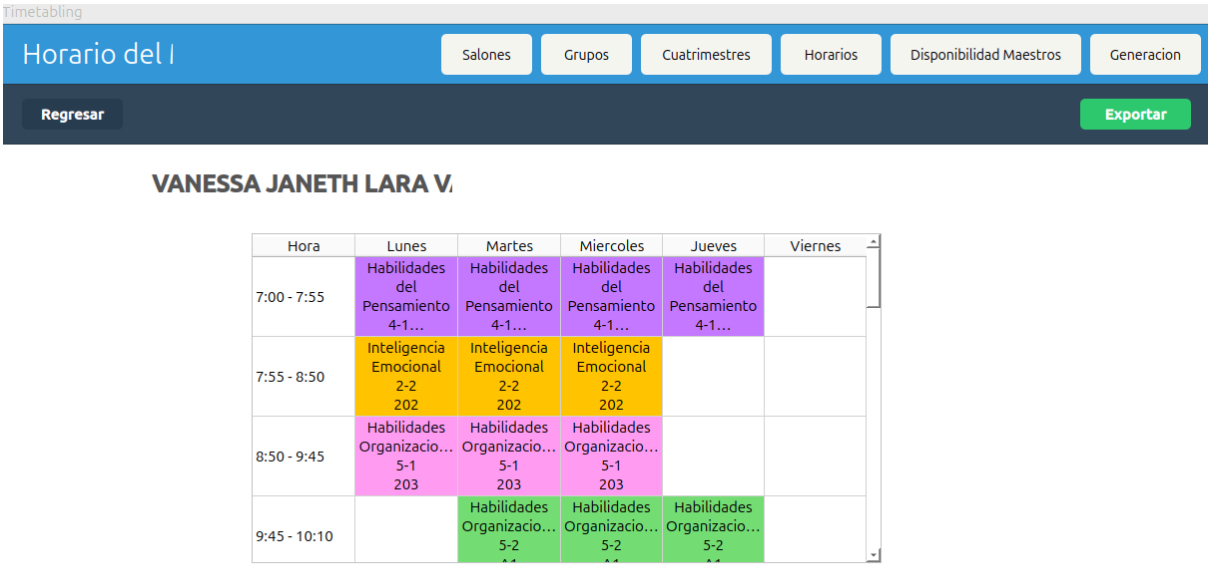
Manual Técnico

Universal TimeTabling, Resolución planteada de asignación de horarios escolares de una universidad.



Integrantes

Carlos Gael Guerrero Salazar 2430481

Hector Abad Contreras Muñiz 2430033

Efrain Abdiel Villanueva Balboa 2430252

Juan Francisco Ortega Pulido 2430450

1. Universal Timetabling

¿Qué es Universal Timetabling?

El Universal Timetabling, también conocido como "Comprehensive University Timetabling System", es un problema computacional de optimización combinatoria que consiste en la asignación eficiente de recursos educativos dentro de restricciones de tiempo y espacio. Este problema se clasifica como NP-difícil (NP-hard), lo que significa que no existe un algoritmo conocido que pueda encontrar la solución óptima de manera rápida para instancias de gran escala.

El problema fundamental radica en insertar recursos (maestros, materias, salones) y asignar situaciones educativas (clases) dentro de un número limitado de espacios temporales y físicos, buscando satisfacer las necesidades institucionales de la mejor forma posible. La complejidad del problema crece exponencialmente a medida que se incrementa el número de variables involucradas: más maestros, materias, grupos o salones significan un crecimiento dramático en el tiempo de procesamiento necesario para encontrar una solución válida.

Aplicación en Universidades y Escuelas

En el contexto educativo universitario y escolar, el Universal Timetabling aborda múltiples dimensiones de planificación académica:

Gestión de Recursos Humanos: Los maestros representan el recurso más crítico del sistema. Cada docente posee horarios de disponibilidad específicos determinados por sus obligaciones profesionales externas, preferencias personales y contratos laborales. El sistema debe respetar estas restricciones mientras maximiza la utilización de su tiempo disponible para la enseñanza.

Coordinación de Espacios Físicos: Los salones de clase presentan limitaciones de capacidad y disponibilidad. El sistema debe asignar grupos de estudiantes a salones con capacidad adecuada, considerando las características específicas de cada materia (laboratorios, aulas especializadas, auditorios).

