Simulación de Horario de Clases

Optimización de Recursos

Cifuentes Argueta, Sergio Daniel
Centro Universitario de Occidente
Quetzaltenango, Septiembre 2023

INDICE

RESUMEN

Este proyecto de simulación aborda la optimización de la programación del aula en una institución de educación superior. El objetivo principal es desarrollar un modelo de programación que maximice la utilización de los recursos disponibles y al mismo tiempo cumpla con las limitaciones relacionadas con la disponibilidad de maestros, el tamaño de las clases y las capacidades de las salas. La simulación emplea un enfoque de modelado de eventos discretos para representar el proceso de programación, con entidades que incluyen clases, profesores y aulas.

El modelo de simulación se construye a partir de datos del mundo real, incluidos registros históricos de programación, preferencias de los docentes y especificaciones del aula. Los supuestos y restricciones se documentan de forma transparente para proporcionar contexto para el proceso de modelado.

Se llevan a cabo una serie de experimentos y escenarios para evaluar el desempeño del modelo de programación en diferentes condiciones. Los resultados se presentan y analizan para evaluar el impacto de varios factores en los resultados de la programación, como los conflictos de programación y la utilización de recursos.

Los hallazgos de este proyecto de simulación contribuyen a la toma de decisiones informadas en la programación del aula, ofreciendo información sobre estrategias de programación óptimas y destacando áreas de mejora. Las conclusiones y recomendaciones del proyecto tienen el potencial de mejorar la asignación de recursos y la eficiencia de la programación dentro de la institución educativa.

INTRODUCCION

En un mundo dinámico y en constante evolución, la asignación eficiente de recursos es fundamental para el éxito de las organizaciones e instituciones en diversos ámbitos. La gestión de recursos, como aulas, profesores y horarios de los estudiantes, juega un papel fundamental en las instituciones educativas. Optimizar la asignación de estos recursos es un desafío multifacético que impacta la calidad de la educación, la satisfacción de docentes y estudiantes y la efectividad general de la institución.

La programación de las aulas y de los profesores en las instituciones de educación superior es un proceso complejo e intrincado que implica numerosas limitaciones y consideraciones. Requiere lograr un delicado equilibrio entre el tamaño de las clases, la disponibilidad de docentes, la capacidad de las salas y las diversas necesidades de estudiantes y educadores. Estos retos aumentan con carreras que comparten clases y docentes. Además, el proceso de programación debe cumplir con las políticas, regulaciones y limitaciones operativas institucionales.

Este proyecto de simulación intenta abordar los desafíos inherentes a la programación del aula mediante el desarrollo de un modelo de programación integral. El objetivo es crear una simulación que optimice la asignación de aulas, maestros y horarios de clases, lo que en última instancia conducirá a una mejor utilización de los recursos, una reducción de los conflictos de programación y mejores resultados educativos.

Este documento sirve como una descripción detallada del proyecto de simulación, proporcionando un examen exhaustivo del desarrollo, la validación, la experimentación y los hallazgos del modelo. A través de una combinación de conocimientos basados en datos y escenarios basados en simulación, este proyecto pretende aportar recomendaciones y estrategias valiosas para optimizar la programación de las aulas en instituciones de educación superior.

En las siguientes secciones, profundizaremos en las complejidades del modelo de simulación, exploraremos los supuestos y restricciones que guían nuestro enfoque y presentaremos los resultados de varios experimentos de programación. Nuestro objetivo final es ofrecer conocimientos prácticos y soluciones basadas en evidencia que puedan informar y mejorar las prácticas de programación del aula, promoviendo así la eficiencia y la calidad de la educación en nuestra institución.

OBJETIVOS

Objetivo General:

- Optimizar la asignación de aulas, docentes y horarios de clases en una institución de educación superior.

Objetivos Específicos:

- Minimizar los casos de hacinamiento en las aulas optimizando las asignaciones de clases y la asignación de aulas.
- Identificar y resolver conflictos de programación, incluidas clases superpuestas y asignaciones dobles de recursos.
- Optimizar la utilización de las aulas y otros recursos, como laboratorios o instalaciones especializadas.

DEFINICIÓN DEL SISTEMA

El sistema que se modela en esta documentación de simulación es el Horario de Clases de una institución de educación superior como lo es el Centro Universitario de Occidente (CUNOC). Más específicamente para las carreras de ingeniería, siendo estas clases llevadas a cabo en el mismo módulo, compartiendo salones y docentes. El Horario de Clases es un sistema complejo y dinámico responsable de la asignación de aulas, docentes, horarios de clases y alumnos para facilitar el proceso educativo. Abarca una variedad de entidades, procesos y limitaciones, todos los cuales interactúan para garantizar el uso eficiente de los recursos y la prestación de una educación de alta calidad.

Entidades: Las entidades principales incluyen clases, profesores, aulas y períodos. Cada clase tiene requisitos específicos en términos de tamaño de clase y materia. Los profesores tienen experiencia en la materia, horarios de disponibilidad y cursos que están contratados para dar. Las aulas tienen diferentes capacidades y diferentes disponibilidades.

Eventos: Los eventos en el sistema incluyen sesiones de clase, disponibilidad de maestros y reservas de salas. Estos eventos interactúan para crear el horario de clases.

Procesos: Los procesos implican la asignación de clases a maestros, aulas específicas y períodos, respetando al mismo tiempo las limitaciones y políticas institucionales. La resolución de conflictos al momento de asignar profesores o salones, los ajustes de programación y la actualización de datos son procesos integrales.

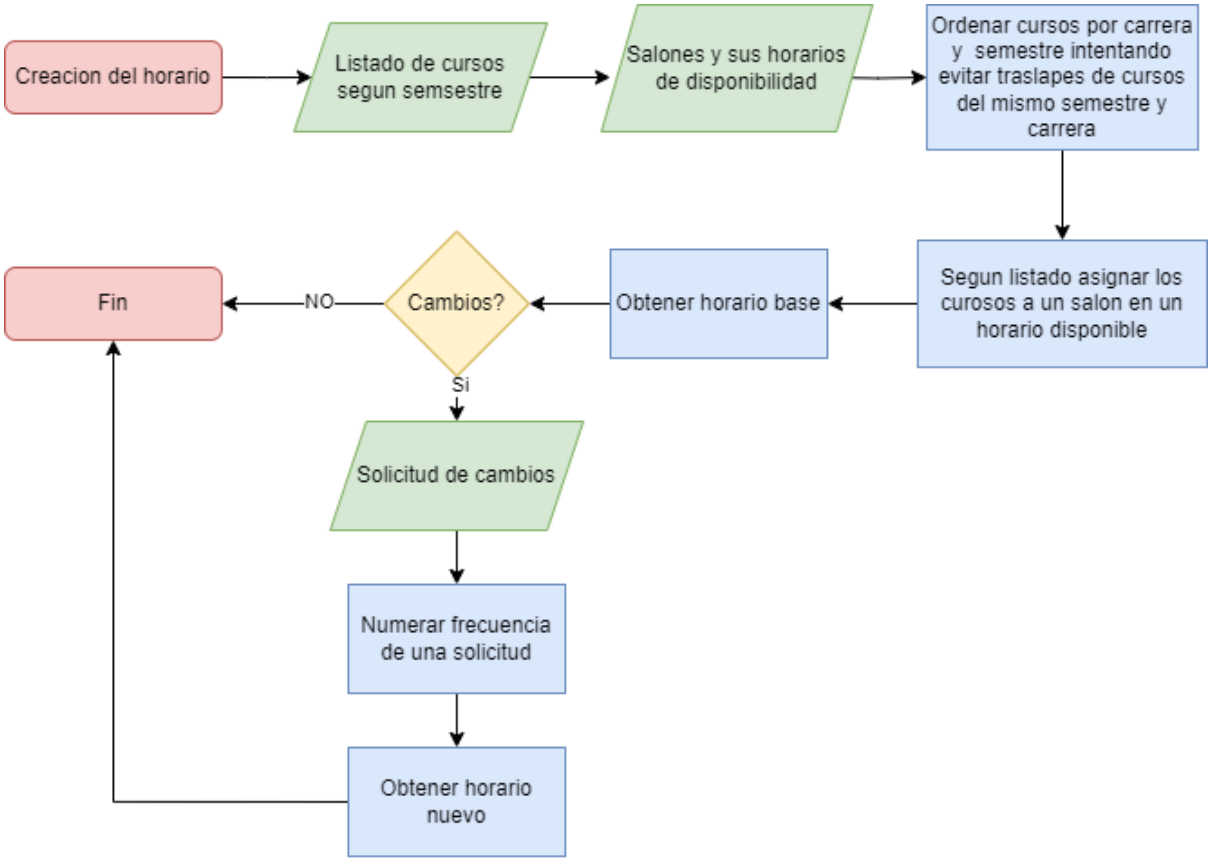
Restricciones: El sistema opera bajo restricciones tales como restricciones en el tamaño de las clases, disponibilidad de maestros, capacidad de las salas, disponibilidad de salas y capacidad para impartir una clase de parte del docente.

Fuentes de datos: Las fuentes de datos para la simulación incluyen registros de cursos existentes, preferencias de los maestros, horarios de disponibilidad, especificaciones del aula y datos de inscripción de estudiantes.

