plt.show()
```

Nota: Elaboración propia.

Figura 23: Creación del grafo de conflictos general del plan 6008.

```
df5.columns = df5.columns.str.lower().str.strip()

paleta_semestre = {
    1: "#ff9999", 2: "#ffb266", 3: "#ffff99", 4: "#b3ffb3", 5: "#80ffff",
    6: "#66b3ff", 7: "#b3b3ff", 8: "#cc99ff", 9: "#ff99ff", 10: "#ff8080", 11: "#ff6666"}
G = nx.Graph()
for idx, row in df5.iterrows():
    G.add_node(row['sigla'], nombre=row['asignatura'], semestre=row['semestre'])

ramos_por_semestre = df5.groupby('semestre')['sigla'].apply(list)
for semestre, ramos in ramos_por_semestre.items():
    for ramo_a, ramo_b in itertools.combinations(ramos, 2):
        G.add_edge(ramo_a, ramo_b)

coloreo = nx.coloring.greedy_color(G, strategy='largest_first')
df5['Bloque Horario'] = df5['sigla'].map(coloreo)
shells = [list(df5[df5['semestre']==s]['sigla']) for s in sorted(df5['semestre'].unique())]
pos = nx.shell_layout(G, nlist=shells)
colores_nodos = [paleta_semestre[df5.loc[df5['sigla']==n, 'semestre'].values[0]] for n in G.nodes()]

plt.figure(figsize=(16,14))
nx.draw_networkx_nodes(G, pos, node_color=colores_nodos, node_size=900, edgecolors='black')
nx.draw_networkx_edges(G, pos, width=1.0, alpha=0.4, edge_color='gray')
nx.draw_networkx_labels(G, pos, labels={n:n for n in G.nodes()}, font_size=7, font_weight='bold')

plt.title("Grafo de Conflictos Completo - USM (Organizado por Semestre)", fontsize=16)
plt.axis('off')
plt.tight_layout()
plt.show()
```

Nota: Elaboración propia.

Figura 24: Creación del grafo de conflictos segmentado por semestre del plan 6006 y 6008.

```
def grafo_conflictos_por_semestre(df, paleta_semestre, titulo_general="Mini-Grafos de Conflictos por Semestre - USM"):
    df.columns = df.columns.str.lower().str.strip()
    todos_semestres = sorted(df['semestre'].unique())
    n_semestres = len(todos_semestres)
    fig = plt.figure(figsize=(24, 5*(n_semestres+1)//2))
    gs = gridspec.GridSpec((n_semestres+1)//2, 2, figure=fig)
    axes = []
    for i in range(n_semestres):
        row = i // 2
        col = i % 2
        axes.append(fig.add_subplot(gs[row, col]))
    for s, ax in zip(todos_semestres, axes):
        df_sem = df[df['semestre']==s]
        G_sem = nx.Graph()
        for idx, row in df_sem.iterrows():
            G_sem.add_node(row['sigla'], nombre=row.get('asignatura',''))
        ramos = df_sem['sigla'].tolist()
        for ramo_a, ramo_b in itertools.combinations(ramos, 2):
            G_sem.add_edge(ramo_a, ramo_b)

        pos = nx.circular_layout(G_sem)
        colores_nodos = [paleta_semestre[s] for _ in G_sem.nodes()]

        nx.draw_networkx_nodes(G_sem, pos, node_color=colores_nodos, node_size=900, edgecolors='black', ax=ax)
        nx.draw_networkx_edges(G_sem, pos, width=1.0, alpha=0.4, edge_color='gray', ax=ax)
        nx.draw_networkx_labels(G_sem, pos, labels={n:n for n in G_sem.nodes()}, font_size=7, font_weight='bold', ax=ax)
        ax.set_title(f"Semestre {s} - {len(ramos)} ramos", fontsize=12)
        ax.axis('off')

    handles = [plt.Line2D([0],[0], marker='o', color='w', label=f'Semestre {s}',
                          markerfacecolor=paleta_semestre[s], markersize=15) for s in todos_semestres]
    fig.legend(handles=handles, loc='lower center', ncol=4, title='Leyenda de Semestres', fontsize=10)
    plt.suptitle(titulo_general, fontsize=16)
    plt.tight_layout(rect=[0,0,1,0.97])
    plt.show()
```

Nota: Elaboración propia.

10.2 Anexo B

En este anexo se presenta el código desarrollado en Python para la implementación del modelo de optimización, incluyendo la preparación de datos, generación de patrones horarios, formulación del modelo MILP, resolución mediante PuLP/CBC, exportación de resultados y cálculo de métricas de desempeño.

```
ARCHIVO_HISTORICO_ESTUDIANTES = a.copy()
ARCHIVO_PENDIENTES = b.copy()
ARCHIVO_INSCRIPCIONES = c.copy()
ARCHIVO_MALLA_ANTIGUA = d.copy()
ARCHIVO_MALLA_NUEVA = e.copy()

CARPETA_DATOS = r"C:\Users\rique\Desktop\Datos demanda"
ARCHIVO_DEMANDA_RF = os.path.join(
    CARPETA_DATOS,
    "demanda_predicha_random_forest.xlsx")
ARCHIVO_SALIDA = os.path.join(
    CARPETA_DATOS,
    "resultado_horarios_modelo.xlsx")
ARCHIVO_MODELO_LP = os.path.join(
    CARPETA_DATOS,
    "modelo_horarios.lp")
CAP_MIN = 25
CAP_MAX = 45
DIAS = ["Lunes", "Martes", "Miercoles", "Jueves", "Viernes"]
BLOQUES = list(range(1, 17))
BLOQUES_PROTEGIDOS = {("Miercoles", 7), ("Miercoles", 8)}
BLOQUES_TARDIOS = {15, 16}
MAX_PATRONES_POR_RAMO = 250
ALPHA = 1.00
BETA = 0.08
GAMMA = 15.00
DELTA = 0.2
TIEMPO_MAX_SOLVER = 300
VERBOSE = True

def normalizar_texto_columna(texto):

    texto = str(texto).lower().strip()

    reemplazos = {
        "á": "a",
        "é": "e",
```

```

        "í": "i",
        "ó": "o",
        "ú": "u",
        "ñ": "n"
    }

    for original, nuevo in reemplazos.items():
        texto = texto.replace(original, nuevo)

    texto = re.sub(r"^[a-z0-9]+", "", texto)

    return texto

def normalizar_sigla(valor):

    if pd.isna(valor):
        return ""