Organización Estudiantil: Los alumnos se agrupan en cohortes que comparten trayectorias académicas similares. El sistema debe generar horarios que minimicen conflictos entre estudiantes del mismo grupo, mientras permite flexibilidad para casos especiales como estudiantes en recursamiento o con trayectorias académicas no convencionales.

Distribución Temporal Óptima: El objetivo es evitar "colisiones" donde un mismo recurso (maestro, alumno o salón) sea requerido simultáneamente en múltiples ubicaciones. Además, se busca minimizar fragmentaciones en los horarios, como huecos excesivos entre clases o distribuciones ineficientes que obliguen a desplazamientos innecesarios.

Complejidad Computacional del Problema

Naturaleza NP-Difícil del University Timetabling Problem

El problema de asignación de horarios universitarios está clasificado formalmente como NP-difícil (NP-hard), una categoría de problemas computacionales que representa algunos de los desafíos más complejos en ciencias de la computación. Esta clasificación tiene implicaciones profundas para cualquier intento de resolución computacional.

¿Qué significa NP-Difícil en este contexto?

Un problema NP-difícil se caracteriza por:

1. **No existe algoritmo polinomial conocido**: No se ha encontrado (y muy probablemente no existe) un algoritmo que pueda garantizar encontrar la solución óptima en tiempo polinomial respecto al tamaño de la entrada.
2. **Crecimiento exponencial del espacio de búsqueda**: A medida que aumenta el número de variables (maestros, materias, grupos, salones), el número de posibles configuraciones de horarios crece de forma exponencial, no lineal.
3. **Verificación vs Construcción**: Aunque verificar si una solución propuesta es válida puede hacerse eficientemente (en tiempo polinomial), construir esa solución desde cero requiere explorar un espacio de búsqueda masivo.

Implicaciones Prácticas del Tiempo de Ejecución

La complejidad exponencial se manifiesta en tiempos de ejecución reales:

- **Búsqueda exhaustiva (Brute Force)**: Para el escenario medio, suponiendo que pudiéramos evaluar 1 billón de configuraciones por segundo, tomaría más tiempo que la edad del universo encontrar la solución óptima mediante búsqueda exhaustiva.
- **Con Backtracking Simple**: El tiempo puede reducirse significativamente al eliminar ramas inválidas tempranamente, pero para casos complejos, el tiempo de ejecución puede seguir siendo de horas o días.
- **Con Backtracking + Heurísticas**: Como en nuestra propuesta, el tiempo se reduce a minutos o segundos para casos medios, aunque no se garantiza encontrar la solución óptima global, sino una solución válida y de buena calidad.

Justificación de Herramientas y Tecnologías

Por qué C++ para este Problema

La elección de C + + como lenguaje de implementación para nuestro sistema de Universal Timetabling no es arbitraria; responde a requisitos técnicos específicos del problema que C + + satisface mejor que alternativas populares.

Razones Técnicas Fundamentales:

1. Rendimiento Computacional Crítico

El problema de timetabling, siendo NP-difícil, requiere explorar potencialmente millones de configuraciones. Cada microsegundo ahorrado en operaciones individuales se multiplica por millones de iteraciones:

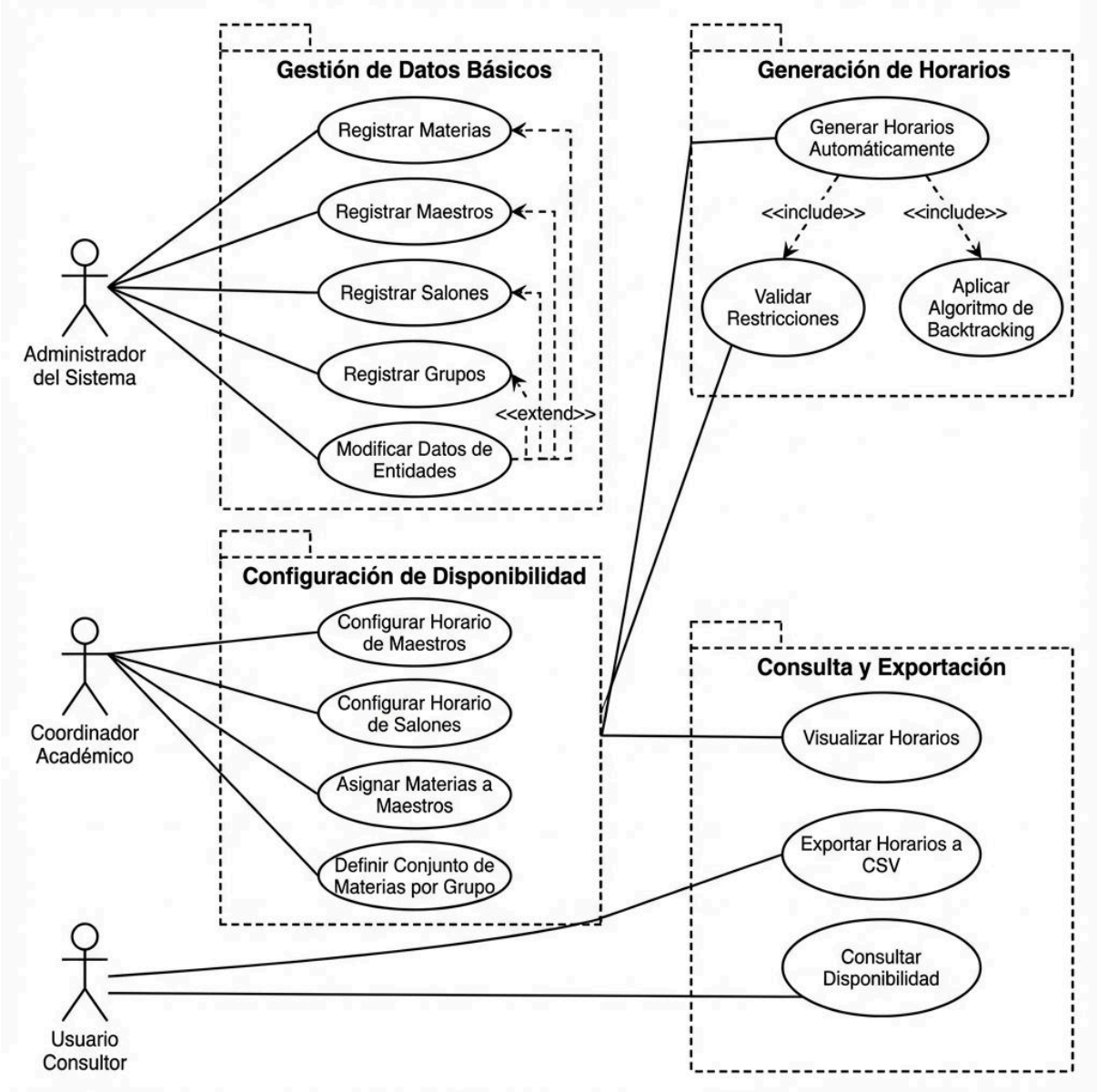
Ejecución Nativa: C + + se compila a código máquina nativo, sin intermediarios como máquinas virtuales o intérpretes , resultando en 10-100x más rápido para operaciones intensivas.

Control de Memoria Manual: La gestión manual de memoria permite optimizaciones imposibles en lenguajes con recolección de basura automática. Para estructuras como grafos con miles de nodos, esto es crucial.

