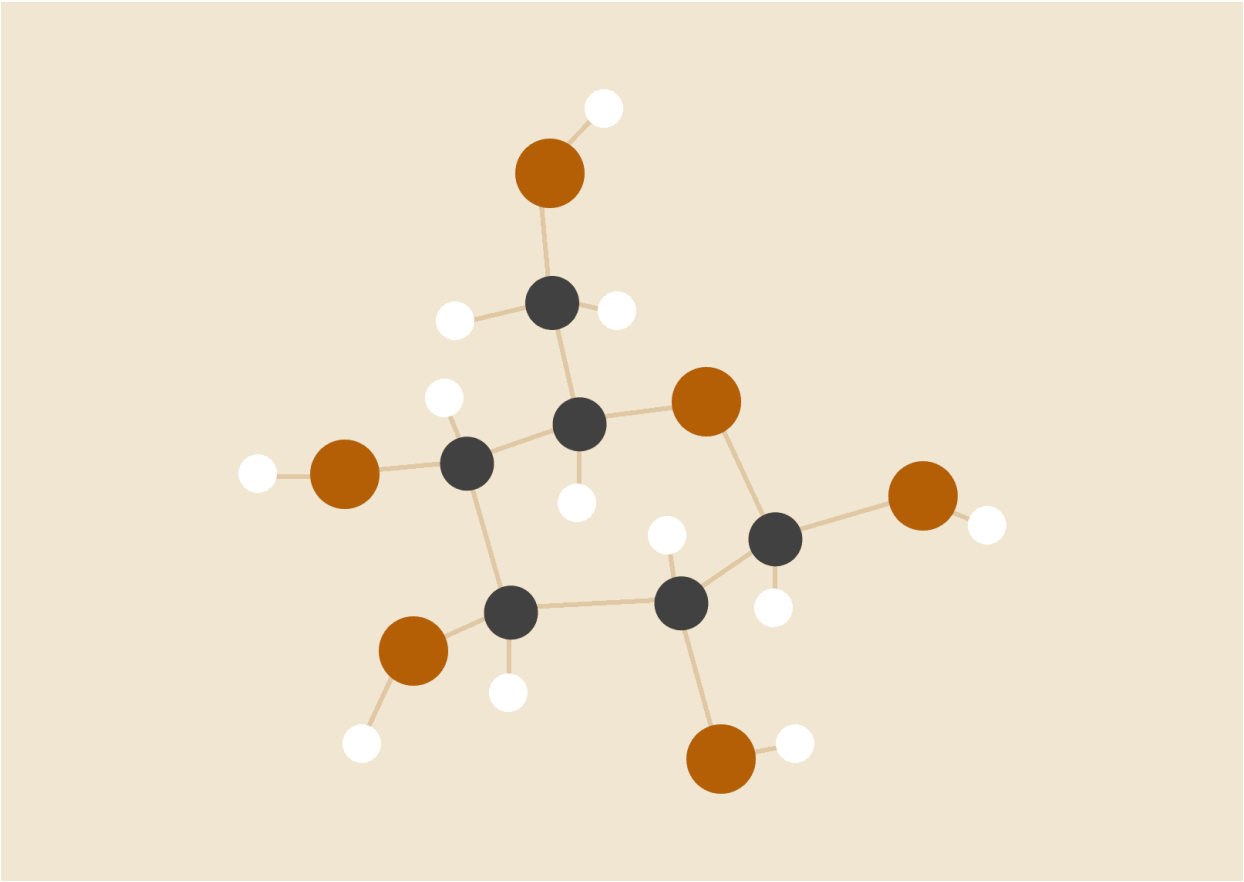
MANUAL TÉCNICO

Proyecto 1 Algoritmos Genéticos



Carlos Benjamin Pac Flores
201931012

14/04/2025
Inteligencia Artificial
Ingeniería en Ciencias y Sistemas CUNOC

ÍNDICE

INTRODUCCIÓN

Este manual técnico documenta el desarrollo e implementación de un sistema diseñado para **generar horarios académicos de forma automatizada**, aplicando los principios de los **algoritmos genéticos** como técnica de optimización.