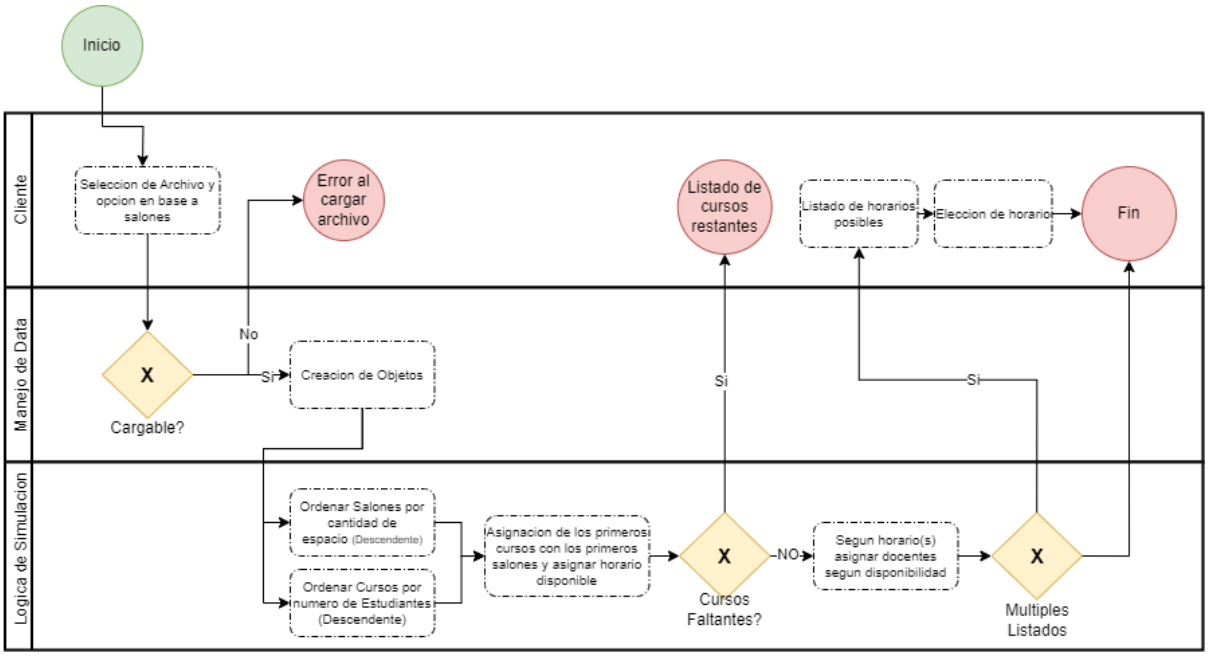
Análisis del modelo actual

Existen varias diferencias entre el modelo actual y el modelo que se quiere obtener. Para empezar la asignación de estudiantes tiene diferente orden, en el modelo actual los estudiantes se asignan después de la creación del horario y para el modelo que se quiere construir es antes. Además

todo este proceso es manual actualmente. Aquí podemos ver un diagrama que representa lo que ya existe.



Y en seguida veremos un diagrama BPMN Modelo(s) del proceso propuesto para la creación de horarios en su proyecto.



Objetivos del modelo

Optimizar la programación del aula: el objetivo principal de esta simulación es optimizar la asignación de aulas, docentes y horarios de clases dentro de una institución de educación superior. La optimización tiene como objetivo mejorar la utilización de recursos, reducir los conflictos de programación y mejorar la eficiencia general de la programación del aula.

Mejorar la calidad educativa: un objetivo secundario es mejorar la calidad de la educación dentro de la institución asegurando que las tareas de clase estén bien equilibradas, los maestros tengan una carga de trabajo manejable y los estudiantes tengan acceso a instalaciones adecuadas.

Índices de medición

Para evaluar la efectividad de la simulación utilizaremos los siguientes índices de medición:

Tasa de utilización de recursos: este índice mide el porcentaje de aulas disponibles y horas de enseñanza que se utilizan de manera efectiva. Una tasa más alta indica un mejor uso de los recursos.

Comodidad de los estudiantes: mide el porcentaje de aulas que se encuentran para ciertos cursos muy llenas y con poco comodidad para los estudiantes. Se buscara minimizar este porcentaje.

Distribución de la carga de trabajo de los docentes: una medida eficaz del equilibrio de la carga de trabajo de los docentes es la desviación estándar de las horas de enseñanza entre los miembros del cuerpo docente. Los valores de desviación estándar más bajos indican una distribución más equitativa.

FORMULACIÓN DEL MODELO

El modelo de simulación para la programación de aulas utiliza técnicas de programación lineal para optimizar la asignación de clases, maestros y aulas, respetando al mismo tiempo las restricciones y las políticas institucionales. Esta sección describe los componentes clave del modelo, incluidas las variables de decisión, las funciones objetivo y las restricciones.

Programación Lineal

En esencia, la programación lineal implica tomar decisiones (representadas como variables de decisión) de una manera que logre de manera óptima un objetivo respetando varias limitaciones o restricciones. En nuestro contexto, el objetivo es programar clases, docentes y aulas de manera eficiente, y las limitaciones incluyen factores como las capacidades de las aulas, la disponibilidad de docentes y capacidad de los docentes.

En nuestro modelo de programación lineal, introducimos variables de decisión (X_ijt e Y_it) que nos ayudan a representar las decisiones de programación. Estas variables binarias toman valores de 0 o 1, donde 1 indica una decisión de programación específica (por ejemplo, programar una clase en un aula en particular durante un intervalo de tiempo determinado) y 0 representa la ausencia de esa decisión.

El objetivo principal del modelo de programación lineal es maximizar la utilización general de los recursos disponibles y minimizar los conflictos de programación. Logramos esto a través de la función objetivo, que cuantifica el impacto de nuestras decisiones de programación en la utilización de recursos.

Para garantizar que la programación siga siendo factible y se adhiera a las restricciones y políticas establecidas por la institución, introducimos un conjunto de restricciones lineales. Estas restricciones rigen las limitaciones del tamaño de las clases, la disponibilidad de los maestros, las reglas de reserva de salas y otras consideraciones.

Variables de Decision

X_it : Variable de decisión binaria que es igual a 1 si la clase i está programada en el horario t; 0 en caso contrario.
Y_itj: Variable de decisión binaria que es igual a 1 si la clase i está programada en el horario t asignado al docente j; 0 en caso contrario.
Z_itf: Variable de decisión binaria que es igual a 1 si la clase i está programada en el horario t asignado al salón f; 0 en caso contrario.

Función objetiva:

El objetivo del modelo de programación lineal es maximizar la asignación de los cursos y minimizar los conflictos de programación. La función objetiva la podemos declarar como un variable que será establecida por el usuario según lo que quiere buscar al momento de realizar la simulación La función objetivo es la siguiente:

Maximizar: $\Sigma(\mathbf{X\_it})$ para todos i, t

Restricciones:

1. Máximo de un periodo asignado por clase:
   $\Sigma(X\_it) \leq 1$ para todo periodo t. Asegura que solo un periodo se le asigna una clase

2. Traslape de cursos del mismo semestre:
   $\Sigma(X\_i_1t) + \Sigma(X\_i_2t) \leq 1$ para todo periodo t, donde i_1 y i_2 representan cursos del mismo semestre, carrera y sección . Asegura que los estudiantes puedan llevar sus cursos sin traslapes dependiendo del semestre en el que se encuentren.

3. Cursos Obligatorios:
   $\Sigma(X\_it) \geq 1$ para todo periodo t, donde i representa un curso mandatorio. Asegura que estos cursos tengan prioridad de asignación a diferencia del resto.

4. Traslape de docentes:
   $\Sigma\ (Y\_itj) \leq 1$ para toda clase i. Asegura que el maestro solo sea asignado a una clase por periodo.

5. Capacidad del docente:
   $\Sigma\ (Y\_itj) \leq$ capacidad_j dónde capacidad_j es 1 si el docente es capacitado para impartir este curso y 0 si lo contrario.

6. Disponibilidad del docente:
   $Y\_Itj \leq$ disponibilidad_j donde disponibilidad_j es 1 si el docente está disponible en ese periodo y 0 si lo contrario.

7. Traslape de salón:
   $\Sigma\ (Z\_itf) \leq 1$ para toda clase i. Asegura que el salón solo sea asignado a una clase por periodo.

8. Capacidad del salón:
   Estudiantes_i ≤ capacidad_f para todo Z_itf. Asegura que los estudiantes de cada clase no sobrepasen la capacidad máxima de los salones.

9. Disponibilidad del salón:
   Z_itf≤ disponibilidad_f donde disponibilidad_f es 1 si el salón está disponible en ese periodo y 0 si lo contrario.

10. Un maestro por clase:
    X_it = Σ (Y_itj) para cada docente j. Asegura que cada asignación de periodo t y clase cuentan con un maestro máximo.

11. Un salon por clase:
    X_it = Σ (Z_itf) para cada salón f. Asegura que cada asignación de periodo t y clase cuentan con un salón máximo.

COLECCIÓN DE DATOS

Los registros históricos de programación sirven como fuente de datos principal para nuestra simulación de programación en el aula. Estos registros brindan información valiosa sobre prácticas de programación anteriores, asignaciones de clases, disponibilidad de maestros y reservas de aulas dentro de nuestra institución educativa.