    texto = str(valor).upper().strip()
    texto = re.sub(r"\s+", "", texto)

    return texto

def nombre_seguro(texto):

    texto = str(texto)

    reemplazos = {
        "á": "a",
        "é": "e",
        "í": "i",
        "ó": "o",
        "ú": "u",
        "Á": "A",
        "É": "E",
        "Í": "I",
        "Ó": "O",
        "Ú": "U",
        "ñ": "n",
        "Ñ": "N"
    }

    for original, nuevo in reemplazos.items():
        texto = texto.replace(original, nuevo)

    return re.sub(
        r"^[A-Za-z0-9_]+",
        "_",

```

```

        texto
    )

def buscar_columna(df, posibles_nombres):

    mapa_columnas = {
        normalizar_texto_columna(col): col
        for col in df.columns
    }

    for nombre in posibles_nombres:

        clave = normalizar_texto_columna(nombre)

        if clave in mapa_columnas:
            return mapa_columnas[clave]

    return None

def verificar_archivo(origen, nombre):

    if origen is None:
        return

    if isinstance(origen, pd.DataFrame):
        return

    if isinstance(origen, (str, os.PathLike)):

        if not os.path.exists(origen):
            raise FileNotFoundError(
                f'No se encontró el archivo {nombre}: {origen}'
            )

        return

    raise TypeError(
        f'La base '{nombre}' debe ser un DataFrame '
        f'or una ruta de archivo Excel.'
    )

def leer_base(origen, nombre):

    verificar_archivo(
        origen,
        nombre
    )

```

```

if isinstance(origen, pd.DataFrame):
    return origen.copy()

return pd.read_excel(origen)

def techo_division(a, b):

    return int(
        math.ceil(
            float(a) / float(b)
        )
    )

def cargar_bases():

    bases = {}

    bases["malla_antigua"] = leer_base(
        ARCHIVO_MALLA_ANTIGUA,
        "malla antigua"
    )

    bases["malla_nueva"] = leer_base(
        ARCHIVO_MALLA_NUEVA,
        "malla nueva"
    )

    bases["inscripciones_actuales"] = leer_base(
        ARCHIVO_INSCRIPCIONES,
        "inscripciones actuales"
    )

    bases["historico_estudiantes"] = leer_base(
        ARCHIVO_HISTORICO_ESTUDIANTES,
        "histórico de estudiantes"
    )

    bases["pendientes"] = leer_base(
        ARCHIVO_PENDIENTES,
        "pendientes"
    )

    bases["demanda_rf"] = leer_base(
        ARCHIVO_DEMANDA_RF,
        "demanda Random Forest"
    )

    return bases

```

```

def preparar_malla(df, nombre_malla):

    col_codigo = buscar_columna(
        df,
        ["Codigo", "Código", "cod_plan", "plan"]
    )

    col_sigla = buscar_columna(
        df,
        ["Sigla", "cod_asign", "asignatura_codigo"]
    )

    col_nombre = buscar_columna(
        df,
        ["Asignatura", "nombre", "nombre_asignatura"]
    )

    col_semestre = buscar_columna(
        df,
        ["Semestre", "nivel"]
    )

    col_bloques = buscar_columna(
        df,
        ["Bloques", "Bloque", "horas", "carga"]
    )

    faltantes = []

    if col_codigo is None:
        faltantes.append("Codigo")

    if col_sigla is None:
        faltantes.append("Sigla")

    if col_nombre is None:
        faltantes.append("Asignatura")

    if col_semestre is None:
        faltantes.append("Semestre")

    if col_bloques is None:
        faltantes.append("Bloques")

    if faltantes:
        raise ValueError(
            f'Faltan columnas en {nombre_malla}: {faltantes}'

```

```

    )

salida = pd.DataFrame({
    "malla": df[col_codigo],
    "sigla_original": df[col_sigla],
    "sigla": df[col_sigla].apply(
        normalizar_sigla
    ),
    "nombre_ramo": df[col_nombre],
    "semestre": pd.to_numeric(
        df[col_semestre],
        errors="coerce"
    ),
    "bloques": pd.to_numeric(
        df[col_bloques],
        errors="coerce"
    ).fillna(0).astype(int)
})

salida = salida.dropna(
    subset=["semestre"]
)

salida["semestre"] = salida[
    "semestre"
].astype(int)

salida = salida[
    salida["sigla"] != ""
].copy()

return salida

def preparar_demanda_rf(df_rf):

    col_sigla = buscar_columna(
        df_rf,
        [
            "sigla",
            "cod_asign",
            "asignatura"
        ]
    )

    col_demanda = buscar_columna(
        df_rf,
        [
            "demanda_predicha",

```

```

        "demanda",
        "prediccion",
        "predicción",
        "y_pred",
        "alumnos_predichos"
    ]
)

if col_sigla is None or col_demanda is None:

    raise ValueError(
        "No se encontraron las columnas de sigla "
        "y demanda predicha en el archivo Random Forest."
    )

demanda = df_rf[[
    col_sigla,
    col_demanda
]].copy()

demanda.columns = [
    "sigla",
    "demanda_estimada"
]

demanda["sigla"] = demanda[
    "sigla"
].apply(
    normalizar_sigla
)

demanda["demanda_estimada"] = pd.to_numeric(
    demanda["demanda_estimada"],
    errors="coerce"
)

demanda = demanda.dropna(
    subset=["demanda_estimada"]
)

demanda = demanda[
    demanda["sigla"] != ""
].copy()

demanda = (
    demanda
    .groupby(
        "sigla",

```

```

        as_index=False
    )["demanda_estimada"]
    .max()
)

return demanda

def construir_conjuntos_modelo(bases):

    malla_6006 = preparar_malla(
        bases["malla_antigua"],
        "malla antigua 6006"
    )

    malla_6008 = preparar_malla(
        bases["malla_nueva"],
        "malla nueva 6008"
    )

    mallas_completas = pd.concat(
        [
            malla_6006,
            malla_6008
        ],
        ignore_index=True
    )

    mallas_completas["malla"] = pd.to_numeric(
        mallas_completas["malla"],
        errors="coerce"
    ).astype("Int64")

    mallas_calendarizables = mallas_completas[
        mallas_completas["bloques"] > 0
    ].copy()

    demanda_rf = preparar_demanda_rf(
        bases["demanda_rf"]
    )

    siglas_malla = set(
        mallas_calendarizables[
            "sigla"
        ].unique()
    )

    siglas_rf = set(
        demanda_rf[

```

```

        "sigla"
    ].unique()
)

siglas_modelo = (
    siglas_malla
    .intersection(siglas_rf)
)

mallas_modelo = mallas_calendarizables[
    mallas_calendarizables[
        "sigla"
    ].isin(siglas_modelo)
].copy()

mallas_modelo = mallas_modelo.merge(
    demanda_rf,
    on="sigla",
    how="left"
)

ramos = (
    mallas_modelo
    .groupby(
        "sigla",
        as_index=False
    )
    .agg(
        nombre_ramo=(
            "nombre_ramo",
            "first"
        ),
        bloques=(
            "bloques",
            "max"
        ),
        demanda_estimada=(
            "demanda_estimada",
            "max"
        ),
        mallas_asociadas=(
            "malla",
            lambda x: ", ".join(
                map(
                    str,
                    sorted(
                        set(
                            x.dropna().astype(int)

```