Como lo propusimos

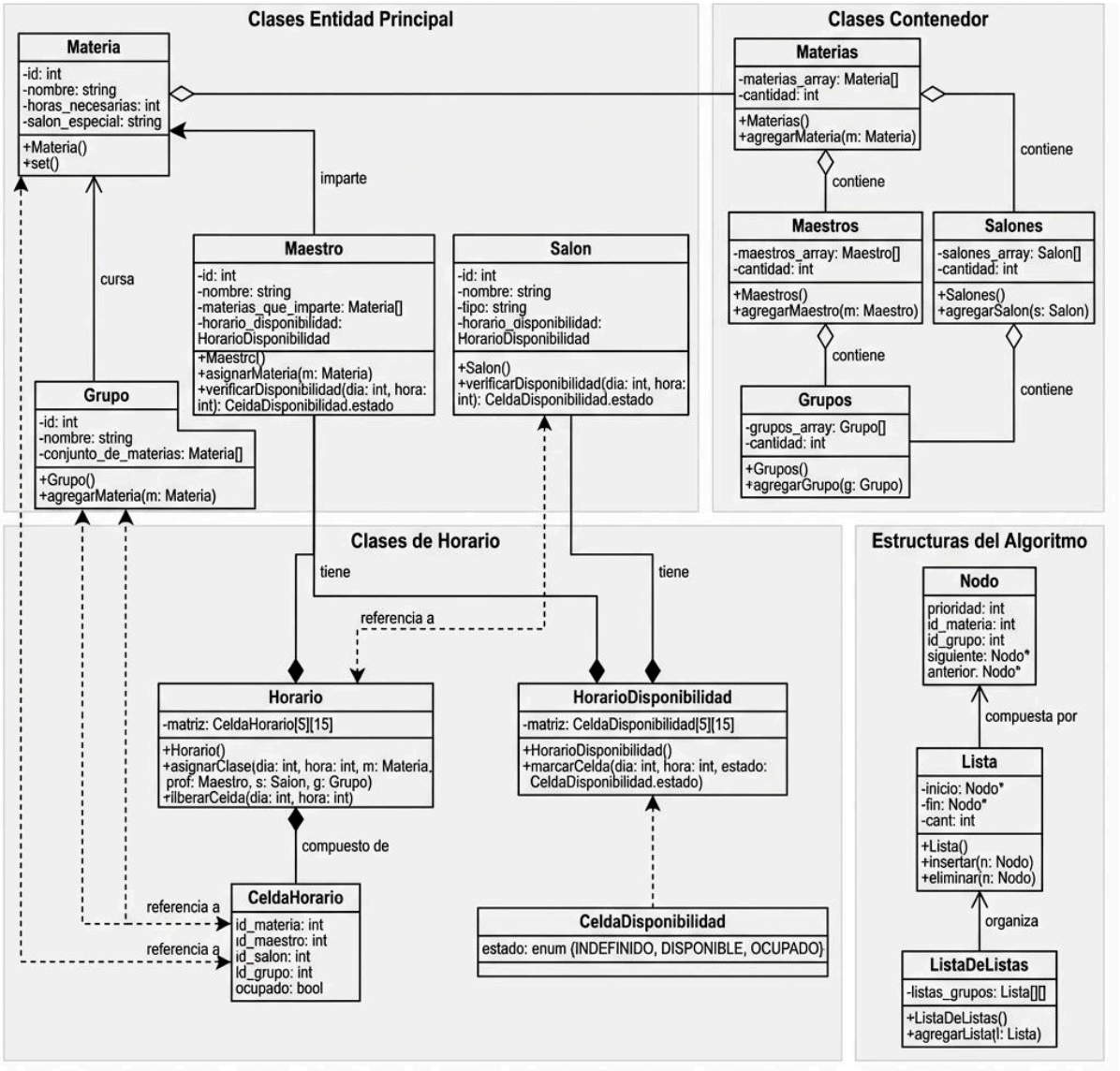
Diagramas

1. **Diagrama de caso de uso.**



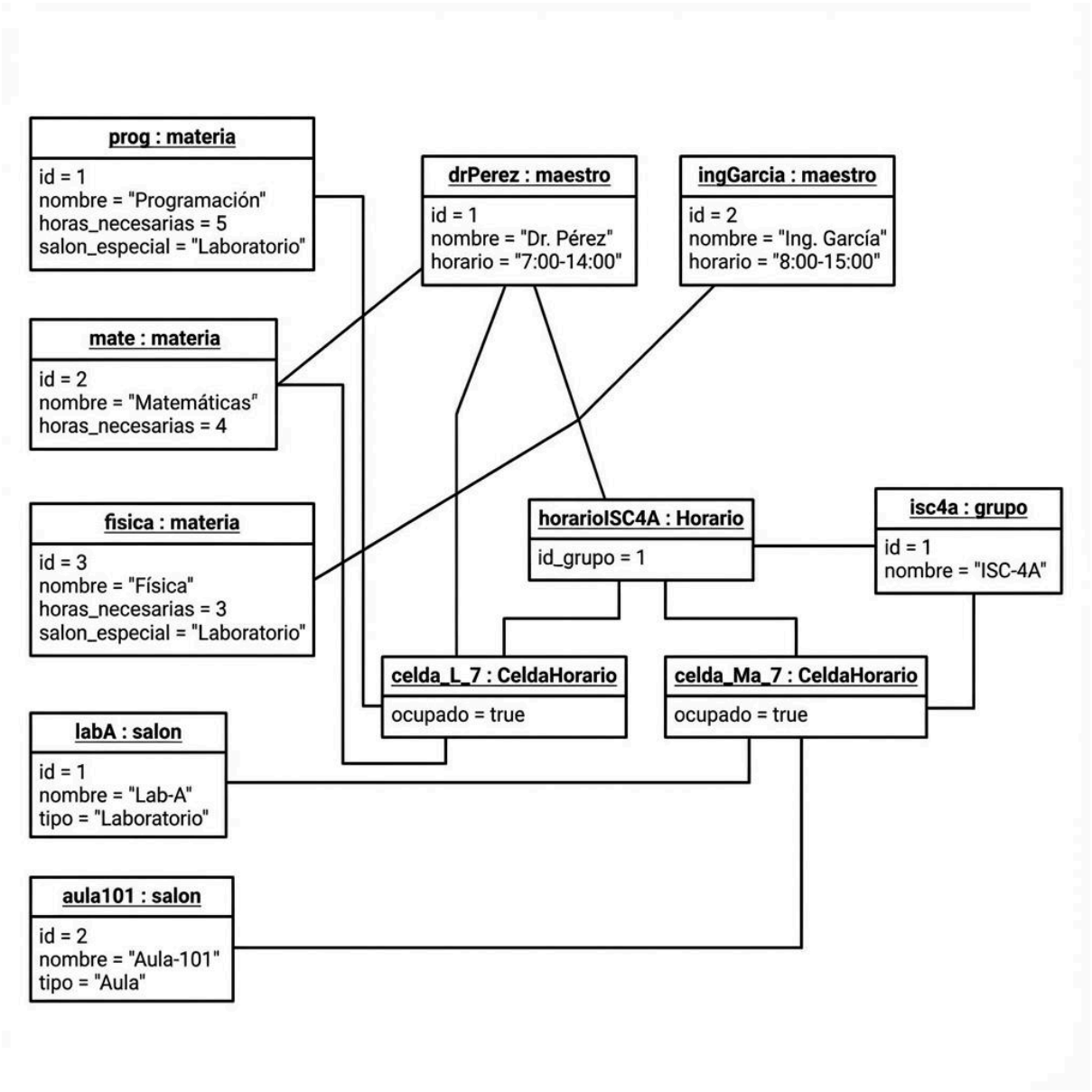
El sistema define 3 actores (Administrador, Coordinador Académico, Usuario Consultor) y 15 casos de uso en 4 paquetes: Gestión de Datos Básicos (registrar/modificar entidades), Configuración de Disponibilidad (horarios y asignaciones), Generación de Horarios (algoritmo automático con validación y backtracking), y Consulta/Exportación (visualización y exportación a CSV).

2. Diagrama de clases



El diagrama muestra 13 clases organizadas en cuatro grupos: entidades principales (materia, maestro, salon, grupo), gestión de horarios (Horario, horario_disponibilidad con matrices 5×15), contenedores dinámicos (materias, maestros, salones, grupos), y estructuras del algoritmo de backtracking (nodo, lista, lista_de_listas). Las relaciones incluyen composición (Horario contiene CeldaHorario), agregación (contenedores), asociación (maestro imparte materia) y dependencia (CeldaHorario referencia entidades).

3. Diagrama de Objetos



El diagrama ilustra un ejemplo concreto con 3 materias, 2 maestros, 2 salones y 1 grupo, mostrando cómo el sistema asigna clases considerando disponibilidad de maestros (7:00-14:00), requerimientos de salón (laboratorios para Programación y Física), y distribución temporal. Cada CeldaHorario mantiene referencias completas a materia, maestro, salón y grupo.

Secuencia principal:

El sistema genera todos los objetos en base a documentos csv, estos documentos contienen una estructura que es legible para el humano, esta estructura permite distinguir entre las diferentes cosas que componen a los objetos anteriormente mostrados.

Una vez el sistema genera los documentos se da la oportunidad de modificación a los datos del sistema con guardado directo, es decir que en caso de cambiar los datos el sistema los cambia en el sistema de csv.

Una vez la persona ya acepta todo lo propuesto comienza la asignación desde el backtracking basándose en las siguientes condiciones.