Formato

Los registros de programación históricos se proporcionan en formato CSV (valores separados por comas). Los archivos CSV se utilizan ampliamente por su simplicidad y compatibilidad con diversas herramientas de procesamiento de datos y sistemas de bases de datos. Cada archivo CSV representa un aspecto específico de los datos históricos de programación, que incluyen:

- Profesores.csv : Contiene información de los docentes y sus respectivas carreras.

```
id,nombre,carrera
1,"Fransisco Rojas","Sistemas"
2,"Moises Granados","Sistemas"
3,"Pedro Domingo","Sistemas"
4,"Mauricio Lopez","Sistemas"
5,"Jose Maldonado","Civil"
6,"Humberto","Civil"
7,"Carlos Chavarria","Civil"
```

- Materias.csv : Contiene información de los diferentes cursos como lo es el semestre y carrera.

```
id,nombre,carrera,semestre,obli,no_periodos
169,"Matematica Basica 1","Comun",1,true,2
17,"Social Humanista 1","Comun",1,false,1
216,"Quimica 1","Comun",1,true,1
170,"Matematica Basica 2","Comun",2,true,2
29,"Social Humanista 2","Comun",2,true,1
5,"Tecnicas De Estudio","Comun",2,true,1
147,"Fisica Basica","Comun",2,true,1
107,"Matematica Intermedia1","Comun",3,true,2
150,"Fisica 1","Comun",3,true,1
112,"Matematica Intermedia 2","Comun",4,true,1
```

- Salons.csv : Contiene información de los salones que dispone el módulo, incluyendo capacidades de estudiantes.

```
id,nombre,capacidad_comoda,capacidad_maxima
1,"Salon 1",40,50
2,"Salon 2",40,50
3,"Salon 3",40,50
4,"Salon 4",60,70
5,"Salon 5",60,70
```

- Disponibilidad_Profesor.csv : Contiene horarios de los profesores, los cuales serán de suma importancia al momento de asignación.

```
id_profesor,inicio,fin
1,"12:00","22:00"
2,"10:00","12:00"
2,"14:00","20:00"
3,"14:00","22:00"
4,"14:00","22:00"
5,"14:00","22:00"
6,"14:00","22:00"
```

- Disponibilidad_Salon.csv : Contiene horarios de la disponibilidad de los salones, los cuales serán de suma importancia al momento de asignación

```
id_salon,inicio,fin
1,"12:00","14:00"
1,"14:00","22:00"
2,"14:00","22:00"
3,"14:00","22:00"
4,"14:00","22:00"
5,"14:00","22:00"
```

- Profesor_Materia.csv : Contiene información sobre la capacidad de cada profesor.

```
id_profesor,id_curso,fijo
1,2820,true
1,2796,false
1,2835,false
1,2819,true
1,2823,false
2,2796,true
2,2798,false
2,2796,false
2,2800,true
```

- Asignaciones.csv: Contiene información de las secciones de los cursos y el número de estudiantes asignados.

```
id_materia,sec,numero_estudiantes
169,"A",60
169,"B",60
169,"C",55
169,"D",45
17,"A",50
17,"B",50
17,"C",40
17,"D",40
216,"A",50
216,"B",50
```

Proceso de importación de datos:

Para incorporar los datos históricos de programación en nuestra simulación, empleamos un proceso de importación de datos que sigue estos pasos:

Análisis de archivos CSV: analizamos los archivos CSV utilizando bibliotecas de procesamiento de datos para extraer información estructurada.

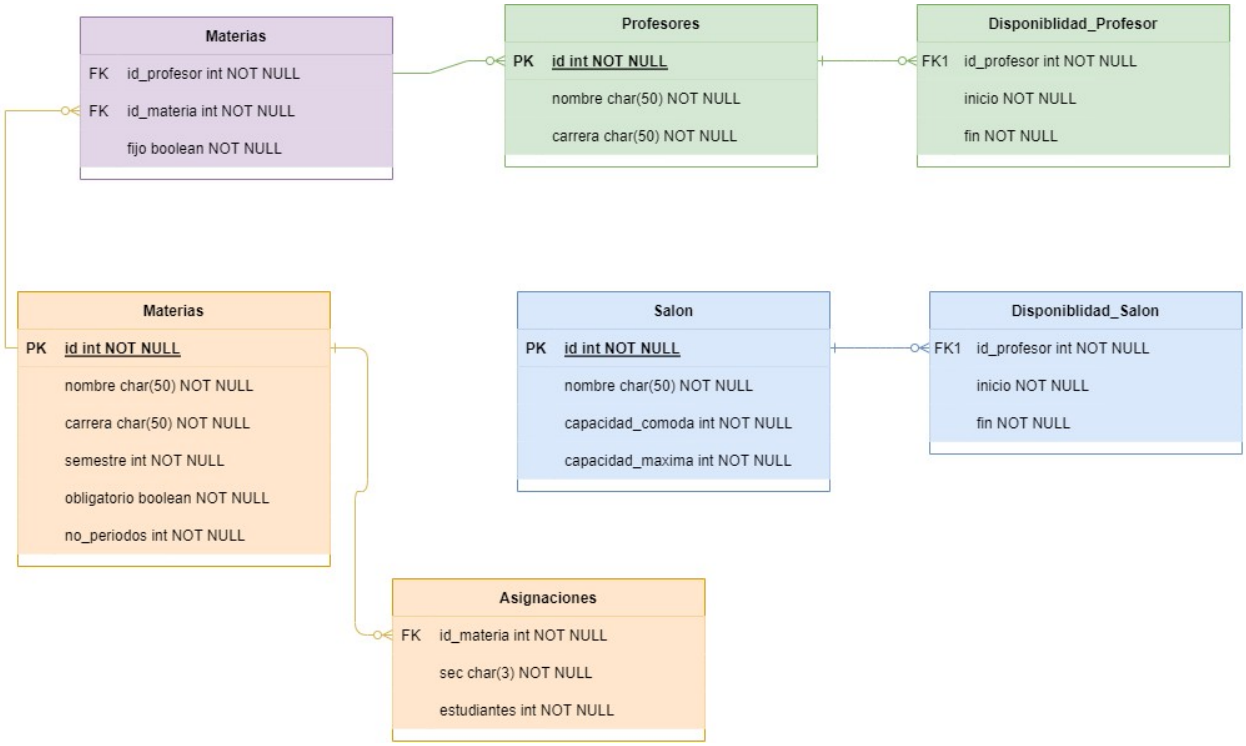
Validación de datos: los datos analizados se someten a comprobaciones de validación para garantizar su integridad y coherencia.

Importación de base de datos: los datos validados se importan a un sistema de gestión de bases de datos relacionales (RDBMS) que sirve como depósito de datos para la simulación.

Normalización: se aplican técnicas de normalización de datos para organizar los datos de manera eficiente, reduciendo la redundancia y garantizando la integridad de los datos.

Esquema de base de datos:

El RDBMS emplea un esquema que refleja la estructura de los datos históricos de programación. Este esquema incluye tablas para clases, profesores, aulas, asignaciones, disponibilidad de profesores y disponibilidad de salas. Las relaciones entre estas tablas se establecen para facilitar la recuperación y el análisis de datos.



IMPLEMENTACIÓN DEL MODELO

Lenguaje de implementación

El modelo de simulación se implementa utilizando Python, un lenguaje de programación versátil y ampliamente utilizado, y la biblioteca Pyomo, que proporciona una poderosa plataforma para modelar y resolver problemas de optimización.

Herramientas a utilizar

Pyomo:

Pyomo es un paquete de software de código abierto basado en Python que admite un conjunto diverso de capacidades de optimización para formular, resolver y analizar modelos de optimización.

Una capacidad central de Pyomo es modelar aplicaciones de optimización estructuradas. Pyomo se puede utilizar para definir problemas simbólicos generales, crear instancias de problemas específicos y resolver estas instancias utilizando solucionadores comerciales y de código abierto. Los objetos de modelado de Pyomo están integrados en un lenguaje de programación de alto nivel con todas las funciones que proporciona un rico conjunto de bibliotecas de soporte, lo que distingue a Pyomo de otros lenguajes de modelado algebraico como AMPL, AIMMS y GAMS.

Pyomo admite una amplia gama de tipos de problemas, que incluyen:

- Programación lineal
- Programación cuadrática
- Programación no lineal

En esta simulación se usa la herramienta de programación lineal

Pandas

Pandas es una muy popular librería de código abierto dentro de los desarrolladores de Python, y sobre todo dentro del ámbito de Data Science y Machine Learning, ya que ofrece unas estructuras muy poderosas y flexibles que facilitan la manipulación y tratamiento de datos.

Pandas surgió como necesidad de aunar en una única librería todo lo necesario para que un analista de datos pudiese tener en una misma herramienta todas las funcionalidades que necesitaba en su día a día, como son: cargar datos, modelar, analizar, manipular y prepararlos.

Las dos estructuras de datos principales dentro del paquete Pandas son:

Series: array unidimensional etiquetado capaz de almacenar cualquier tipo de dato.
DataFrame: estructura bidimensional con columnas que pueden ser también de cualquier tipo. Estas columnas son a su vez Series.

Dado que vivimos en un mundo en el que los datos son de muy distintas categorías, Pandas se realizó con el objetivo de poder tratar con el mayor número posible de casuísticas entre tipos de datos. Es muy simple cargar datos desde diferentes tipos de archivos (csv, json, html, etc.), así como guardarlos.

CBC

Cbc (Coin-or branch and cut) es un solucionador de programación lineal entera mixta de código abierto escrito en C++. Se puede utilizar como una biblioteca invocable o como un ejecutable independiente. Se puede utilizar de diversas formas a través de varios sistemas de modelado, paquetes, etc.

Método de la M grande

Corresponde a una variación del Algoritmo simplex para penalizar la presencia de variables artificiales mediante la introducción de una constante M definida como un valor muy grande.

Se inicia obteniendo la forma aumentada del modelo de programación lineal considerando para cada restricción funcional la variable de holgura, de exceso o artificial necesarias para tener el modelo en forma de ecuación. Como las variables artificiales no forman parte del modelo de programación lineal original entonces se necesita hacer algo para igualarlas a cero en el momento que se alcance la iteración óptima, esto se logra asignando una penalización.

Implementacion

Importar Datos

Ya definimos los datos de entrada de esta simulación y su formato CSV o Tablas de bases de datos. Así que los vamos a convertir a DataFrames para su facil uso, con ayuda de la librería Pandas.