El objetivo principal del proyecto es demostrar cómo es posible utilizar métodos inspirados en la evolución natural para resolver problemas complejos del mundo real, como la asignación eficiente de horarios de clases en una institución educativa, tomando en cuenta restricciones de docentes, disponibilidad de salones y compatibilidad entre cursos.

Para su desarrollo se utilizó el lenguaje de programación **Python**, el cual ofrece una sintaxis clara, bibliotecas robustas y una comunidad activa que facilita la experimentación y aplicación de técnicas de inteligencia artificial, como los algoritmos evolutivos.

El sistema permite seleccionar los cursos que se desean programar, y a partir de esta información, genera automáticamente una propuesta de horario optimizado. A través de una interfaz gráfica desarrollada con **Tkinter**, el usuario puede visualizar, editar y exportar los resultados obtenidos.

Este documento explica cómo está compuesto el sistema, cómo se organiza internamente cada componente, y de qué forma interactúan los datos y algoritmos para alcanzar una solución funcional.

TECNOLOGÍAS UTILIZADAS

Python 3.11

Lenguaje de programación base del sistema, elegido por su versatilidad, claridad sintáctica y amplia disponibilidad de bibliotecas para procesamiento de datos, visualización y desarrollo de interfaces gráficas.

BIBLIOTECAS Y HERRAMIENTAS

- **Tkinter**

  Librería estándar de Python utilizada para construir la interfaz gráfica del sistema (GUI), facilitando la interacción del usuario con los módulos de carga, edición y visualización de resultados.

- **tkhtmlview**

  Extensión de Tkinter que permite incrustar contenido HTML dentro de la interfaz gráfica, ideal para mostrar información enriquecida de manera clara.

- **pandas**

  Biblioteca esencial para la manipulación de datos tabulares, permitiendo leer, limpiar y transformar archivos CSV de entrada con facilidad.

- **jinja2**

  Motor de plantillas utilizado para generar contenido HTML dinámico, especialmente para los reportes exportables del horario.

- **pdfkit**

  Herramienta que, junto a wkhtmltopdf, permite convertir plantillas HTML en archivos PDF, lo cual es útil para la generación visual de reportes de horarios.

- **openpyxl**

  Utilizada para la exportación de los horarios generados a archivos de Excel (.xlsx), permitiendo su análisis o distribución posterior.

- **matplotlib**

  Biblioteca de visualización gráfica empleada para representar la evolución de la aptitud durante las generaciones del algoritmo genético y otros datos analíticos del sistema.

- **time y tracemalloc (módulos estándar)**

  Usados para el análisis de rendimiento del sistema, permitiendo medir el tiempo de ejecución y el consumo de memoria durante el proceso evolutivo.

ALGORITMO GENÉTICO

Los algoritmos genéticos son métodos de optimización y búsqueda inspirados en los principios de la evolución natural. Basados en ideas como la selección natural, cruce genético y mutación, estos algoritmos permiten explorar grandes espacios de soluciones con el objetivo de encontrar combinaciones óptimas o cercanas al óptimo.

Cada posible solución a un problema es representada como un individuo, y un conjunto de soluciones forma una población. A través de múltiples generaciones, el algoritmo evalúa, selecciona, cruza y muta estos individuos, guiado por una función de aptitud que mide qué tan buena es cada solución.

Gracias a esta capacidad de poder adaptarse e ir evolucionando a lo largo de las iteraciones del algoritmo son útiles en problemas de múltiples soluciones, en especial en entornos con restricciones con múltiples variables como en este ejemplo la generación de horarios.