```

    )
    )
    )
    )
    )
    )
    )
    )

ramos_por_malla_semestre = {}

for (
    malla,
    semestre
), grupo in mallas_modelo.groupby(
    [
        "malla",
        "semestre"
    ]
):

    if pd.isna(malla):
        continue

    clave = (
        int(malla),
        int(semestre)
    )

    ramos_por_malla_semestre[
        clave
    ] = sorted(
        grupo[
            "sigla"
        ].dropna().unique().tolist()
    )

rf_sin_malla = demanda_rf[
    ~demanda_rf[
        "sigla"
    ].isin(siglas_malla)
].copy()

malla_sin_rf = mallas_calendarizables[
    ~mallas_calendarizables[
        "sigla"
    ].isin(siglas_rf)
].copy()

```

```

resumen_cruce = pd.DataFrame([
    {
        "Indicador": "Ramos distintos en las mallas",
        "Cantidad": len(siglas_malla)
    },
    {
        "Indicador": "Ramos distintos con demanda RF",
        "Cantidad": len(siglas_rf)
    },
    {
        "Indicador": "Ramos incluidos en R",
        "Cantidad": len(siglas_modelo)
    },
    {
        "Indicador": "Ramos RF sin coincidencia en malla",
        "Cantidad": len(
            siglas_rf - siglas_malla
        )
    },
    {
        "Indicador": "Ramos de malla sin demanda RF",
        "Cantidad": len(
            siglas_malla - siglas_rf
        )
    }
])

diagnosticos = {
    "resumen_cruce": resumen_cruce,
    "mallas_modelo": mallas_modelo,
    "rf_sin_malla": rf_sin_malla,
    "malla_sin_rf": malla_sin_rf
}

return (
    ramos,
    ramos_por_malla_semestre,
    mallas_completas,
    diagnosticos
)

def preparar_demanda_y_paralelos(ramos):

    ramos_dem = ramos.copy()

    ramos_dem[
        "demanda_estimada"
    ] = pd.to_numeric(

```

```

    ramos_dem[
        "demanda_estimada"
    ],
    errors="coerce"
)

ramos_dem = ramos_dem.dropna(
    subset=["demanda_estimada"]
)

ramos_dem[
    "demanda_estimada"
] = (
    ramos_dem[
        "demanda_estimada"
    ]
    .round()
    .astype(int)
)

ramos_dem = ramos_dem[
    ramos_dem[
        "demanda_estimada"
    ] > 0
].copy()

ramos_dem[
    "n_paralelos"
] = ramos_dem[
    "demanda_estimada"
].apply(
    lambda demanda: max(
        1,
        techo_division(
            demanda,
            CAP_MAX
        )
    )
)

ramos_dem[
    "capacidad_sugerida"
] = ramos_dem.apply(
    lambda fila: max(
        CAP_MIN,
        min(
            CAP_MAX,
            techo_division(

```

```

        fila[
            "demanda_estimada"
        ],
        fila[
            "n_paralelos"
        ]
    )
)
),
axis=1
)

ramos_dem[
    "fuente_demanda"
] = "Random Forest"

return ramos_dem

def expandir_ramo_paralelo(ramos_dem):

    filas = []

    for _, fila in ramos_dem.iterrows():

        for paralelo in range(
            1,
            int(
                fila["n_paralelos"]
            ) + 1
        ):

            capacidad = int(
                fila[
                    "capacidad_sugerida"
                ]
            )

            filas.append({
                "sigla": fila["sigla"],
                "nombre_ramo": fila[
                    "nombre_ramo"
                ],
                "paralelo": paralelo,
                "bloques": int(
                    fila["bloques"]
                ),
                "demanda_estimada": int(
                    fila["demanda_estimada"]
                )
            })

```

```

    ),
    "n_paralelos": int(
        fila["n_paralelos"]
    ),
    "capacidad_sugerida": capacidad,
    "capacidad_paralelo": capacidad,
    "mallas_asociadas": fila[
        "mallas_asociadas"
    ],
    "fuente_demanda": fila[
        "fuente_demanda"
    ]
}
})

return pd.DataFrame(filas)

def intervalo_valido(dia, bloques, bloques_protegidos):

    for bloque in bloques:

        if bloque not in BLOQUES:
            return False

        if (dia, bloque) in bloques_protegidos:
            return False

    return True

def generar_sesiones_por_dia(bloques_protegidos):

    sesiones = {
        2: [],
        4: []
    }

    for dia in DIAS:

        for inicio in range(1, 17, 2):

            bloques = tuple(
                range(inicio, inicio + 2)
            )

            if max(bloques) <= 16:

                if intervalo_valido(
                    dia,
                    bloques,

```

```

        bloques_protegidos
    ):
        sesiones[2].append(
            (dia, bloques)
        )

for inicio in range(1, 14, 2):

    bloques = tuple(
        range(inicio, inicio + 4)
    )

    if max(bloques) <= 16:

        if intervalo_valido(
            dia,
            bloques,
            bloques_protegidos
        ):
            sesiones[4].append(
                (dia, bloques)
            )

return sesiones

def particiones_bloques(total_bloques):

    if total_bloques % 2 != 0:

        raise ValueError(
            f"La carga horaria {total_bloques} no es par."
        )

    resultados = []

    max_cuatro = total_bloques // 4

    for n4 in range(max_cuatro + 1):

        resto = total_bloques - 4 * n4

        if resto % 2 == 0:

            n2 = resto // 2

            particion = (
                [4] * n4
                + [2] * n2

```

```

    )

    resultados.append(
        sorted(
            particion,
            reverse=True
        )
    )