CVS según PATH

```
    def load_data(self, path=constant.PATH):
        try:
            self.df_asignacion = pd.read_csv(path+constant.ASIGNACIONES)
    self.df_asignacion.insert(0,'id',range(0, 0 + len(self.df_asignacion)))
        except FileNotFoundError:
            print("Asignacion data not found.")
            return False
        try:
            self.df_disp_prof = pd.read_csv(path+constant.DISP_PROF)
        except FileNotFoundError:
            print("Disponibilidad Profesor data not found")
            return False
        try:
            self.df_disp_salon = pd.read_csv(path+constant.DISP_SALON)
        except FileNotFoundError:
            print("Disponibilidad Salon data not found")
            return False
        try:
            self.df_materias = pd.read_csv(path+constant.MATERIAS)
```

```
        except FileNotFoundError:
            print("Materias data not found")
            return False
        try:
            self.df_profesores = pd.read_csv(path+constant.PROFESORES)
        except FileNotFoundError:
            print("Profesores data not found")
            return False
        try:
            self.df_prof_materia = pd.read_csv(path+constant.PROF_MATERIA)
        except FileNotFoundError:
            print("Profesores Materia data not found")
            return False
        try:
            self.df_salon = pd.read_csv(path+constant.SALONES)
        except FileNotFoundError:
            print("Salones data not found")
            return False
        return True
```

Base de Datos

```
self.df_materias = pd.read_sql_table("materia", conn)
self.df_profesores = pd.read_sql_table("profesor", conn)
self.df_salon = pd.read_sql_table("salon", conn)
self.df_asignacion = pd.read_sql_table("asignacion", conn)
self.df_prof_materia = pd.read_sql_table("prof_materia", conn)
self.df_disp_prof = pd.read_sql_table("disp_profesor", conn)
self.df_disp_salon = pd.read_sql_table("disp_salon", conn)
```

Comenzamos importando los datos relevantes a un objeto Pyomo *ConcreteModel* usando Sets (similares a matrices) y Params (pares clave-valor). Estos Sets y Params representarán información que necesitaremos para nuestra implementación. Sets representan listas de datos importantes y Params son descripciones de estas listas.

Sets

```
        # List of classes IDs
model.CLASSES = pe.Set(initialize=self.df_asignacion["id"].tolist())
        # List of teachers IDs
model.TEACHERS = pe.Set(initialize=self.df_profesores["id"].tolist())
```

```
        # List of rooms IDs
model.ROOMS = pe.Set(initialize=self.df_salon["id"].tolist())
```

Params

```
model.CLASS_NUM_STUDENTS=pe.Param(model.CLASSES,initialize=
self._generate_class_students())
model.ROOM_MAX_CAPACITY=pe.Param(model.ROOMS,initialize=
self._generate_room_maximum())
```

Usamos una serie de funciones auxiliares para inicializar los conjuntos y parámetros de Pyomo. Estos se omiten aquí por brevedad, pero se pueden encontrar en el repositorio de Github.

También definimos un Set llamado *ASSIGNMENTS* que contiene una lista de todas las combinaciones posibles de (Clase, Período). Esto representa todas las posibles decisiones de asignación de casos disponibles:

```
model.ASSIGNMENTS=pe.Set(initialize=model.CLASSES*model.PERIODS,dimen=2)
```

Variables de Decision

La decisión principal es asignar las clases a los periodos. Esto requiere que se tome una decisión binaria de sí/no para cada combinación de caso y sesión en el conjunto de *ASSIGNMENTS* anterior.

También queremos asignar un profesor a este *ASSIGNMENTS* para poder impartirlo y finalmente un respectivo salón para llevar a cabo la clase. Así que representaremos todas las posibles opciones con *ASSIGNMENTS*TEACHERS* y *ASSIGNMENTS*ROOMS* respectivamente y les asignaremos un si o no, siendo de un dominio Binario.

Definimos estas variables de decisión en nuestro modelo Pyomo de la siguiente manera:

```
model.PERIOD_ASSIGNED = pe.Var(model.ASSIGNMENTS, domain=pe.Binary)

model.TEACHER_ASSIGNED=pe.Var(model.ASSIGNMENTS*model.TEACHERS,domain=pe.B
inary)

model.ROOM_ASSIGNED=pe.Var(model.ASSIGNMENTS*model.ROOMS,domain=pe.Binary)
```

Función Objetiva

Una ventaja de la programación lineal es la flexibilidad para definir una función objetivo que represente nuestras necesidades comerciales. Somos libres de definir cualquier función (lineal) y, en nuestro caso, nuestro objetivo es maximizar la asignación de las clases, pero esto lo podemos cambiar después:

```
def objective_function(model):
  return  sum([model.PERIOD_ASSIGNED[classS,  period]  for  classS  in  model.CLASSES
for          period          in          model.PERIODS])model.OBJECTIVE          =
pe.Objective(rule=objective_function, sense=pe.maximize)
```

Restricciones

A continuación, agregamos nuestras restricciones. Las restricciones capturan todas las reglas que garantizan que la solución devuelta por el modelo constituya un horario de clases factible. Ya definimos las restricciones necesarias anteriormente así que solo los representaremos con pyomo utilizando sus variables *Constraint* y utilizando en ocasiones el método del M Grande.

```
        #Only one period assigned to each class
def period_assignment(model, classS):
  return sum([model.PERIOD_ASSIGNED[(classS, period)] for period in model.PERIODS]) <= 1
model.PERIOD_ASSIGNMENT=pe.Constraint(model.CLASSES,rule=period_assignment)

        #Classes from the same semester and career can't overlap
def same_semester(model, class1, class2, num_period, period):
  return  model.PERIOD_ASSIGNED[(class1,  period)]  +  model.PERIOD_ASSIGNED[(class2,  period+num_period)]
<=1
model.SAME_SEMESTER = pe.Constraint(model.DISJUNCTIONS_SAME_SEMESTER, rule=same_semester)

#Mandatory classes MUST be assigned before non mandatory classes
def mandatory_classes(model, classS):
     return sum([model.PERIOD_ASSIGNED[(classS, period)] for period in model.PERIODS]) >=1
model.MANDATORY_CLASSES=pe.Constraint(model.DISJUNCTIONS_MANDATORY_CLASSES,  rule=mandatory_classes)

#Teacher can only teach one class at a time
def no_teacher_overlap(model, period, teacher):
      return sum([model.TEACHER_ASSIGNED[( class1, period, teacher)] for class1 in model.CLASSES]) <=1
model.TEACHER_OVERLAP = pe.Constraint(model.DISJUNCTIONS_TEACHER, rule=no_teacher_overlap)

#Teachers assigned to classes must be capable of teaching
def teacher_capacity(model, classS, teacher):
    return  sum([model.TEACHER_ASSIGNED[(  classS,  period,  teacher)]  for  period  in  model.PERIODS])
<=model.PARAM_TEACHER_CLASS[classS,teacher]
model.TEACHER_CAPACITY = pe.Constraint(model.TEACHER_CLASS, rule=teacher_capacity)

#Classes assigned to a teacher must be within the room schedule
def teacher_schedule(model, classS, num_period, period, teacher):
```

```
    return    model.TEACHER_ASSIGNED[(claS,    period,    teacher)]    <=    model.PARAM_TEACHER_PERIOD
[teacher,period+num_period]
model.DISJUNCTIONS_PERIOD_TEACHER_RULE=pe.Constraint(model.TEACHER_DURATION_TWO,rule=teacher_schedule)

#Room can only be used by one clas at a time
def no_room_overlap(model, period, room):
     return sum([model.ROOM_ASSIGNED[( class1, period, room)] for class1 in model.CLASSES]) <=1
model.DISJUNCTIONS_ROOM_RULE = pe.Constraint(model.DISJUNCTIONS_ROOM, rule=no_room_overlap)

#Students assigned to a class must not surpass the room capacity
def room_capacity(model, claS, period, room):
  return model.CLASS_STUDENTS[claS] <= model.ROOM_MAX_CAPACITY[room]+ ((1 - model.ROOM_ASSIGNED[claS,
period,room]) *model.M)
model.ROOM_CAPACITY_RULE = pe.Constraint(model.DISJUNCTIONS_ROOM_CAPACITY, rule=room_capacity)

#Classes assigned to a room must be within the room schedule
def room_schedule(model, claS, num_period, period, room ):
  return model.ROOM_ASSIGNED[( claS, period, room)]<=  model.PARAM_ROOM_PERIOD[room,period+num_period]
model.DISJUNCTIONS_PERIOD_ROOM_RULE = pe.Constraint(model.CLASS_DURATION_TWO, rule=room_schedule)

#A class can only be assigned to one teacher
def teacher_per_class(model, claS, period):
   return   model.PERIOD_ASSIGNED[(claS,   period)]   ==   sum([model.TEACHER_ASSIGNED[(claS,   period,
teachers)] for teachers in model.TEACHERS])
model.TEACHER_PER_CLASS = pe.Constraint(model.ASSIGNMENTS, rule=teacher_per_class)

#A class can only be assgned to one room
def room_per_class(model, claS, period):
   return model.PERIOD_ASSIGNED[(claS, period)] == sum([model.ROOM_ASSIGNED[(claS, period, room)] for
room in model.ROOMS])
model.ROOM_PER_CLASS = pe.Constraint(model.ASSIGNMENTS, rule=room_per_class)
```

Resolviendo el modelo

Una ventaja de utilizar una interfaz como Pyomo es que es fácil probar diferentes solucionadores lineales sin tener que reescribir el modelo en otro lenguaje de codificación.

Aquí utilizamos Coin-or Branch and Cut (CBC), un solucionador de programas de entornos mixtos de código abierto (https://github.com/coin-or/Cbc).