APLICACION PYTHON

La generación de horarios implica un problema que involucra múltiples restricciones, como la disponibilidad de docentes, la cantidad de salones, la distribución de cursos por semestre y la necesidad de evitar conflictos de horario.

Cada solución generada por el algoritmo representa un posible horario completo, compuesto por asignaciones de cursos a horarios y salones. A lo largo de las generaciones, el algoritmo mejora dichas asignaciones, evaluándose según una función de aptitud personalizada que penaliza los conflictos (como traslapes o asignaciones múltiples en la misma hora) y premia criterios deseables como la continuidad de clases de cursos de una misma ingeniería y semestre.

El sistema fue desarrollado utilizando el lenguaje de programación Python, debido a su sintaxis clara, sus capacidades orientadas a objetos, y su ecosistema de bibliotecas robusto que facilita el trabajo con estructuras de datos, visualización y generación de documentos.

MÉTODOS GENÉTICOS IMPLEMENTADOS

Para llegar a una solución viable y optimizada, se implementaron cada unos de los siguientes métodos característicos de los algoritmos genéticos, también justificando la utilización de cada uno de ellos y los problemas afrontados en la creación de la solución.

POBLACIÓN INICIAL

Se genera un población inicial de individuos que por defecto es **100** pero que puede ser modificada por el usuario, la población inicial representa un horario completo con todos los cursos a ingresar asignados en un horario con un docente y salon en específico, todo esto realizado de forma aleatoria, pero controlada, respetando las restricciones básicas de disponibilidad, es decir un docente se genera solamente en su rango de tiempo asociado y con los cursos que él puede impartir, de esta forma generacion versiones básicas y útiles de horario, ya que dejarlo de manera aleatoria estos no tendrían un sentido y serían demasiado difíciles de corregir.

Todo esto es controlado por los cursos, ya que un horario tendrá un curso asignado una sola vez de esta forma controlando el crecimiento y validez del mismo es decir un mínimo de validez y funcionalidad.

FUNCIÓN DE APTITUD

La evaluación de qué tan bueno es nuestro horario es controlado por esta función la cual está confeccionada de la siguiente manera, tomando en cuenta los bonus de continuidad y los conflictos que pueden llegar a pasar en el horario.

$$AP = cantidad\_cursos + (bonus * 0.25) - (esh + edh + (traslapes * 1.5)) * 1.5$$

donde:
esh = ***errores_salon_hora***
edh = ***errores_docente_hora***

¿Por qué la confección de esta forma?

Es debido a que el funcionamiento de la implementación necesita de obtener una puntuación alta para reconocer el mejor debido a que se basa en el elitismo tema que veremos más adelante ya que implementan diferentes variaciones.

Al cantidad de cursos es la nota "**mínima**" de un buen horario es decir todo asignado en su lugar pero sin tener bonus o errores, los bonus le dan más visibilidad a un horario pero hay que tener cuidado ya que el bonus puede estar mezclado con errores, por esa

razón el peso es de **0.25** o **25%** es importante pero puede meter ruido al resultado por eso es incluido con un peso no tan alto, ahora los errores están compuestos por 3 diferentes en la implementación:

- **Errores de cruce de salón y hora:** Mismo salon usado a la misma hora, este error a nivel de error representa una penalización de **1** error o **100%** de un error, este error se puede solucionar fácil al final del algoritmo.
- **Errores de cruce de salón y hora:** El docente imparte otro curso a la misma hora, al igual que el anterior es un error fácil de solucionar y representa un punto de error **1** o **100%** de un error.
- **Errores de traslape:** Son errores de que un curso obligatorio de una ingeniería se cruza con otro curso obligatorio de las misma ingeniería, lo que genera un problema serio de continuidad, por ende estos horarios hay que evitarlo lo mejor posible, debido a esto este tipo de error representa **1.5** puntos o **150%** de un error, ya que es un problema serio y no estan facil de resolver sin realizar varios movimientos en un horario, por eso el peso de penalización.