resultados = sorted(
    resultados,
    key=lambda x: (
        sum(
            1
            for valor in x
            if valor == 4
        ),
        len(x)
    )
)

return resultados

def sesiones_no_se_topan(sesiones):

    usados = set()

    for dia, bloques in sesiones:

        for bloque in bloques:

            slot = (
                dia,
                bloque
            )

            if slot in usados:
                return False

            usados.add(slot)

    return True

def describir_sesiones(sesiones):

    partes = []

    for dia, bloques in sesiones:

```

```

    partes.append(
        f"{dia} "
        f"{min(bloques)}-"
        f"{max(bloques)}"
    )

    return " / ".join(partes)

def calcular_penalizacion_patron(sesiones):

    penalizacion = 0.0
    dias_usados = set()

    for dia, bloques in sesiones:

        dias_usados.add(dia)

        for bloque in bloques:

            if bloque in BLOQUES_TARDIOS:
                penalizacion += 8

            if bloque in {13, 14}:
                penalizacion += 2

        if len(bloques) == 4:
            penalizacion += 4

        if (
            dia == "Viernes"
            and any(
                bloque >= 11
                for bloque in bloques
            )
        ):
            penalizacion += 3

    total_bloques = sum(
        len(bloques)
        for _, bloques in sesiones
    )

    sesiones_esperadas = max(
        1,
        total_bloques // 2
    )

```

```

    penalizacion += max(
        0,
        sesiones_esperadas
        - len(dias_usados)
    ) * 5

    return penalizacion

def seleccionar_distribuidos(lista, cantidad):

    if cantidad >= len(lista):
        return lista

    if cantidad == 1:
        return [lista[len(lista) // 2]]

    indices = [
        round(
            i * (len(lista) - 1)
            / (cantidad - 1)
        )
        for i in range(cantidad)
    ]

    indices = list(
        dict.fromkeys(indices)
    )

    if len(indices) < cantidad:

        for i in range(len(lista)):

            if i not in indices:
                indices.append(i)

            if len(indices) == cantidad:
                break

    indices = sorted(
        indices[:cantidad]
    )

    return [
        lista[i]
        for i in indices
    ]

def seleccionar_patrones(patrones):

```

```

patrones = sorted(
    patrones,
    key=lambda patron: (
        patron["penalizacion"],
        tuple(
            (
                DIAS.index(dia),
                bloque
            )
            for dia, bloque
            in patron["slots"]
        )
    )
)

if len(patrones) <= MAX_PATRONES_POR_RAMO:
    return patrones

grupos = {}

for patron in patrones:

    penalizacion = patron[
        "penalizacion"
    ]

    if penalizacion not in grupos:
        grupos[penalizacion] = []

    grupos[penalizacion].append(
        patron
    )

seleccionados = []

for penalizacion in sorted(grupos):

    grupo = grupos[
        penalizacion
    ]

    faltan = (
        MAX_PATRONES_POR_RAMO
        - len(seleccionados)
    )

    if faltan <= 0:

```

```

        break

    if len(grupo) <= faltan:

        seleccionados.extend(
            grupo
        )

    else:

        seleccionados.extend(
            seleccionar_distribuidos(
                grupo,
                faltan
            )
        )

        break

    return seleccionados

def generar_patrones_para_bloques(
    total_bloques,
    bloques_protegidos
):

    sesiones_por_largo = generar_sesiones_por_dia(
        bloques_protegidos
    )

    patrones = []

    for particion in particiones_bloques(
        total_bloques
    ):

        listas_sesiones = [
            sesiones_por_largo[largo]
            for largo in particion
        ]

        for combo in product(
            *listas_sesiones
        ):

            sesiones = list(combo)

            dias = [

```

```

        dia
        for dia, _ in sesiones
    ]

    if len(dias) != len(set(dias)):
        continue

    if not sesiones_no_se_topan(
        sesiones
    ):
        continue

    slots = sorted(
        [
            (dia, bloque)
            for dia, bloques in sesiones
            for bloque in bloques
        ],
        key=lambda x: (
            DIAS.index(x[0]),
            x[1]
        )
    )

    patrones.append({
        "sesiones": sesiones,
        "slots": slots,
        "penalizacion": calcular_penalizacion_patron(
            sesiones
        ),
        "descripcion": describir_sesiones(
            sesiones
        )
    })

patrones_unicos = {}

for patron in patrones:

    clave = tuple(
        patron["slots"]
    )

    if clave not in patrones_unicos:

        patrones_unicos[
            clave
        ] = patron

```

```

        elif (
            patron["penalizacion"]
            < patrones_unicos[
                clave
            ]["penalizacion"]
        ):

            patrones_unicos[
                clave
            ] = patron

    patrones = list(
        patrones_unicos.values()
    )

    patrones = seleccionar_patrones(
        patrones
    )

    for numero, patron in enumerate(
        patrones,
        start=1
    ):

        patron[
            "patron_id"
        ] = numero

    if not patrones:

        raise ValueError(
            f"No se generaron patrones factibles "
            f"para {total_bloques} bloques."
        )

    return patrones

def construir_patrones(ramos_dem):

    bloques_protegidos = set(
        BLOQUES_PROTEGIDOS
    )

    patrones_por_bloques = {}
    patrones_por_ramo = {}

    for bloques in sorted(

```

```

    ramos_dem["bloques"].unique()
):

    bloques = int(bloques)

    patrones_por_bloques[
        bloques
    ] = generar_patrones_para_bloques(
        bloques,
        bloques_protegidos
    )

for _, fila in ramos_dem.iterrows():

    patrones_por_amo[
        fila["sigla"]
    ] = patrones_por_bloques[
        int(fila["bloques"])
    ]

return patrones_por_amo

def resolver_modelo(
    ramos_dem,
    ramos_paralelos,
    ramos_por_malla_semestre,
    patrones_por_amo
):

    modelo = pulp.LpProblem(
        "Asignacion_Horarios_Academicos",
        pulp.LpMinimize
    )

    R = ramos_dem["sigla"].tolist()

    K = {
        fila["sigla"]: list(
            range(
                1,
                int(fila["n_paralelos"]) + 1
            )
        )
        for _, fila in ramos_dem.iterrows()
    }

    Dem = dict(
        zip(

```

```

        ramos_dem["sigla"],
        ramos_dem["demanda_estimada"]
    )
)

H = dict(
    zip(
        ramos_dem["sigla"],
        ramos_dem["bloques"]
    )
)