El siguiente fragmento de código resuelve el modelo utilizando la clase SolverFactory de Pyomo. La clase puede tomar una serie de parámetros de ajuste para controlar el funcionamiento del solucionador elegido, pero por simplicidad, mantenemos la configuración predeterminada excepto por un límite de tiempo de 100 segundos.

```
cbc_name = "cbc"
```

```
path="C:\\Users\\sergi\\Documents\\Coding\\Python\\schedule-sim\\schedule-sim\\
resources\\scripts\\2\\"
options = {"seconds": 100}
scheduler = Scheduler()

scheduler.solve(solver_name=cbc_name, solver_path=cbc_path, options= options)
```

```
solver_results = solver.solve(self.model, tee=True)
results=[]
self.dict={}
for (claS, period, room) in self.model.ROOM_ASSIGNED:
    if value(self.model.ROOM_ASSIGNED[claS, period,room]) == 1:
       results.append({"Class": claS,
                       "Period": period,
                        "Room": room})
       self.df_times = pd.DataFrame(results)
       for(claS, period, teacher) in self.model.TEACHER_ASSIGNED:
            if value(self.model.TEACHER_ASSIGNED[claS, period,teacher]) ==1:
                self.df_times.loc[self.df_times['Class'] == claS, 'Teacher']=teacher
        all_classes = self.model.CLASSES.value_list
        self.classes_assigned = []
        for (claS, period) in self.model.PERIOD_ASSIGNED:

            if value(self.model.PERIOD_ASSIGNED[claS, period]) == 1:
                self.classes_assigned.append(claS)

        self.classes_missed = list(set(all_classes).difference(self.classes_assigned))
        self.is_solved=True
```

Resultados

Al correr la simulación con archivos CSV de prueba los resultados se muestran de la siguiente manera en consola en donde se asignaron los 31 clases en menos de 25 segundos:

```
[{'Class': 0, 'Period': 7, 'Room': 4}, {'Class': 1, 'Period': 9, 'Room': 4},
{'Class': 2, 'Period': 4, 'Room': 2}, {'Class': 3, 'Period': 6, 'Room': 3},
{'Class': 4, 'Period': 5, 'Room': 2}, {'Class': 5, 'Period': 8, 'Room': 2},
{'Claoom': 1}, {'Class': 12, 'Period': 5, 'Room': 8}, {'Class': 13, 'Period': 3,
'Room': 8}, {'Class': 14, 'Period': 8, 'Room': 1}, {'Class': 15, 'Period': 7,
'Room': 8}, {'Class': 16, 'Period': 6, 'Room': 10}, {'Class': 17, 'Period': 5,
'Room': 6}, {'Class': 18, 'Period': 4, 'Room': 10}, {'Class': 19, 'Period': 8,
'Room': 5}, {'Class': 20, 'Period': 7, 'Room': 1}, {'Class': 21, 'Period': 5,
'Room': 1}, {'Class': 22, 'Period': 7, 'Room': 3}, {'Class': 23, 'Period': 4,
```

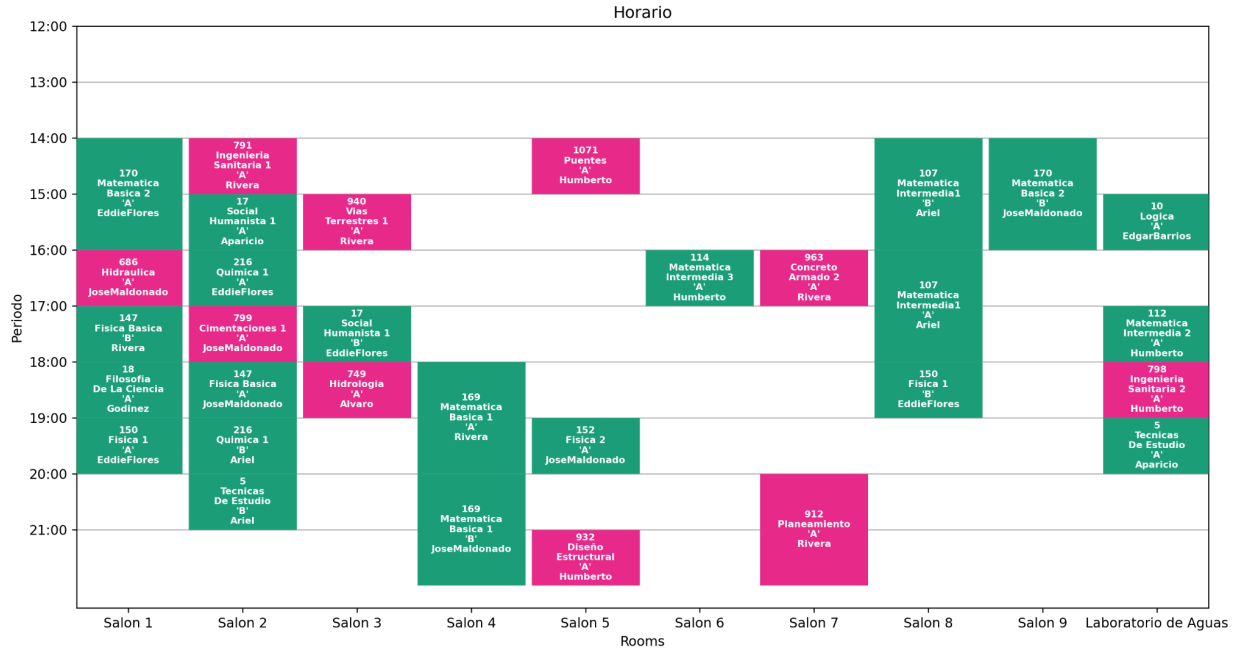
```
'Room': 3}, {'Class': 24, 'Period': 6, 'Room': 2}, {'Class': 25, 'Period': 10,
'Room': 5}, {'Class': 26, 'Period': 7, 'Room': 10}, {'Class': 27, 'Period': 3,
'Room': 2}, {'Class': 28, 'Period': 5, 'Room': 7}, {'Class': 29, 'Period': 3,
'Room': 5}, {'Class': 30, 'Period': 9, 'Room': 7}]
```

```
Result - Optimal solution found

Objective value:                -31.00000000
Enumerated nodes:               73
Total iterations:               53148
Time (CPU seconds):             24.74
Time (Wallclock seconds):       24.74

Total time (CPU seconds):       24.84   (Wallclock seconds):       24.84
```

Utilizamos una libreria graficadora para tener una mejor visualizacion de lo que estamos construyendo. En este caso utilizamos *matplotlib.pyplot* y el resultado es el siguiente:



VERIFICACIÓN Y VALIDACIÓN

Verificacion

Para llevar a cabo el proceso de verificación, se revisará a fondo el modelo y los pasos que se implementan para resolverlo.

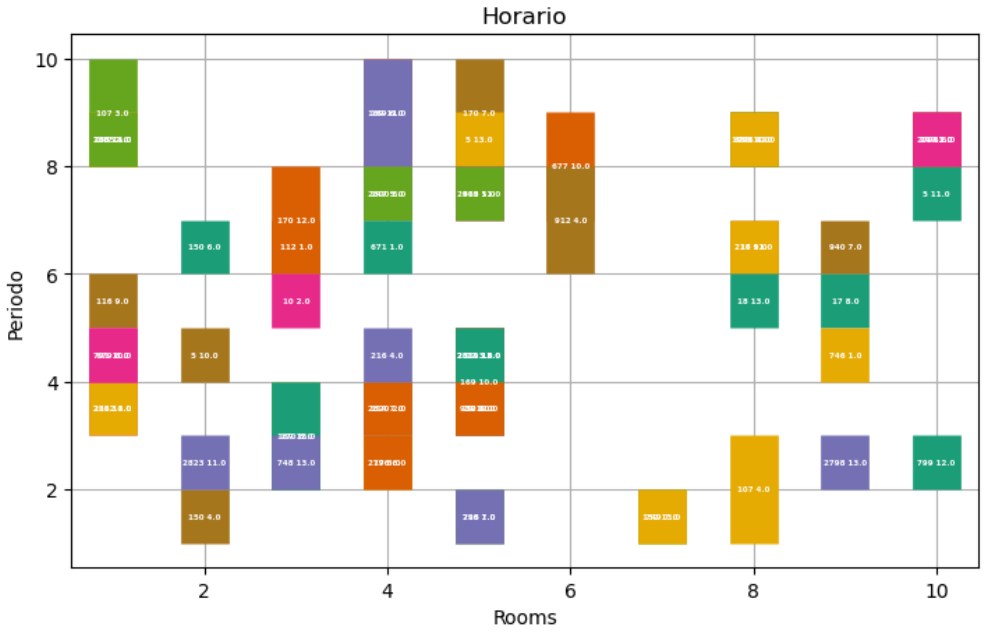
Comprobaciones de coherencia del modelo:

Se incorporaron varias comprobaciones de coherencia en el modelo para garantizar que las decisiones tomadas durante el proceso de programación fueran coherentes con las limitaciones definidas y las políticas institucionales. Cualquier infracción o inconsistencia se marcó para una mayor investigación. Se fue probando uno por uno las restricciones para comprobar su funcionalidad como por ejemplo:

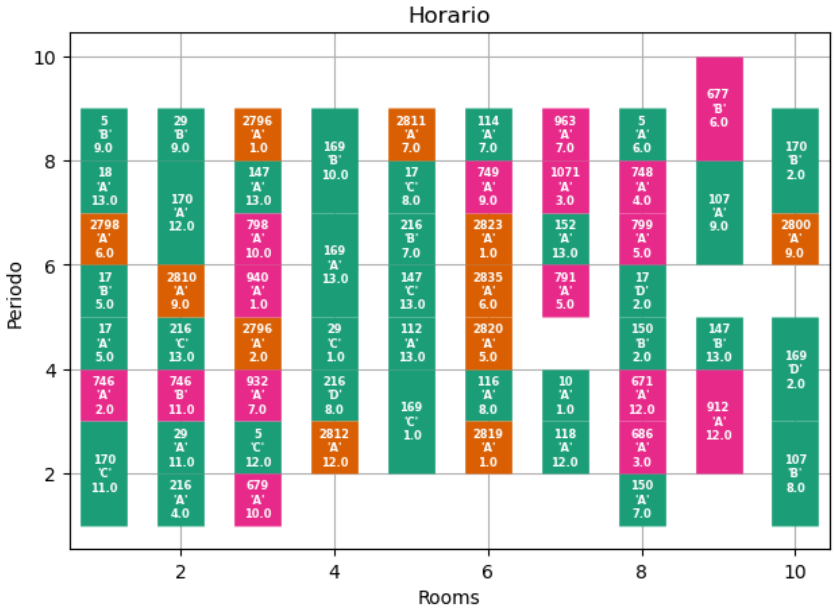
- Que no existieran traslapes de salones o de maestros en el mismo periodo.
- Que cada clase estuviera asignada a un maestro y salón.
- Que cada clase solo estuviera asignada a un periodo.