Condiciones del Backtracking:

1. Compatibilidad con el salón esperado.
2. Que el grupo esté disponible a esa hora.
3. Se prohíbe la selección de maestros que estén ocupados o no esten en ese momento.
4. Verifica que el salón esté disponible.
5. No se permite el mismo maestro en el mismo grupo.
6. Se verifica si el maestro imparte la materia seleccionada.
7. El backtracking no debe ser infinito.

Proceso de asignación y prioridad

El sistema **agarra todos grupos y les da una prioridad** en cuestion de parametros establecidos por nosotros, entre los cuales se destaca:

- La cantidad de horas totales que existen entre las diferentes personas que pueden llevar la materia.
- Cantidad de horas que se necesitan de la materia en comparación del resto de las materias del grupo.
- Promedio de horas en cantidad que los profesores pueden tomar la clase.

Estas condiciones permiten asignar una prioridad alta a clases que pueden afectar mucho el resultado en cuestión del tiempo que tomarán, ya que cada vez que esta prioridad sea definida se ordenan en una fila de nodos, permitiendo apuntar al siguiente y volver en reversa en las acciones que se escogen.

Al tener la selección se lleva toda la lista a estructurarse al horario, ya que utilizan estructuras diferentes y su función es volverse a ordenar una vez eso se ejecuta el sistema de universal Timetabling utilizando la lista completa y ordenadas de nodos y materias.

Proceso del Backtracking

El sistema de asignación de horarios utiliza un algoritmo de backtracking recursivo para resolver el problema de asignación de materias a grupos, maestros y salones. Este es un problema de satisfacción de restricciones (CSP - Constraint Satisfaction Problem) donde se deben cumplir múltiples condiciones simultáneamente.

Se implementaron dos modos para asignar de manera horizontal las horas, en nuestro caso intentamos dar una clase por día a los grupos, esto se consigue mediante el modo estricto y en caso de que el sistema detecte que las horas necesarias son más de 5 o el intentar asignar por modo estricto falla se aplica el modo relajado.

Modo Estricto: Intenta asignar todas las horas de la materia en la **misma franja horaria** a lo largo de la semana (horizontalmente).
Modo Relajado: Permite asignar las horas en **cualquier franja disponible**, sin restricción de horizontalidad.

Que pasa en caso de no cumplir las condiciones:
Si ningún maestro candidato pudo asignar la materia (ni en modo estricto ni relajado), el backtracking retorna false, lo que provoca que el nodo anterior deshaga su asignación y pruebe con otro maestro.

En caso de que todos los maestro fallen se busca a otro maestro que esté disponible para la asignación, si ningún maestro no está disponible para esa asignación busca a aquellos maestros que pueden ser utilizados, intentando cambiar su horas ocupadas por los de otro maestro que pueda ser validado y por ende ocupar sus horas.

En caso de que eso no sea posible se vuelve una asignación atrás para cambiar de maestro, esto se repite hasta que el sistema encuentre la solución a todos los posibles escenarios de una manera más rápida en comparación de volver hacia atrás.

Guardado y demostración de los resultados

Cuando se completa el sistema se agarran los horarios obtenidos y se transcriben de una manera legible a archivos csv, estos archivos son diferentes a los de los maestro, ya que uno almacena el horario del maestro en general y el otro señala las clases que tendrá que hacer a lo largo de la semana, esto pasa también para tanto grupos como salones.

Conclusión del proceso:
El sistema de backtracking implementado es un algoritmo **exhaustivo y robusto** que:

1. Prioriza las materias con más restricciones.
2. Valida condiciones especificadas para que un agrupamiento sea permitido.
3. Retrocede si el sistema detecta que no se puede resolver.
4. Prueba otras configuraciones para el sistema.
5. Garantiza una solución para todos los grupos al menos que la agrupación sea muy complicada.

Generación de páginas:

Para poder hacer posible el uso del sistema para cualquier persona no especializada en tecnologías, era necesario el desarrollo de una interfaz gráfica. Para ello, se concluyó usar Qt como framework de desarrollo.

¿Qué es Qt?

Qt es un framework de desarrollo de aplicaciones multiplataforma para desktop, embebido y móvil. Es un entorno completo que permite crear interfaces de manera sencilla, robusta y eficiente.