El conjunto de estos errores sumados esta tambien proporcionado a que representen un **150%** más, mientras más errores tenga mi horario mayor tiempo de solución de conflictos, por ende es un horario no tan útil, y mientras menos errores tengamos si representara un peso grande pero no será tanto para que este horario si valga la pena de resolver sus conflictos.

SELECCIÓN

Elitismo

Es una técnica dentro de los algoritmos genéticos que consiste en preservar un subconjunto de los mejores individuos de una generación para transferirlos directamente a la siguiente, sin ser alterados por procesos de cruce o mutación. Esta estrategia garantiza que la calidad de las soluciones no se degrade a lo largo de las generaciones, asegurando que los mejores individuos no se pierdan por efecto aleatorio.

Elitismo Intergeneracional

Es una extensión del elitismo clásico que compara los mejores individuos de distintas generaciones para preservar y seleccionar el mejor entre ellos a lo largo del tiempo. Su objetivo principal es asegurar que, al finalizar el proceso evolutivo, se conserve el mejor individuo global generado durante todas las generaciones, y no solo el mejor de la

última.

Sistema de Torneo

Es un método de selección utilizado en algoritmos genéticos para elegir los individuos que participarán en el proceso de cruce (reproducción). Este mecanismo simula una competencia entre un subconjunto aleatorio de individuos de la población, de la cual se selecciona al mejor según su aptitud.

Aplicación

Se utiliza selección elitista, donde el **20%** superior de la población (con mayor aptitud) se conserva directamente para la siguiente generación. El resto se seleccionó mediante muestreo aleatorio (**Sistema de Torneo**) dentro del **60%** más apto, esto anterior una implementación para la creación de nuevas generaciones de horarios.

Ya para encontrar una solución apta, se mantiene un registro de todas las aptitudes por generación y guardar la mejor de todas, y luego de eso volver a aplicar elitismo sobre todos los resultados que se han obtenido, así aseguramos obtener un máximo local mejor entre todos los máximos locales que se generaron.

CRUCE (CROSSOVER)

Cruce de Dos Puntos con Reparación (Utilizado)

El cruce de dos puntos con reparación es una estrategia utilizada en algoritmos genéticos para combinar dos soluciones (padres) y generar un nuevo individuo (hijo), garantizando que los elementos (genes) del resultado no se repitan.

El proceso consiste en seleccionar aleatoriamente dos puntos de corte dentro del genotipo del individuo. La sección comprendida entre estos dos puntos se copia directamente desde uno de los padres. Luego, se completa el resto del genotipo con los elementos del otro padre, excluyendo aquellos que ya han sido heredados, asegurando así la ausencia de duplicados.

Esta técnica permite preservar parcialmente el orden y la estructura de los padres, mientras se asegura la validez del individuo generado. Se considera una variación del **Order Crossover (OX)**, adaptada para contextos donde los elementos asignados deben mantenerse únicos.

Order Crossover (OX) (Probado pero no agregado a solución final)

El Order Crossover (OX) es un operador de cruce ampliamente utilizado en algoritmos

genéticos, especialmente en problemas donde el orden de los elementos es crucial, como en rutas, secuencias o asignaciones, por ejemplo en la planificación de horarios.

El objetivo principal del OX es preservar el orden relativo y la posición de los elementos entre los padres, garantizando al mismo tiempo que los genes del hijo sean únicos. Esto lo convierte en un método apropiado para problemas de tipo Permutación, donde no se pueden repetir elementos.

Crossover Aleatorio (Probado pero no agregado a solución final)

El Crossover Aleatorio, también conocido como Crossover Uniforme Básico, es una técnica de recombinación utilizada en algoritmos genéticos cuyo principio se basa en la selección aleatoria de genes desde ambos padres sin aplicar restricciones respecto al orden o la unicidad de los elementos. En cada posición del cromosoma hijo, se elige aleatoriamente uno de los genes del padre 1 o del padre 2, sin verificar si dicho gen ya ha sido asignado previamente.