Cap = {
    (
        fila["sigla"],
        int(fila["paralelo"])
    ): int(fila["capacidad_paralelo"])
    for _, fila in ramos_paralelos.iterrows()
}

X = {}

for r in R:
    for k in K[r]:
        for patron in patrones_por_ramo[r]:

            patron_id = patron["patron_id"]

            X[
                (r, k, patron_id)
            ] = pulp.LpVariable(
                f"X_{nombre_seguro(r)}_{k}_{patron_id}",
                cat="Binary"
            )

def Y(r, k, d, b):

    return pulp.lpSum(
        X[
            (
                r,
                k,
                patron["patron_id"]
            )
        ]
        for patron in patrones_por_ramo[r]
        if (d, b) in patron["slots"]
    )

```

```

for r in R:
    for k in K[r]:

        modelo += (
            pulp.lpSum(
                X[
                    (
                        r,
                        k,
                        patron["patron_id"]
                    )
                ]
                for patron in patrones_por_ramo[r]
            ) == 1,
            f"Asignacion_unica_{nombre_seguro(r)}_{k}"
        )

for (
    malla,
    semestre
), lista_ramos in ramos_por_malla_semestre.items():

    lista_ramos = [
        r
        for r in lista_ramos
        if r in R
    ]

    for r, q in combinations(
        lista_ramos,
        2
    ):

        for k in K[r]:
            for l in K[q]:
                for d in DIAS:
                    for b in BLOQUES:

                        modelo += (
                            Y(r, k, d, b)
                            + Y(q, l, d, b)
                            <= 1,
                            (
                                f"No_tope_"
                                f"m{malla}_"
                                f"s{semestre}_"
                                f"{nombre_seguro(r)}_"

```

```

        f"{k}_"
        f"{nombre_seguro(q)}_"
        f"{l}_"
        f"{nombre_seguro(d)}_"
        f"{b}"
    )
)

ZP = {}

for r in R:

    if len(K[r]) <= 1:
        continue

    for k, l in combinations(
        K[r],
        2
    ):

        for d in DIAS:
            for b in BLOQUES:

                ZP[
                    (
                        r,
                        k,
                        l,
                        d,
                        b
                    )
                ] = pulp.LpVariable(
                    (
                        f"ZP_{nombre_seguro(r)}_"
                        f"{k}_{l}_"
                        f"{nombre_seguro(d)}_"
                        f"{b}"
                    ),
                    cat="Binary"
                )

                modelo += (
                    ZP[
                        (
                            r,
                            k,
                            l,
                            d,

```

```

        b
    )
]
>= (
    Y(r, k, d, b)
    + Y(r, l, d, b)
    - 1
),
(
    f"Coincidencia_"
    f"{nombre_seguro(r)}_"
    f"{k}_{l}"
    f"{nombre_seguro(d)}_"
    f"{b}"
)
)

L = pulp.LpVariable(
    "Carga_Maxima_Global",
    lowBound=0,
    cat="Continuous"
)

for d in DIAS:
    for b in BLOQUES:

        modelo += (
            pulp.lpSum(
                Y(r, k, d, b)
                for r in R
                for k in K[r]
            )
            <= L,
            (
                f"Balance_global_"
                f"{nombre_seguro(d)}_"
                f"{b}"
            )
        )

costo_patrones = pulp.lpSum(
    ALPHA
    * patron["penalizacion"]
    * X[
        (
            r,
            k,
            patron["patron_id"]

```

```

    )
]
for r in R
for k in K[r]
for patron in patrones_por_rama[r]
)

costo_tarde = pulp.lpSum(
    BETA
    * Cap[(r, k)]
    * sum(
        1
        for _, bloque in patron["slots"]
        if bloque in BLOQUES_TARDIOS
    )
    * X[
        (
            r,
            k,
            patron["patron_id"]
        )
    ]
for r in R
for k in K[r]
for patron in patrones_por_rama[r]
)

costo_coincidencia = pulp.lpSum(
    GAMMA * variable
    for variable in ZP.values()
)

costo_balance = DELTA * L

modelo += (
    costo_patrones
    + costo_tarde
    + costo_coincidencia
    + costo_balance
)

os.makedirs(
    CARPETA_DATOS,
    exist_ok=True
)

modelo.writeLP(
    ARCHIVO_MODELO_LP

```

```

)

archivo_log = os.path.join(
    CARPETA_DATOS,
    "cbc_solver.log"
)

solver = pulp.PULP_CBC_CMD(
    msg=VERBOSE,
    timeLimit=TIEMPO_MAX_SOLVER,
    logPath=archivo_log
)

codigo_estado = modelo.solve(solver)
estado = pulp.LpStatus[codigo_estado]
estado_reporte = estado
if os.path.exists(archivo_log):

    with open(
        archivo_log,
        "r",
        encoding="utf-8",
        errors="ignore"
    ) as archivo:

        log_cbc = archivo.read()

        if "Result - Stopped on time limit" in log_cbc:
            estado_reporte = "Factible - límite de tiempo"

        elif "Result - Optimal solution found" in log_cbc:
            estado_reporte = "Optimal"

print()
print("Costo patrones:", pulp.value(costo_patrones))
print("Costo tarde:", pulp.value(costo_tarde))
print("Costo coincidencia:", pulp.value(costo_coincidencia))
print("Costo balance:", pulp.value(costo_balance))
print("L:", pulp.value(L))
print()

return {
    "modelo": modelo,
    "estado": estado,
    "objetivo": (
        pulp.value(modelo.objective)
        if estado == "Optimal"
        else None
    ),
    "X": X,

```