Aquí podemos ver resultados en donde las restricciones no se cumplían.

Traslape de cursos



Clases de más de un periodo fuera de la disponibilidad del salón.



Revisión de código:

El código de simulación se sometió a rigurosas revisiones por parte de un ingeniero en sistemas y expertos en el dominio. Este proceso de revisión ayudó a identificar y rectificar errores de codificación, inconsistencias lógicas y posibles problemas de modelado.

Validación de datos

Las entradas de datos, incluidos archivos CSV que contienen registros de programación históricos y otros datos relevantes, se validaron rigurosamente. Este proceso implicó comparar los datos importados (Sets y Data Frames) con las fuentes originales y verificar anomalías, valores faltantes o inconsistencias en los datos. Este proceso llegó a correcciones por culpa de inconsistencias con nombres de columnas.

Pruebas de condiciones de contorno:

El modelo de simulación se sometió a pruebas de condiciones límite para evaluar su comportamiento cuando se encuentran escenarios extremos o extremos. Esto ayuda a identificar posibles problemas relacionados con limitaciones de recursos, conflictos de programación u otros factores críticos. Principalmente se sometió a pruebas en donde se le daba poco tiempo para resolverse, gracias a eso se noto la inconsistencia de los resultados al momento de no contar con el tiempo necesario para la simulación.

Validacion

Comparación de datos históricos:

El resultado de la simulación se comparó con datos de programación históricos de períodos académicos anteriores. Esta comparación incluyó métricas como horarios de clases, asignaciones de maestros y reservas de aulas. La simulación se consideró válida si coincidía estrechamente con los registros históricos. Los datos se tomaron de el horario de ingenieria CUNOC del 2023 semestre impar.

En donde se intentó utilizar la mayor cantidad de datos iguales para la simulación, estos datos incluyen:

Mismos ingenieros y capacidades de su carrera

```
1,"Fransisco Rojas","Sistemas"
2,"Moises Granados","Sistemas"
3,"Pedro Domingo","Sistemas"
4,"Mauricio Lopez","Sistemas"
```

```
2796,"IPC1","Sistemas",3,true,1
2798,"Lenguajes De Programacion","Sistemas",3,true,1
2799,"Matematica de Computo 2","Sistemas",3,true,1
2800,"IPC2","Sistemas",4,false,1
2812,"Manejo E Implementacion de Archivos","Sistemas",6,true,1
2810,"Organizacion De Compiladores 2","Sistemas",6,true,1
```

```
2811,"Arquitectura De Compiladores1","Sistemas",6,true,1
2820,"Redes De Computadoras 2","Sistemas",8,true,1
2823,"Seminario De Sisteams 2","Sistemas",8,true,1
2819,"Sistemas Operativos 2","Sistemas",8,true,1
2835,"Modelacion y Simulacion 2","Sistemas",10,true,1
```

```
1,2835,false
1,2819,true
1,2823,false
2,2796,true
2,2798,false
2,2796,false
3,2823,true
3,2835,true
3,2819,false
3,2799,false
4,2796,false
4,2812,false
4,2811,true
```

Mismos salones y capacidades de estudiantes

```
id,nombre,capacidad_comoda,capacidad_maxima
1,"Salon 1",40,50
2,"Salon 2",40,50
3,"Salon 3",40,50
4,"Salon 4",60,70
5,"Salon 5",60,70
6,"Salon 6",20,25
7,"Salon 7",20,25
8,"Salon 8",30,40
9,"Salon 9",30,45
10,"Laboratorio de Aguas",30,45
```

Horarios comunes para la disponibilidad de los profesores

```
id_profesor,inicio,fin
1,"12:00","22:00"
2,"10:00","12:00"
```

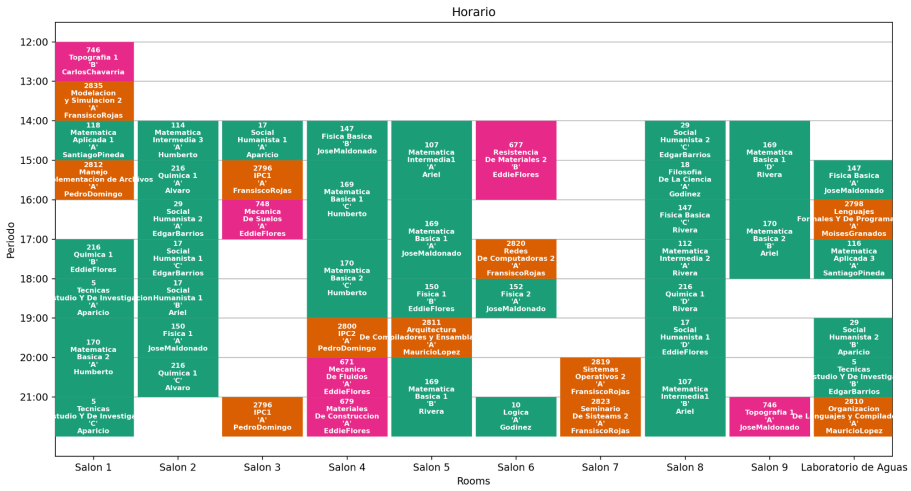
```
2,"14:00","20:00"
3,"14:00","22:00"
4,"14:00","22:00"
5,"14:00","22:00"
6,"14:00","22:00"
7,"8:00","14:00"
```

Misma cantidad de secciones y número de periodos para cursos de las carreras.

```
id,nombre,carrera,semestre,obli,no_periodos
169,"Matematica Basica 1","Comun",1,true,2
17,"Social Humanista 1","Comun",1,false,1
216,"Quimica 1","Comun",1,true,1
170,"Matematica Basica 2","Comun",2,true,2
29,"Social Humanista 2","Comun",2,true,1
5,"Tecnicas De Estudio","Comun",2,true,1
147,"Fisica Basica","Comun",2,true,1
107,"Matematica Intermedia1","Comun",3,true,2
150,"Fisica 1","Comun",3,true,1
112,"Matematica Intermedia 2","Comun",4,true,1
114,"Matematica Intermedia 3","Comun",4,true,1
```

```
id_materia,sec,numero_estudiantes
169,"A",60
169,"B",60
169,"C",55
169,"D",45
17,"A",50
17,"B",50
17,"C",40
17,"D",40
216,"A",50
216,"B",50
```

Finalmente se obtuvieron resultados muy similares a los del ejemplo.



Comentarios de las partes interesadas:

Se solicitaron comentarios de las partes interesadas clave, con respecto a los horarios de clases simulados. Siendo un estudiante de ingeniería y sabiendo mucho de los horarios de las carreras de diferentes años, valide los resultados de la programación frente a las expectativas del mundo real. Estos resultados para mi son muy apegados a la realidad y similares a horarios que he seguido en el pasado.

Análisis de sensibilidad:

El análisis de sensibilidad se realizó variando los parámetros de entrada, como el tamaño de las clases, la disponibilidad de los maestros y las capacidades del aula. Se analizaron las respuestas de la simulación a estas variaciones para garantizar que exhibiera un comportamiento razonable y esperado en diferentes escenarios.

EXPERIMENTACIÓN

Experimento semestre par normal

Objetivo:

Realizar la simulación sobre cursos de semestre para condiciones normales y recursos suficientes.

Description:

En este experimento, nuestro objetivo es evaluar cómo se asignan cursos de semestre par. La disponibilidad de docentes y de salones no serán un factor crítico ya que se ingresarán datos que representen disponibilidad suficiente según los cursos.

Parametros

Número de experimento: Experimento 1
Escenario de simulación: semestre par de 2023
Tiempo de simulación: 200 segundos
Carreras: 3 (Sistemas, Civil, Común)
Cursos: 42
Secciones: 62
Docentes: 15 - suficientes para cada carrera
Salones: 10 - suficientes para cada curso
Restricciones de disponibilidad: Ninguno
Datos: \schedule-sim\resources\scripts\1

Hipótesis:

- Asignación de todos los cursos, sin mayor problema ni conflictos.
- Encontrar la mejor distribución en un tiempo corto (menor a los 150 segundos).

Procedimiento

1. Seleccione un subconjunto de profesores y salones para experimentar, garantizando la diversidad en la experiencia en la materia y los horarios de clase preferidos.
2. Ejecutar el modelo de simulación para cada escenario, generando horarios de clases para el semestre de 2023.
3. Registre las siguientes métricas para cada escenario:

- Tasa de asignación de cursos
- Recuento de conflictos de programación
- Tiempo de simulación necesaria

Resultados y Análisis

Los resultados de esta experimentación se presentarán en secciones posteriores.