MUTACIÓN

La **mutación** es una de las operaciones fundamentales en los algoritmos genéticos, y su objetivo principal es **preservar la diversidad genética dentro de la población** a lo largo de las generaciones. En el contexto del presente sistema de generación de horarios académicos, se ha implementado una estrategia de mutación adaptada al tipo de solución tratada: un conjunto de asignaciones curso-docente-salón-horario.

Mecanismo aplicado

La mutación se ejecuta de manera probabilística, controlada por un parámetro denominado **tasa de mutación**. Para cada individuo (horario completo), se genera un número aleatorio entre 0 y 1. Si este valor es mayor a la tasa establecida (por ejemplo, 0.1), se procede con la mutación.

El proceso consiste en los siguientes pasos:

1. **Selección de una asignación aleatoria** dentro del individuo (es decir, una clase específica).

2. Se identifica al **docente responsable** de dicha asignación y se recupera su rango de disponibilidad horaria (hora de entrada y salida).

3. Se calcula un conjunto de **horarios válidos** considerando los bloques de clase

disponibles dentro de ese rango.

4. Se selecciona aleatoriamente un nuevo horario de este conjunto.

5. Finalmente, **se reemplaza el horario original de la asignación** por el nuevo valor seleccionado.

Este procedimiento garantiza que la mutación no invalide la solución, ya que siempre se respetan las restricciones de disponibilidad del docente, pero puede crear conflictos pero debido a que necesitamos tener una mejor visibilidad de opciones la tasa de mutación es alta.

ARQUITECTURA DEL SISTEMA

Interfaz Gráfica de Usuario (GUI)

La aplicación presenta una interfaz gráfica desarrollada en **Tkinter**, una biblioteca estándar de Python. Esta interfaz permite a los usuarios:

- Seleccionar los cursos a incluir en el horario.
- Configurar los parámetros del algoritmo genético (generaciones, población).
- Ejecutar el algoritmo con un solo clic.
- Visualizar y editar manualmente el horario generado.
- Analizar el rendimiento del algoritmo mediante gráficas y métricas.

Esta capa es totalmente interactiva y encapsula la lógica de visualización y entrada de datos, sin depender directamente de la lógica de negocio.

Módulo de Datos

El sistema utiliza clases dedicadas para representar las distintas entidades que intervienen en el problema:

- Curso: contiene atributos como código, nombre, carrera, semestre, sección y tipo.
- Docente: incluye datos identificativos y su disponibilidad horaria.
- Salon: representa los espacios físicos disponibles.
- Asignacion: una unidad que relaciona un curso, un docente, un salón y un

horario.
- RelacionDocenteCurso: representa la compatibilidad entre docentes y cursos.

La carga de estos datos se realiza desde archivos .csv utilizando el módulo pandas, y se valida cuidadosamente antes de alimentar el motor genético.

Núcleo del Algoritmo Genético

La clase AlgoritmoGenetico es el motor de optimización del sistema. Contiene la lógica necesaria para:

- Generar la **población inicial**.
- Evaluar individuos con una función de aptitud basada en restricciones duras y blandas.
- Aplicar operadores de **selección**, **cruce** y **mutación**.
- Implementar **elitismo intergeneracional**, asegurando la conservación de los mejores individuos.
- Registrar estadísticas de cada generación para visualización posterior.

Este módulo es independiente del entorno gráfico, lo que facilita su reutilización y prueba por separado.

Exportación y Reportes

Una vez generado el horario, se dispone de módulos para su **exportación a Excel** (openpyxl) y a **formato PDF** (pdfkit y jinja2). También se generan gráficas con matplotlib, integradas directamente en la interfaz, lo cual proporciona retroalimentación inmediata al usuario.