Características principales de Qt:

- Multiplataforma: El mismo código funciona en Windows, Linux, macOS, Android e iOS
- Orientado a objetos: Utiliza C++ con extensiones propias (signals/slots)
- Qt Designer: Herramienta visual para diseñar interfaces gráficas mediante arrastrar y soltar
- Widgets: Componentes reutilizables de interfaz (botones, tablas, cuadros de texto, etc.)
- Sistema de señales y slots: Mecanismo de comunicación entre objetos sin acoplamiento directo

Estructura del Proyecto Qt

El sistema de horarios utiliza la arquitectura estándar de Qt con separación clara entre presentación, lógica y estructura de datos.

Organización de Directorios

```
timetabling/
 ├── forms/ # Archivos de interfaz (.ui)
 ├── include/ # Archivos de cabecera (.h)
 ├── src/ # Archivos de implementación (.cpp)
 ├── build/ # Archivos compilados
 └── CMakeLists.txt # Configuración de compilación
```

Componentes de la Interfaz

Se componen de 3 tipos de extensiones para realizar la interfaz de este sistema:

1. Archivos .ui (forms/archivowidget.ui)

Descripción:

Es la estructura principal de la interfaz, se realiza gracias al lenguaje de marcado XML.

Función:

Define la disposición visual de los elementos (layout), sus propiedades y estilos CSS.

Ejemplo de estructura (groupdetailswidget.ui):

```
<?xml version="1.0" encoding="UTF-8"?>
 <ui version="4.0">
 <class>GroupDetailsWidget</class>
 <widget class="QWidget" name="GroupDetailsWidget">
 <property name="geometry">
 <rect>
 <x>0</x>
 <y>0</y>
 <width>1231</width>
 <height>551</height>
 </rect>
 </property>
 <property name="styleSheet">
 <string notr="true">background-color: white;</string>
 </property>
 <widget class="QPushButton" name="backButton">
 <!-- Definición del botón -->
 </widget>
 </widget>
 </ui>
```

Características:

- Generado automáticamente por Qt Designer
- Define widgets como botones (QPushButton), tablas (QTableWidget), combos (QComboBox)
- Establece propiedades visuales (colores, tamaños, fuentes)
- Especifica el layout (distribución espacial de los elementos)

2. Archivos .h (include/archivowidget.h)

Descripción:

Es la estructura base de la lógica del programa (archivo de cabecera).

Función:

Declara la clase del widget, sus métodos públicos/privados, slots, señales y atributos.

Ejemplo de estructura (groupdetailswidget.h):

```
#ifndef GROUPDETAILSWIDGET_H
 #define GROUPDETAILSWIDGET_H

 #include <QWidget>
```

```
namespace Ui {
class GroupDetailsWidget;
}

class GroupDetailsWidget : public QWidget
{
Q_OBJECT // Macro necesaria para signals/slots

public:
explicit GroupDetailsWidget(QWidget *parent = nullptr);
~GroupDetailsWidget();

// Métodos públicos
void setData(grupos* g, materias* m, maestros* ma, ConjuntoMaterias* c);
void cargarGrupoForEdit(int id);
void prepararNuevoGrupo();

protected:
bool eventFilter(QObject* obj, QEvent* event) override;

private slots:
// Slots para eventos de UI
void on_createButton_clicked();
void on_backButton_clicked();
void on_deleteButton_clicked();

private:
Ui::GroupDetailsWidget *ui;

// Punteros a estructuras de datos
grupos* m_grupos = nullptr;
materias* m_materias = nullptr;
maestros* m_maestros = nullptr;
ConjuntoMaterias* m_conjuntos = nullptr;

int m_currentGroupId = -1;
QColor m_selectedColor;

signals:
// Señales que emite el widget
void backClicked();
void applyClicked();
void finalizado();
};

#endif
```

Elementos clave:

- Q_OBJECT: Macro obligatoria para usar el sistema de señales y slots
- Constructor/Destructor: Inicialización y limpieza de recursos
- Métodos públicos: API externa del widget
- Slots privados: Responden a eventos de la interfaz (clicks, cambios de texto, etc.)
- Signals: Notificaciones que emite el widget hacia otros componentes
- Atributos privados: Estado interno del widget

3. Archivos .cpp (src/archivowidget.cpp)

Descripción:

Es la lógica de la interfaz, aquí se encuentran los eventos de los elementos y otras funciones.