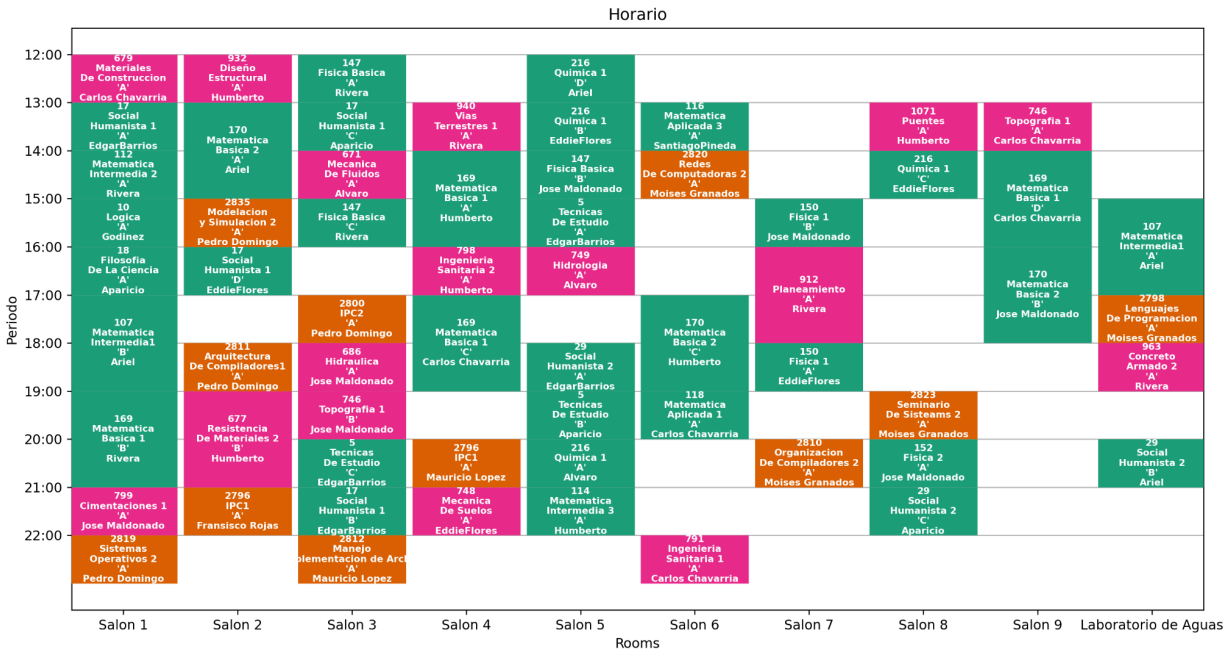
```
Result - Optimal solution found

Objective value:                -62.00000000
Enumerated nodes:               0
Total iterations:               14739
Time (CPU seconds):             75.13
Time (Wallclock seconds):       75.13

Total time (CPU seconds):       75.39   (Wallclock seconds):       75.39

Finish solver
Number of classes assigned = 62 out of 62:
Classes assigned:  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26,
 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
56, 57, 58, 59, 60, 61]
Number of Classes missed = 0 out of 62:
Classes missed:  []
Number of constraints = 3631
```

Los 62 cursos fueron exitosamente asignados en un tiempo de 75.39 segundos. Con una cantidad de iteraciones de 14739 con 3661 restricciones encontradas. Esto va acorde a nuestra hipótesis y para ver los resultados de manera más fácil aquí mostramos la gráfica. En donde podemos apreciar que todos las restricciones fueron cumplidas.



Tasa de asignación de cursos: 62/62
Recuento de conflictos de programación: 3661
Tiempo de simulación necesaria: 75.39 segundos

Experimento análisis de sensibilidad de docentes

Objetivo:

Evaluar la sensibilidad de la simulación de la programación del aula a las variaciones en las limitaciones de disponibilidad y capacidad de los docentes.

Description:

En este experimento, nuestro objetivo es evaluar cómo los cambios en las limitaciones de disponibilidad de docentes impactan los resultados de la programación. La disponibilidad de docentes es un factor crítico que puede influir en la eficiencia general de la programación y la utilización de recursos. Manipularemos las limitaciones de disponibilidad para un subconjunto de profesores y observaremos las asignaciones de clase y los conflictos de programación resultantes.

Parametros

Número de experimento: Experimento 1
Escenario de simulación: semestre par de 2023
Tiempo de simulación: 200 segundos
Carreras: 2 (Civil, Común)
Cursos: 23
Secciones: 31
Docentes: 10 - insuficiente para cada carrera
Salones: 10 - suficientes para cada curso
Restricciones de disponibilidad: Disponibilidad y capacidad de los docentes
Datos: \schedule-sim\resources\scripts\2

Hipótesis:

- Relajar las limitaciones de disponibilidad de docentes puede conducir a una mayor utilización de recursos, pero podría generar conflictos de programación.
- Restringir las limitaciones de disponibilidad de los docentes puede reducir los conflictos de programación, pero podría conducir a una menor utilización de los recursos.

Procedimiento

1. Seleccione un subconjunto de profesores y salones para experimentar, garantizando la diversidad en la experiencia en la materia y los horarios de clase preferidos.
2. Ajuste las restricciones de disponibilidad de los profesores seleccionados para crear escenarios con distintos grados de flexibilidad y restricción.
3. Ejecutar el modelo de simulación para cada escenario, generando horarios de clases para el semestre de 2023.
4. Registre las siguientes métricas para cada escenario:
    - Tasa de asignación de cursos
    - Recuento de conflictos de programación
    - Tiempo de simulación necesaria

Resultados y Análisis

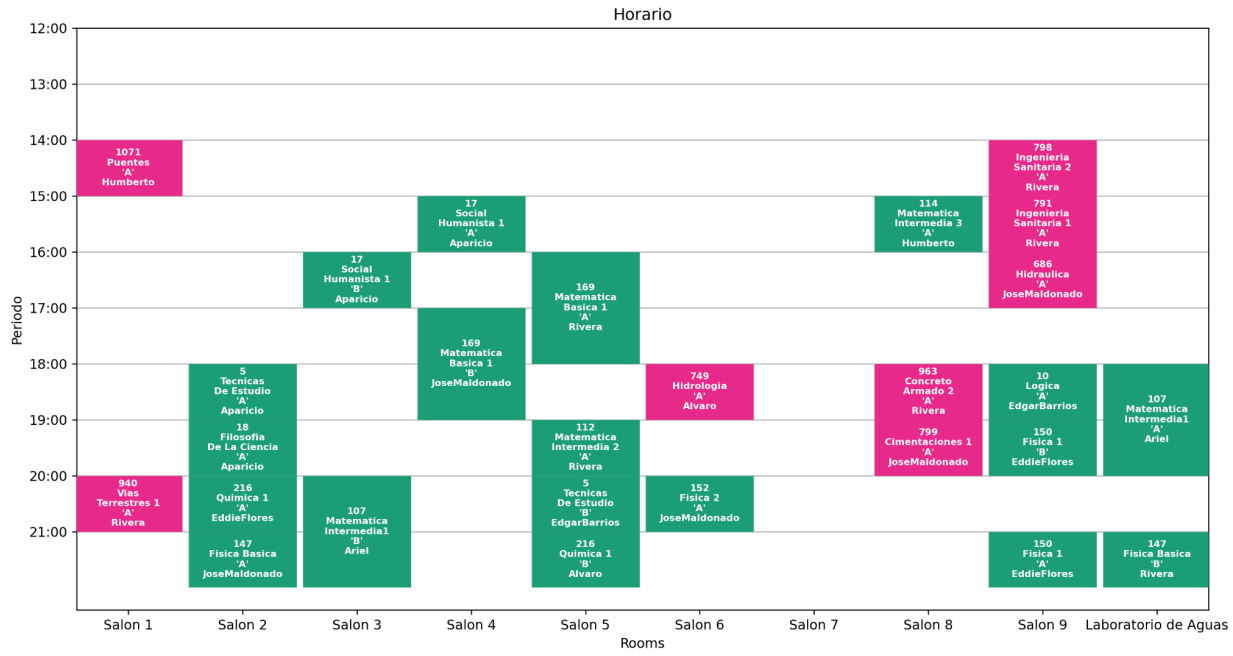
Los resultados de esta experimentación se presentarán en secciones posteriores, proporcionando información sobre cómo las limitaciones de disponibilidad de los docentes influyen en la programación del aula y en la experiencia educativa general dentro de la institución.

```
Objective value:                -27.00000000
Enumerated nodes:               1
Total iterations:               3233
Time (CPU seconds):             4.04
Time (Wallclock seconds):       4.04

Total time (CPU seconds):       4.13   (Wallclock seconds):       4.13

Finish solver
Number of classes assigned = 27 out of 31:
Classes assigned:  [0, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 26, 27, 28, 2
9]
Number of Classes missed = 4 out of 31:
Classes missed:  [6, 7, 25, 30]
Number of constraints = 961
```

Los 31 cursos no fueron asignados. Pero se encontró una solución asignando 27 cursos en un tiempo de 4.13 segundos. Con una cantidad de iteraciones de 3233 con 961 restricciones encontradas. Esto gracias a que se redujo la cantidad de clases de la simulación. Esto va acorde a nuestra hipótesis y para ver los resultados de manera más fácil aquí mostramos la gráfica. En donde podemos apreciar que todos las restricciones fueron cumplidas.



Tasa de asignación de cursos: 27/31
Recuento de conflictos de programación: 961
Tiempo de simulación necesaria: 4.13 segundos

Experimento análisis de sensibilidad de salones

Objetivo:

Evaluar la sensibilidad de la simulación de la programación del aula a las variaciones en las limitaciones de disponibilidad y capacidad de los salones.

Description:

En este experimento, nuestro objetivo es evaluar cómo los cambios en las limitaciones de disponibilidad de salones impactan los resultados de la programación. La disponibilidad de salones es un factor crítico que puede influir en la eficiencia general de la programación y la utilización de recursos. Manipularemos las limitaciones de disponibilidad para un subconjunto de salones y observaremos las asignaciones de clase y los conflictos de programación resultantes.