Validación y Diagnóstico

El sistema ofrece soporte para:

- Visualización de **errores de carga** de datos.
- Identificación de **conflictos de asignación** (por color en la vista de edición).
- Cálculo de métricas como aptitud, penalizaciones, bonificaciones y uso de memoria/tiempo.
- Visualización de **porcentaje de cursos continuos** por carrera.

Carga de Archivos CSV

El sistema realiza la carga inicial de datos a partir de archivos CSV ubicados en el directorio estático data/, el cual forma parte de la estructura base de la aplicación. Esta carga es esencial para el funcionamiento del algoritmo genético, ya que proporciona toda la información requerida para generar horarios válidos y evaluables.

Los siguientes archivos .csv deben estar correctamente formateados y presentes en dicho directorio:

cursos.csv

Este archivo contiene la lista de cursos disponibles para ser asignados en el horario. Su estructura es la siguiente:

codigo	nombre	carrera	semestre	seccion	tipo
INF101	Algoritmos	Ingeniería en Sistemas	1	A	obligatorio
CIV102	Topografía	Ingeniería Civil	2	B	opcional

tipo puede ser "obligatorio" o "opcional", sin distinguir mayúsculas o minúsculas.

docentes.csv

Contiene los datos de los docentes disponibles, incluyendo su identificación, nombre y disponibilidad horaria.

registro	nombre	hora_entrada	hora_salida
D001	Ana López	13:40	19:30
D002	Carlos Martínez	14:30	20:20

docente_curso.csv

Define las relaciones válidas entre docentes y cursos. Cada fila indica qué docente está autorizado para impartir qué curso.

registro_docente	codigo_curso
D001	INF101
D002	CIV102

salones.csv

Lista de salones disponibles donde pueden ser asignadas las clases, el sistema le agrega los ids automaticamente

nombre
Salon 1
Salon 2
Salon 3

Validaciones durante la carga

El sistema implementa una serie de validaciones para asegurar la integridad de los datos al momento de la carga:

- Verificación de campos obligatorios no vacíos.
- Semestre debe ser un número entre 1 y 10.
- No deben repetirse cursos con el mismo código y sección.
- Verificación del formato de los campos hora_entrada y hora_salida en docentes.csv.

- Validación del tipo de curso (obligatorio, opcional).
- Coincidencia entre docentes y cursos mediante sus claves primarias.

En caso de errores en los archivos, estos se notifican al usuario en una ventana emergente que muestra todos los problemas detectados.

NÚCLEO DE FUNCIONAMIENTO

El archivo **genetico.py** representa el núcleo de procesamiento del sistema, donde se lleva a cabo la ejecución completa del algoritmo genético para la generación de horarios académicos. Este contiene la definición de las clases Horario y AlgoritmoGenetico, las cuales encapsulan la representación de una solución y el mecanismo de evolución respectivamente.

Clase Horario

Esta clase representa un individuo dentro del algoritmo genético. Un objeto de esta clase contiene una lista de asignaciones curso-docente-salón-horario, y permite calcular su calidad (aptitud).

Principales Métodos:

- **calcular_aptitud():** Evalúa la aptitud total del horario considerando bonificaciones por continuidad y penalizaciones por conflictos.
- **contar_conflictos():** Identifica conflictos de horario entre docentes, salones y cursos obligatorios que se traslapan.
- **contar_bonus():** Calcula bonificaciones cuando los cursos de una misma carrera y semestre están en horarios consecutivos.
- **copy():** Devuelve una copia profunda del horario para preservar individuos durante la evolución.

Clase AlgoritmoGenetico

Controla la ejecución del algoritmo genético, desde la generación inicial hasta la aplicación de los operadores genéticos.

Principales Atributos:

- **cursos, docentes, salones, relaciones:** Datos base para construir los individuos.

- **poblacion:** Lista de horarios actuales.
- **mejores_generaciones:** Lista de los mejores horarios por generación.
- **historial_aptitudes:** Registro de la aptitud de cada individuo a lo largo del tiempo.