Función:

Implementa los métodos declarados en el .h, gestiona la lógica de negocio y la interacción con los datos.

Ejemplo de estructura (groupdetailswidget.cpp):

```
#include "groupdetailswidget.h"
 #include "ui_groupdetailswidget.h"
 #include "../include/grupos.h"
 #include "../include/materias.h"
 #include <QMessageBox>
 #include <QColorDialog>

 // Constructor
 GroupDetailsWidget::GroupDetailsWidget(QWidget *parent) :
 QWidget(parent),
 ui(new Ui::GroupDetailsWidget)
 {
 ui->setupUi(this); // Inicializa la interfaz del .ui

 // Configuración inicial
 ui->colorPicker->installEventFilter(this);
 m_selectedColor = QColor(255, 0, 0);
 }

 // Destructor
 GroupDetailsWidget::~GroupDetailsWidget()
 {
 delete ui;
 }

 // Método para establecer referencias a datos
 void GroupDetailsWidget::setData(grupos* g, materias* m,
 maestros* ma, ConjuntoMaterias* c)
 {
 m_grupos = g;
 m_materias = m;
 m_maestros = ma;
 m_conjuntos = c;

 // Llenar combobox con datos
 ui->quarterComboBox->clear();
 if (m_conjuntos) {
 for(int i=0; i<m_conjuntos->getCantidad(); i++) {
 Set* s = m_conjuntos->getSet(i);
 if (s) {
 ui->quarterComboBox->addItem(
 QString::fromStdString(s->nombre), i);
 }
```

```
}
}
}

// Slot para manejar el click del botón crear
void GroupDetailsWidget::on_createButton_clicked()
{
// Validar datos
if (ui->nameLineEdit->text().isEmpty()) {
QMessageBox::warning(this, "Error",
"El nombre del grupo no puede estar vacío");
return;
}

// Crear o actualizar grupo
if (m_currentGroupId == -1) {
// Crear nuevo grupo
grupo* newGroup = new grupo();
newGroup->setNombre(ui->nameLineEdit->text().toStdString());
// ... más lógica
m_grupos->agregarGrupo(newGroup);
} else {
// Actualizar grupo existente
grupo* g = m_grupos->getGrupo(m_currentGroupId);
g->setNombre(ui->nameLineEdit->text().toStdString());
// ... más lógica
}

emit applyClicked(); // Emitir señal
}
```

Responsabilidades:

- Inicialización: Configurar la UI y establecer valores por defecto
- Validación: Verificar que los datos ingresados sean correctos
- Lógica de negocio: Crear, actualizar o eliminar entidades (grupos, maestros, salones)
- Comunicación: Emitir señales para notificar cambios a otros widgets
- Gestión de eventos: Responder a clicks, cambios de texto, selecciones, etc.

Arquitectura de Widgets del Sistema

El sistema está compuesto por múltiples widgets especializados:

Widgets Principales

Widget	Archivo .ui	Descripción
MainWindow	mainwindow.ui	Ventana principal, contiene el menú de navegación

GenerationWidget	generationwidget.ui	Genera horarios automáticamente usando algoritmos/
SchedulesWidget	scheduleswidget.ui	Visualiza y gestiona horarios generados.

Widgets de Gestión de Entidades

Widget	Archivo.ui	Descripción
GruposWidget	gruposwidget.ui	Lista todos los grupos (niveles académicos)
GroupDetailsWidget	groupdetailswidget.ui	Crea/edita detalles de un grupo específico
QuartersWidget	quarterswidget.ui	Gestiona cuatrimestres y conjuntos de materias
QuarterDetailsWidget	quarterdetailswidget.ui	Edita detalles de un cuatrimestre
SalonesWidget	saloneswidget.ui	Lista todos los salones disponibles
RoomDetailsWidget	roomdetailswidget.ui	Crea/edita información de un salón

Widgets de Maestros

Widget	Archivo.ui	Descripción

TeacherDisponibilityWidget	teacherdisponibilitywidget.ui	Define horarios de disponibilidad de maestros
TeacherDetailsWidget	teacherdetailswidget.ui	Gestiona información de maestros
TeacherScheduleWidget	teacherschedulewidget.ui	Visualiza horario asignado a un maestro