Parametros

Número de experimento: Experimento 1
Escenario de simulación: semestre par de 2023
Tiempo de simulación: 200 segundos
Carreras: 3 (Sistemas, Civil, Común)
Cursos: 42
Secciones: 62
Docentes: 15 - suficientes para cada carrera
Salones: 7 - insuficiente para cada curso
Restricciones de disponibilidad: Disponibilidad y capacidad de los salones
Datos: \schedule-sim\resources\scripts\3

Hipótesis:

- Relajar las limitaciones de disponibilidad de salones puede conducir a una mayor utilización de recursos, pero podría generar conflictos de programación.
- Restringir las limitaciones de disponibilidad de los salones puede reducir los conflictos de programación, pero podría conducir a una menor utilización de los recursos.

Procedimiento

1. Seleccione un subconjunto de profesores y salones para experimentar, garantizando la diversidad en la experiencia en la materia y los horarios de clase preferidos.
2. Ajuste las restricciones de disponibilidad de los salones seleccionados para crear escenarios con distintos grados de flexibilidad y restricción.
3. Ejecutar el modelo de simulación para cada escenario, generando horarios de clases para el semestre de 2023.
4. Registre las siguientes métricas para cada escenario:
    - Tasa de asignación de cursos
    - Recuento de conflictos de programación
    - Tiempo de simulación necesaria

Resultados y Análisis

Los resultados de esta experimentación se presentarán en secciones posteriores, proporcionando información sobre cómo las limitaciones de disponibilidad de los salones influyen en la programación del aula y en la experiencia educativa general dentro de la institución.

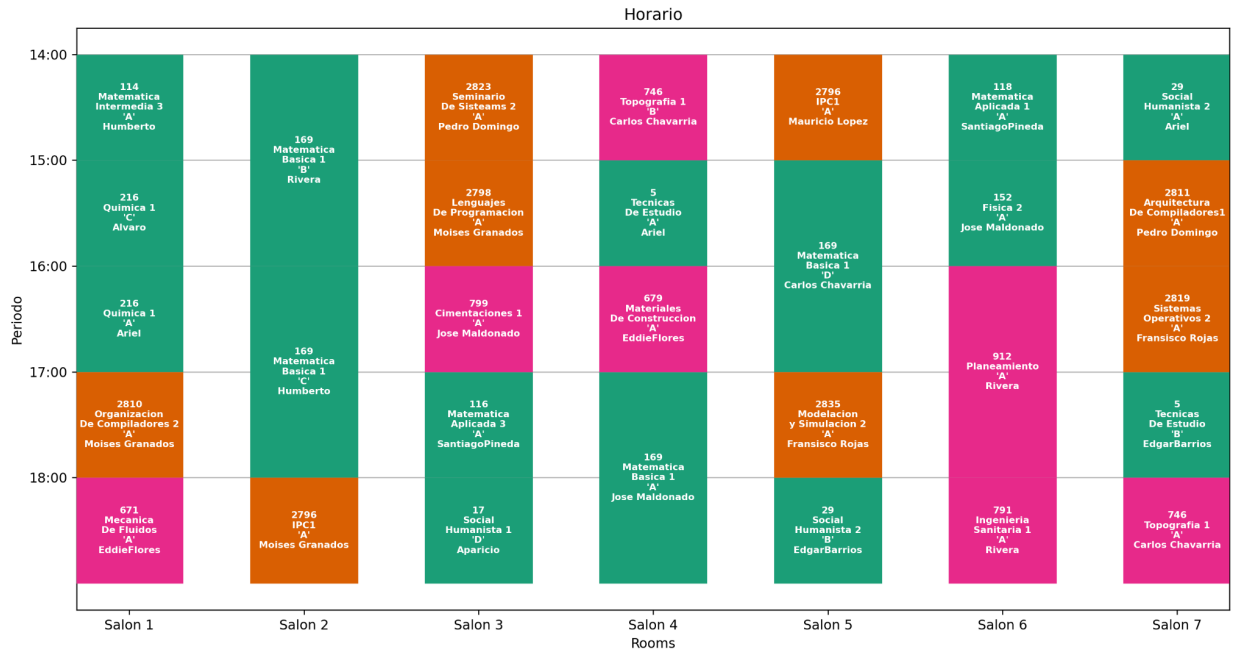
```
Result - Stopped on time limit

Objective value:                -30.00000000
Lower bound:                    -35.000
Gap:                            -0.14
Enumerated nodes:               43735
Total iterations:               383586
Time (CPU seconds):             201.56
Time (Wallclock seconds):       201.56

Total time (CPU seconds):       201.69   (Wallclock seconds):       201.69

WARNING: Loading a SolverResults object with an 'aborted' status, but
    containing a solution
Finish solver
Number of classes assigned = 30 out of 62:
Classes assigned:  [0, 1, 2, 3, 7, 8, 10, 15, 16, 18, 19, 29, 31, 32, 33, 35, 36, 37, 40, 41, 43, 44, 45, 46, 47, 48,
 51, 55, 58, 61]
Number of Classes missed = 32 out of 62:
Classes missed:  [4, 5, 6, 9, 11, 12, 13, 14, 17, 20, 21, 22, 23, 24, 25, 26, 27, 28, 30, 34, 38, 39, 42, 49, 50, 52,
 53, 54, 56, 57, 59, 60]
Number of constraints = 1584
```

Los 62 cursos no fueron asignados. Pero se encontró una solución asignando 30 cursos en un tiempo de 201.69 segundos utilizando cada segundo del tiempo establecido. Se encontró con una cantidad de iteraciones de 383586 con 1584 restricciones encontradas. Esto va acorde a nuestra hipótesis aunque si se tardó más de lo esperado y para ver los resultados de manera más fácil aquí mostramos la gráfica. En donde podemos apreciar que todos las restricciones fueron cumplidas.



Como se puede observar no existe espacio para ningún otro curso, priorizando los que son obligatorios.

INTERPRETACIÓN

Resumen de resultados:

Los experimentos de análisis de sensibilidad se centraron en ajustar las limitaciones de disponibilidad de los docentes y salones para explorar su impacto en la programación del aula. Realizamos múltiples escenarios para evaluar la utilización de recursos, los conflictos de programación, la distribución de la carga de trabajo de los maestros y la satisfacción de los estudiantes.

Utilización de recursos:

Como era de esperar, al no modificar las limitaciones de disponibilidad y capacidad de docentes y salones generalmente condujo a tasas más altas de utilización de recursos. Esto se atribuye a la mayor flexibilidad en la programación de clases.
Por el contrario, las limitaciones de disponibilidad de docentes dieron como resultado una menor utilización de recursos debido a las limitadas opciones de programación.

Conflictos de programación:

La flexibilización de las restricciones de disponibilidad a menudo introdujo más conflictos de programación, incluidas reservas dobles en las aulas y horarios superpuestos de los docentes.
El endurecimiento de las restricciones de disponibilidad redujo los conflictos de programación y mejoró la confiabilidad de la programación.

Tiempo de Simulación

Basados en los resultados podemos ver que el tiempo de ejecución de la simulación depende de la facilidad de encontrar el mejor resultado, esto depende de las disponibilidades de los recursos y las restricciones de asignaciones obligatorias.

CONCLUSIONES

La simulación de programación en el aula ha proporcionado información valiosa sobre el proceso de programación dentro de nuestra institución de educación superior (CUNOC). Ha resaltado las complejidades e interdependencias involucradas en la optimización de las asignaciones en el aula, la asignación de maestros, salones y la utilización de recursos. De esta simulación han surgido varias consideraciones importantes.

La sensibilidad de los resultados de la programación a las limitaciones de disponibilidad de recursos subraya la necesidad de un enfoque equilibrado que optimice el uso de los recursos y al mismo tiempo gestione los conflictos de programación.

La utilización de recursos y la confiabilidad de la programación presentan una compensación, y las decisiones deben sopesar cuidadosamente los beneficios de maximizar los recursos frente a las posibles interrupciones causadas por los conflictos.

El modelo de simulación puede servir como una valiosa herramienta de apoyo a las decisiones para los administradores educativos, ayudando en la toma de decisiones informadas relacionadas con las políticas de asignación, las asignaciones de los docentes y la asignación de aulas.

RECOMENDACIONES

Con base en los resultados de la experimentación, se proponen las siguientes recomendaciones:

Flexibilidad y optimización: considere un enfoque equilibrado de las limitaciones de disponibilidad de los docentes que proporcione flexibilidad para una utilización óptima de los recursos y al mismo tiempo gestione los conflictos de programación.

Programación de resolución de conflictos: implementar mecanismos sólidos de resolución de conflictos para abordar los conflictos que puedan surgir cuando se relajen las restricciones.

Programación centrada en el estudiante: esforzarse por brindar a los estudiantes opciones para los horarios de clases, garantizando al mismo tiempo la confiabilidad y minimizando las interrupciones.

Revisión periódica: monitorear y revisar continuamente las limitaciones de disponibilidad de docentes para adaptarse a las necesidades y preferencias institucionales cambiantes.

Los hallazgos de este análisis de sensibilidad brindan información valiosa para los tomadores de decisiones a la hora de optimizar el proceso de programación del aula y, en última instancia, mejorar la experiencia educativa dentro de la institución.

En este ejemplo, la interpretación de los resultados de la simulación incluye una descripción general de los hallazgos clave, las conclusiones extraídas de los experimentos y las recomendaciones basadas en los resultados. Proporciona una comprensión clara de cómo las variaciones en las limitaciones de disponibilidad de los docentes afectan el sistema de programación del aula y ofrece información útil para la toma de decisiones.