```

    "ZP": ZP,
    "L": (
        pulp.value(L)
        if estado == "Optimal"
        else None
    ),
    "R": R,
    "K": K,
    "Dem": Dem,
    "H": H,
    "Cap": Cap,
    "Y": Y,
    "patrones_por_ramo": patrones_por_ramo
}
def obtener_patron_elegido(resultado, ramo, paralelo):

    if resultado["estado"] != "Optimal":
        return None

    for patron in resultado["patrones_por_ramo"][ramo]:

        valor = pulp.value(
            resultado["X"][
                (ramo, paralelo, patron["patron_id"])
            ]
        )

        if valor is not None and valor > 0.5:
            return patron

    return None

def validar_solucion(
    resultado,
    ramos_paralelos,
    ramos_por_malla_semestre
):

    errores = []
    ocupaciones = {}

    for _, fila in ramos_paralelos.iterrows():

        ramo = fila["sigla"]
        paralelo = int(fila["paralelo"])

        patron = obtener_patron_elegido(
            resultado,

```

```

        ramo,
        paralelo
    )

    if patron is None:
        errores.append({
            "Tipo": "Patrón faltante",
            "Detalle": f"{ramo}-P {paralelo}"
        })

    else:
        ocupaciones[(ramo, paralelo)] = set(
            patron["slots"]
        )

for (malla, semestre), lista_ramos in ramos_por_malla_semestre.items():

    lista_ramos = [
        r for r in lista_ramos
        if r in resultado["R"]
    ]

    for r, q in combinations(lista_ramos, 2):

        for k in resultado["K"][r]:
            for l in resultado["K"][q]:

                coincidencias = ocupaciones.get(
                    (r, k),
                    set()
                ).intersection(
                    ocupaciones.get(
                        (q, l),
                        set()
                    )
                )

                for dia, bloque in coincidencias:

                    errores.append({
                        "Tipo": "Tope curricular",
                        "Detalle": (
                            f"Malla {malla}, semestre {semestre}: "
                            f"{r}-P {k} y {q}-P {l}, "
                            f"{dia}, bloque {bloque}"
                        )
                    })

```

```

if not errores:

    errores.append({
        "Tipo": "Validación",
        "Detalle": (
            "Todos los paralelos tienen patrón "
            "y no existen topes curriculares."
        )
    })

return pd.DataFrame(errores)

def ajustar_hojas_excel(writer):

    for nombre_hoja, hoja in writer.sheets.items():

        hoja.freeze_panes = "A2"

        for columna in hoja.columns:

            letra = columna[0].column_letter

            maximo = max(
                (
                    len(str(celda.value))
                    for celda in columna
                    if celda.value is not None
                ),
                default=0
            )

            hoja.column_dimensions[letra].width = min(
                max(maximo + 2, 12),
                50
            )

def exportar_resultados(
    resultado,
    ramos_dem,
    ramos_paralelos,
    ramos_por_malla_semestre,
    mallas_completas,
    diagnosticos
):

    if resultado["estado"] != "Optimal":

        raise RuntimeError(

```

```

        "No se exportará porque el modelo terminó "
        f'con estado '{resultado['estado']}'.'"
    )

filas_patrones = []
filas_detalle = []

for _, fila in ramos_paralelos.iterrows():

    ramo = fila["sigla"]
    paralelo = int(fila["paralelo"])

    patron = obtener_patron_elegido(
        resultado,
        ramo,
        paralelo
    )

    if patron is None:

        raise RuntimeError(
            f'No se encontró patrón para "
            f'{ramo}-P{paralelo}.'"
        )

    filas_patrones.append({
        "Sigla": ramo,
        "Asignatura": fila["nombre_ramo"],
        "Paralelo": paralelo,
        "Mallas asociadas": fila["mallas_asociadas"],
        "Bloques semanales": int(fila["bloques"]),
        "Demanda estimada": int(fila["demanda_estimada"]),
        "N paralelos": int(fila["n_paralelos"]),
        "Capacidad paralelo": int(fila["capacidad_paralelo"]),
        "Patrón elegido": int(patron["patron_id"]),
        "Horario resumido": patron["descripcion"],
        "Penalización patrón": patron["penalizacion"]
    })

    for dia, bloques in patron["sesiones"]:

        filas_detalle.append({
            "Sigla": ramo,
            "Asignatura": fila["nombre_ramo"],
            "Paralelo": paralelo,
            "Día": dia,
            "Bloque inicio": min(bloques),
            "Bloque término": max(bloques),

```

```

        "Bloques": f"{min(bloques)}-{max(bloques)}",
        "Demanda estimada": int(fila["demanda_estimada"]),
        "Capacidad paralelo": int(fila["capacidad_paralelo"])
    })

df_patrones = pd.DataFrame(filas_patrones)
df_detalle = pd.DataFrame(filas_detalle)

matriz = []

for bloque in BLOQUES:

    fila = {
        "Bloque": bloque
    }

    for dia in DIAS:

        asignados = [
            f"{clase['Sigla']}-P{clase['Paralelo']}"
            for _, clase in df_detalle.iterrows()
            if (
                clase["Día"] == dia
                and clase["Bloque inicio"] <= bloque
                <= clase["Bloque término"]
            )
        ]

        fila[dia] = " / ".join(asignados)

    matriz.append(fila)

df_matriz = pd.DataFrame(matriz)

df_paralelos = ramos_dem[[
    "sigla",
    "nombre_ramo",
    "bloques",
    "demanda_estimada",
    "n_paralelos",
    "capacidad_sugerida",
    "mallas_asociadas",
    "fuente_demanda"
]].copy()

df_paralelos.columns = [
    "Sigla",
    "Asignatura",

```

```

        "Bloques semanales",
        "Demanda estimada",
        "N paralelos",
        "Capacidad por paralelo",
        "Mallas asociadas",
        "Fuente demanda"
    ]

    df_paralelos["Capacidad total ofrecida"] = (
        df_paralelos["N paralelos"]
        * df_paralelos["Capacidad por paralelo"]
    )

    df_restricciones = pd.DataFrame([
        {
            "Malla": malla,
            "Semestre": semestre,
            "Ramos incluidos en R_m_s": ", ".join(lista_ramos)
        }
        for (malla, semestre), lista_ramos
        in ramos_por_malla_semestre.items()
    ])

    df_validacion = validar_solucion(
        resultado,
        ramos_paralelos,
        ramos_por_malla_semestre
    )

    df_resumen = pd.DataFrame([
        {
            "Indicador": "Estado del modelo",
            "Valor": resultado["estado_reporte"]
        },
        {
            "Indicador": "Valor función objetivo",
            "Valor": resultado["objetivo"]
        },
        {
            "Indicador": "Carga máxima global L",
            "Valor": resultado["L"]
        },
        {
            "Indicador": "Cantidad de ramos en R",
            "Valor": len(ramos_dem)
        },
        {
            "Indicador": "Cantidad de ramo-paralelos",

```