Métodos Relevantes:

- **generar_poblacion_inicial():** Genera individuos iniciales de forma aleatoria pero válida, respetando la disponibilidad horaria y compatibilidad docente-curso.
- **evolucionar():** Ejecuta el ciclo de vida del algoritmo, iterando sobre las generaciones, evaluando aptitud, aplicando elitismo y seleccionando, cruzando y mutando individuos.
- **seleccionar_cruzar_mutar():** Aplica elitismo y selecciona padres para cruce mediante torneo. Aplica el operador de cruce **cruzarConDosPuntos()** y posteriormente **mutar()** al hijo.
- **mutar(individuo):** Modifica aleatoriamente el horario de una asignación, eligiendo un nuevo horario disponible dentro del rango del docente.
- **cruzarConDosPuntos(padre1, padre2):** Implementación de cruce de dos puntos con reparación que garantiza unicidad de asignaciones por curso y sección.
- **cruzarFaltante(padre1, padre2):** Identifica asignaciones conflictivas en un padre, las elimina y las reemplaza con asignaciones del otro padre.
- **cruzarRandom(padre1, padre2):** Cruce simple donde se combinan asignaciones sin garantizar consistencia. Usado con fines experimentales.
- **cursos_conflicto(asignaciones):** Detecta cursos cuyas asignaciones presentan conflictos según horario, salón o docente.
- **eliminar_conflictos(codigos, asignaciones):** Remueve del individuo las asignaciones que generan conflicto.
- **generar_horarios_validos(docente):** Devuelve los horarios que están dentro del rango disponible del docente.
- **calcular_mapas(...):** Utilidad interna para mapear combinaciones clave de horario con docente/salón/curso para detección de conflictos.

REQUISITOS TÉCNICOS PARA EJECUTAR EL SISTEMA

Para garantizar una correcta instalación y ejecución del sistema de generación de horarios mediante algoritmos genéticos, es necesario cumplir con los siguientes requerimientos técnicos:

Versión mínima de Python

- **Python 3.10 o superior**
  Se recomienda utilizar versiones recientes de Python para asegurar la compatibilidad con bibliotecas modernas y aprovechar mejoras de rendimiento.

Dependencias del sistema

El sistema requiere la instalación de las siguientes bibliotecas de Python, especificadas en el archivo requirements.txt.

```
pip install -r requirements.txt
```

> **Nota:** Es necesario contar con wkhtmltopdf instalado en el sistema operativo para que pdfkit pueda generar correctamente los archivos PDF.

Comando para iniciar la aplicación

Una vez instaladas las dependencias, el sistema puede ejecutarse desde la terminal con:

```
python app.py
```

Estructura del Proyecto

La organización del proyecto sigue una estructura modular y clara:

```
proyecto/
│
├── data/                # Archivos CSV necesarios para el funcionamiento
│   ├── cursos.csv
│   ├── docente_curso.csv
│   ├── docentes.csv
│   └── salones.csv
│
├── venv/                # Entorno virtual (opcional, recomendado)
│
├── app.py               # Archivo principal que inicia la GUI
├── main.py              # (opcional) Archivo alternativo para ejecución
│
├── clases.py            # Definición de entidades (Curso, Docente, etc.)
├── carga.py                  # Lógica de carga y validación de CSV
├── genetico.py               # Núcleo del algoritmo genético
├── exportador_excel.py       # Exportación de horarios a Excel
├── exportador_pdf.py         # Exportación de horarios a PDF (usa
Jinja2)
│
├── horario_generado.html          # Plantilla HTML generada antes del PDF
├── horario_generado.pdf           # Reporte PDF exportado
├── horario_generado.xlsx          # Reporte Excel exportado
│
├── requirements.txt               # Dependencias del sistema
└── README.md                      # Descripción general del proyecto
```