```

        "Valor": len(ramos_paralelos)
    },
    {
        "Indicador": "Cap mínimo",
        "Valor": CAP_MIN
    },
    {
        "Indicador": "Cap máximo",
        "Valor": CAP_MAX
    },
    {
        "Indicador": "Bloques protegidos",
        "Valor": str(sorted(BLOQUES_PROTEGIDOS))
    },
    {
        "Indicador": "Bloques tardíos",
        "Valor": str(sorted(BLOQUES_TARDIOS))
    },
    {
        "Indicador": "Máximo de patrones por ramo",
        "Valor": MAX_PATRONES_POR_RAMO
    },
    {
        "Indicador": "alpha",
        "Valor": ALPHA
    },
    {
        "Indicador": "beta",
        "Valor": BETA
    },
    {
        "Indicador": "gamma",
        "Valor": GAMMA
    },
    {
        "Indicador": "delta",
        "Valor": DELTA
    }
]

os.makedirs(
    CARPETA_DATOS,
    exist_ok=True
)

with pd.ExcelWriter(
    ARCHIVO_SALIDA,
    engine="openpyxl"

```

) as writer:

```
df_resumen.to_excel(
    writer,
    sheet_name="Resumen",
    index=False
)

diagnosticos["resumen_cruce"].to_excel(
    writer,
    sheet_name="Diagnostico_cruce",
    index=False
)

df_patrones.to_excel(
    writer,
    sheet_name="Patrones_elegidos",
    index=False
)

df_detalle.to_excel(
    writer,
    sheet_name="Horario_detalle",
    index=False
)

df_matriz.to_excel(
    writer,
    sheet_name="Horario_matriz",
    index=False
)

df_paralelos.to_excel(
    writer,
    sheet_name="Paralelos_demanda",
    index=False
)

df_restricciones.to_excel(
    writer,
    sheet_name="Restricciones_malla",
    index=False
)

df_validacion.to_excel(
    writer,
    sheet_name="Validacion",
    index=False
```

```

    )

    ramos_dem.to_excel(
        writer,
        sheet_name="Base_ramos_modelo",
        index=False
    )

    diagnosticos["mallas_modelo"].to_excel(
        writer,
        sheet_name="Mallas_filtradas_R",
        index=False
    )

    diagnosticos["rf_sin_malla"].to_excel(
        writer,
        sheet_name="RF_sin_malla",
        index=False
    )

    diagnosticos["malla_sin_rf"].to_excel(
        writer,
        sheet_name="Malla_sin_RF",
        index=False
    )

    mallas_completas.to_excel(
        writer,
        sheet_name="Mallas_completas",
        index=False
    )

    ajustar_hojas_excel(writer)

    ARCHIVO_METRICAS = os.path.join(
        CARPETA_DATOS,
        "metricas_eficiencia_modelo.xlsx"
    )

    def obtener_ocupaciones(resultado, ramos_paralelos):

        ocupaciones = {}

        for _, fila in ramos_paralelos.iterrows():

            ramo = fila["sigla"]
            paralelo = int(fila["paralelo"])

```

```

patron = obtener_patron_elegido(
    resultado,
    ramo,
    paralelo
)

if patron is None:
    raise RuntimeError(
        f"No se encontró horario para {ramo}-P {paralelo}."
    )

ocupaciones[(ramo, paralelo)] = set(
    patron["slots"]
)

return ocupaciones

def calcular_superposiciones(
    resultado,
    ramos_por_malla_semestre,
    ocupaciones
):
    filas = []

    total_superposiciones = 0
    total_pares_conflictivos = 0
    total_pares_evaluados = 0

    for (plan, semestre), lista_ramos in sorted(
        ramos_por_malla_semestre.items()
    ):
        lista_ramos = [
            ramo for ramo in lista_ramos
            if ramo in resultado["R"]
        ]

        superposiciones = 0
        pares_conflictivos = 0
        pares_evaluados = 0

        for ramo_1, ramo_2 in combinations(lista_ramos, 2):
            for paralelo_1 in resultado["K"][ramo_1]:
                for paralelo_2 in resultado["K"][ramo_2]:

                    pares_evaluados += 1

```

```

        horario_1 = ocupaciones[
            (ramo_1, paralelo_1)
        ]

        horario_2 = ocupaciones[
            (ramo_2, paralelo_2)
        ]

        coincidencias = horario_1.intersection(
            horario_2
        )

        if coincidencias:
            pares_conflictivos += 1
            superposiciones += len(coincidencias)

    total_superposiciones += superposiciones
    total_pares_conflictivos += pares_conflictivos
    total_pares_evaluados += pares_evaluados

    filas.append({
        "Plan": plan,
        "Semestre": semestre,
        "Asignaturas consideradas": len(lista_ramos),
        "Pares evaluados": pares_evaluados,
        "Pares con conflicto": pares_conflictivos,
        "Cantidad de superposiciones": superposiciones,
        "Resultado": (
            "Sin superposiciones"
            if superposiciones == 0
            else "Presenta superposiciones"
        )
    })

    filas.append({
        "Plan": "TOTAL",
        "Semestre": "",
        "Asignaturas consideradas": "",
        "Pares evaluados": total_pares_evaluados,
        "Pares con conflicto": total_pares_conflictivos,
        "Cantidad de superposiciones": total_superposiciones,
        "Resultado": (
            "Sin superposiciones"
            if total_superposiciones == 0
            else "Presenta superposiciones"
        )
    })

```

```

return pd.DataFrame(filas)

def calcular_bloques_ociosos(
    resultado,
    ramos_por_malla_semestre,
    ocupaciones
):

    filas = []

    for (plan, semestre), lista_ramos in sorted(
        ramos_por_malla_semestre.items()
    ):

        lista_ramos = [
            ramo for ramo in lista_ramos
            if ramo in resultado["R"]
        ]

        jornadas_con_clases = 0
        bloques_ociosos = 0

        for dia in DIAS:

            bloques_ocupados = set()

            for ramo in lista_ramos:
                for paralelo in resultado["K"][ramo]:

                    horario = ocupaciones[
                        (ramo, paralelo)
                    ]

                    for dia_horario, bloque in horario:

                        if dia_horario == dia:
                            bloques_ocupados.add(bloque)

            if not bloques_ocupados:
                continue

            jornadas_con_clases += 1

            primer_bloque = min(bloques_ocupados)
            ultimo_bloque = max(bloques_ocupados)

            for bloque in range(

```

```

        primer_bloque,
        ultimo_bloque + 1
    ):

        es_protegido = (
            dia,
            bloque
        ) in BLOQUES_PROTEGIDOS

        if (
            bloque not in bloques_ocupados
            and not es_protegido
        ):
            bloques_ociosos += 1

    promedio = (
        bloques_ociosos / jornadas_con_clases
        if jornadas_con_clases > 0
        else 0
    )

    filas.append({
        "Plan": plan,
        "Semestre": semestre,
        "Jornadas con clases": jornadas_con_clases,
        "Bloques ociosos": bloques_ociosos,
        "Promedio": round(promedio, 2)
    })

    return pd.DataFrame(filas)

def calcular_utilizacion_bloques(ocupaciones):

    filas = []

    bloques_ocupados_totales = set()
    asignaturas_totales = set()

    for dia in DIAS:

        bloques_disponibles = [
            bloque for bloque in BLOQUES
            if (dia, bloque) not in BLOQUES_PROTEGIDOS
        ]

        bloques_ocupados = set()
        asignaturas_programadas = set()

```

```

for (ramo, paralelo), horario in ocupaciones.items():

    bloques_ramo_dia = {
        bloque
        for dia, horario, bloque in horario
        if dia_horario == dia
    }

    if bloques_ramo_dia:

        bloques_ocupados.update(
            bloques_ramo_dia
        )

        asignaturas_programadas.add(ramo)
        asignaturas_totales.add(ramo)

        for bloque in bloques_ramo_dia:
            bloques_ocupados_totales.add(
                (dia, bloque)
            )

    porcentaje = (
        len(bloques_ocupados)
        / len(bloques_disponibles)
        * 100
        if bloques_disponibles
        else 0
    )

    filas.append({
        "Día": dia,
        "Bloques disponibles": len(bloques_disponibles),
        "Bloques ocupados": len(bloques_ocupados),
        "Porcentaje de utilización": round(
            porcentaje,
            2
        ),
        "Asignaturas programadas": len(
            asignaturas_programadas
        )
    })

total_disponibles = sum(
    1
    for dia in DIAS
    for bloque in BLOQUES
    if (dia, bloque) not in BLOQUES_PROTEGIDOS

```

```

    )

    total_ocupados = len(
        bloques_ocupados_totales
    )

    porcentaje_total = (
        total_ocupados
        / total_disponibles
        * 100
        if total_disponibles > 0
        else 0
    )

    filas.append({
        "Día": "Total",
        "Bloques disponibles": total_disponibles,
        "Bloques ocupados": total_ocupados,
        "Porcentaje de utilización": round(
            porcentaje_total,
            2
        ),
        "Asignaturas programadas": len(
            asignaturas_totales
        )
    })

    return pd.DataFrame(filas)

def exportar_metricas(
    resultado,
    ramos_paralelos,
    ramos_por_malla_semestre
):
    if resultado["estado"] != "Optimal":
        raise RuntimeError(
            "Las métricas no pueden calcularse porque "
            "el modelo no obtuvo una solución óptima."
        )

    ocupaciones = obtener_ocupaciones(
        resultado,
        ramos_paralelos
    )

    df_superposiciones = calcular_superposiciones(

```

```

        resultado,
        ramos_por_malla_semestre,
        ocupaciones
    )

    df_bloques_ociosos = calcular_bloques_ociosos(
        resultado,
        ramos_por_malla_semestre,
        ocupaciones
    )

    df_utilizacion = calcular_utilizacion_bloques(
        ocupaciones
    )

    os.makedirs(
        CARPETA_DATOS,
        exist_ok=True
    )

    with pd.ExcelWriter(
        ARCHIVO_METRICAS,
        engine="openpyxl"
    ) as writer:

        df_superposiciones.to_excel(
            writer,
            sheet_name="Superposiciones",
            index=False
        )

        df_bloques_ociosos.to_excel(
            writer,
            sheet_name="Bloques_ociosos",
            index=False
        )

        df_utilizacion.to_excel(
            writer,
            sheet_name="Utilizacion_bloques",
            index=False
        )

        ajustar_hojas_excel(writer)

    return {
        "superposiciones": df_superposiciones,
        "bloques_ociosos": df_bloques_ociosos,

```

```

        "utilizacion": df_utilizacion
    }

def main():

    if VERBOSE:
        print("1. Cargando bases...")

    bases = cargar_bases()

    if VERBOSE:
        print("2. Construyendo R y R_m_s...")

    (
        ramos,
        ramos_por_malla_semestre,
        mallas_completas,
        diagnosticos
    ) = construir_conjuntos_modelo(
        bases
    )

    if VERBOSE:
        print("3. Calculando Dem_r, N_r y Cap_r_k...")

    ramos_dem = preparar_demanda_y_paralelos(
        ramos
    )

    ramos_paralelos = expandir_ramo_paralelo(
        ramos_dem
    )

    if VERBOSE:
        print("4. Generando P_r...")

    patrones_por_ramo = construir_patrones(
        ramos_dem
    )

    print(
        f"Ramos incluidos en R: {len(ramos_dem)}"
    )

    print(
        f"Ramo-paralelos creados: {len(ramos_paralelos)}"
    )

```

```

if VERBOSE:
    print("5. Resolviendo el modelo...")

    resultado = resolver_modelo(
        ramos_dem,
        ramos_paralelos,
        ramos_por_malla_semestre,
        patrones_por_ramo
    )

    print()

    print(
        "Estado del modelo:",
        resultado["estado"]
    )

    if resultado["estado"] != "Optimal":

        print(
            "No se exportarán resultados ni métricas."
        )

        print(
            "Revise el modelo LP en:"
        )

        print(
            ARCHIVO_MODELO_LP
        )

        raise RuntimeError(
            f"El solver terminó con estado "
            f"{resultado['estado']}."
        )

    print(
        "Valor de la función objetivo:",
        resultado["objetivo"]
    )

if VERBOSE:
    print("6. Exportando horario...")

    exportar_resultados(
        resultado,
        ramos_dem,
        ramos_paralelos,

```

```

    ramos_por_malla_semestre,
    mallas_completas,
    diagnosticos
)

if VERBOSE:
    print("7. Calculando métricas...")

metricas = exportar_metricas(
    resultado,
    ramos_paralelos,
    ramos_por_malla_semestre
)

total_superposiciones = metricas[
    "superposiciones"
].iloc[-1]["Cantidad de superposiciones"]

utilizacion_total = metricas[
    "utilizacion"
].iloc[-1]["Porcentaje de utilización"]

print()

print(
    "Archivo de horario generado:"
)

print(
    ARCHIVO_SALIDA
)

print()

print(
    "Archivo de métricas generado:"
)

print(
    ARCHIVO_METRICAS
)

print()

print(
    "Superposiciones totales:",
    total_superposiciones
)

```

```
print(
    "Utilización total de bloques:",
    f"{utilizacion_total}%"
)
main()
```