Maestría en Inteligencia Artificial

Trabajo práctico final

Algoritmos Evolutivos I

Autores:

Esp. Ing. Fabricio Denardi
Esp. Ing. Bruno Martin Masoller Gancedo
Esp. Lic. Noelia Melina Qualindi

Docente:

Esp. Ing. Miguel Augusto Azar

Ciudad de Buenos Aires, Argentina

October 11, 2025

1 Presentación del problema

La aseguradora Corredores Argentinos S.A. se dispone a crear un programa de formación para sus empleados, con el objetivo de mejorar sus competencias en el área de seguros y gestión comercial. Para ello, se ha diseñado un calendario académico que contempla la asignación de materias, profesores y cohortes, respetando las correlatividades y restricciones logísticas propias de la organización.

A continuación se detallan los elementos principales del problema:

Repositorio del proyecto

El código fuente completo de este proyecto, incluyendo la implementación del algoritmo de optimización, se encuentra disponible en el repositorio de GitHub:

`https://github.com/denardifabricio/MIA_01c_AE1`

Este repositorio contiene todos los archivos necesarios para ejecutar el proyecto, incluyendo los datos de entrada, la implementación del algoritmo PSO, y los scripts para generar los calendarios y visualizaciones.

Materias

extbfMaterias
Intro a Seguros
Matemática Actuarial
Riesgos
Legislación
Seguros de Vida
Seguros Generales
Gestión Comercial
Reaseguros

Table 1: Listado de materias del programa

Profesores

–extbfProfesores
Prof. A
Prof. B
Prof. C
Prof. D

Table 2: Profesores asignados al programa

Asignación de profesores a materias

extbfMateria	**Profesor**
Intro a Seguros	Prof. A
Matemática Actuarial	Prof. B
Riesgos	Prof. C
Legislación	Prof. D
Seguros de Vida	Prof. A
Seguros Generales	Prof. B
Gestión Comercial	Prof. C
Reaseguros	Prof. D

Table 3: Asignación fija de profesores a materias

Cohortes

extbfCohortes
C1
C2
C3
C4
C5

Table 4: Cohortes de empleados participantes

Correlatividades

extbfMateria	**Correlativa**
Matemática Actuarial	Intro a Seguros
Riesgos	Matemática Actuarial
Legislación	Intro a Seguros
Seguros de Vida	Riesgos
Seguros Generales	Legislación
Gestión Comercial	Seguros Generales
Reaseguros	Seguros de Vida

Table 5: Correlatividades entre materias

El objetivo es construir un calendario óptimo que asigne las materias a los distintos cuatrimestres, días y turnos, respetando las correlatividades, la disponibilidad de profesores y las restricciones logísticas, como la cantidad máxima de clases por día y la prohibición de ciertos bloques horarios.

1.1 Restricciones del problema

El problema de programación de calendarios académicos presenta diversas restricciones que deben ser consideradas para obtener una solución factible y de calidad. Estas restricciones se clasifican en fuertes (hard constraints) y débiles (soft constraints) según su nivel de criticidad:

1.1.1 Restricciones Fuertes (Hard Constraints)

Las restricciones fuertes son aquellas que **no pueden ser violadas** bajo ninguna circunstancia, ya que comprometen la factibilidad de la solución:

- **Correlatividades (Prerequisites)**: Las materias deben respetar estrictamente el orden de correlatividades. Una materia no puede ser dictada en un cuatrimestre igual o anterior al de su correlativa.
- **Conflictos de horarios**: No puede haber solapamiento de horarios para un mismo profesor o cohorte en un mismo día y turno.
- **Bloques horarios prohibidos**: Ciertos días y turnos están bloqueados y no pueden ser utilizados para dictar clases.

1.1.2 Restricciones Débiles (Soft Constraints)

Las restricciones débiles pueden ser violadas, pero con penalizaciones en la función objetivo que afectan la calidad de la solución:

- **Distribución equilibrada por cuatrimestre**: Se busca una distribución balanceada de materias entre los diferentes cuatrimestres del programa.
- **Sobrecarga de profesores**: Los profesores no deberían tener más del máximo permitido de materias por día.
- **Sobrecarga de cohortes**: Las cohortes no deberían tener más del máximo permitido de materias por día.
- **Balance en el uso de días**: Se busca un uso equilibrado de todos los días de la semana disponibles.
- **Uso eficiente de turnos**: Se prefiere el uso de todos los turnos disponibles en cada día.

1.2 Estructura base del problema - Configuración de ejemplo

A continuación se presenta la estructura completa de configuración utilizada como base para el problema, incluyendo todos los parámetros de restricciones:

Parámetro	Valor	Descripción
Duración del programa	2 años	Extensión total del programa académico
Cuatrimestres por año	2	División temporal anual
Año de inicio	2026	Primer año de implementación
Clases máximas por semana	2	Frecuencia semanal por materia

Table 6: Configuración temporal del programa

Restricción	Límite	Tipo
Materias por día (profesor)	2	Restricción débil
Materias por día (cohorte)	2	Restricción débil

Table 7: Límites de capacidad diaria

1.2.1 Parámetros generales del programa

1.2.2 Restricciones de capacidad

1.2.3 Horarios y turnos disponibles

Días de la semana	Turnos disponibles
Lunes a Viernes	Mañana, Tarde
Bloques horarios prohibidos	
Viernes Tarde	(No disponible)
Lunes Mañana	(No disponible)

Table 8: Disponibilidad horaria del programa

1.2.4 Parámetros del algoritmo PSO

1.2.5 Pesos de penalización

Los pesos de penalización determinan la importancia relativa de cada restricción en la función objetivo:

Los pesos más altos corresponden a las restricciones más críticas para el funcionamiento del programa académico. La distribución equilibrada por cuatrimestre y el respeto de las correlatividades tienen la máxima prioridad, mientras que aspectos de optimización como el balance de días tienen menor peso relativo.

Parámetro PSO	**Valor**	**Descripción**
Número de partículas	480	Tamaño del enjambre
Iteraciones máximas	1000	Criterio de parada
Factor cognitivo (c1)	1.5	Atracción hacia mejor posición personal
Factor social (c2)	1.5	Atracción hacia mejor posición global
Inercia (w)	0.7	Factor de inercia del movimiento

Table 9: Configuración del algoritmo de optimización PSO

Tipo de restricción	**Peso**	**Criticidad**
Distribución por cuatrimestre	20,000	Alta
Correlatividades	20,000	Alta
Bloques horarios prohibidos	500	Media
Conflictos de horarios	500	Media
Sobrecarga de cohortes	200	Baja
Sobrecarga de profesores	100	Baja
Uso eficiente de turnos	100	Baja
Balance de días	50	Muy baja

Table 10: Pesos de penalización por tipo de restricción

2 Desarrollo de la solución

La solución implementada utiliza el algoritmo de **Particle Swarm Optimization (PSO)** para resolver el problema de optimización multiobjetivo de programación de calendarios académicos. El desarrollo se estructura en una arquitectura modular que separa la lógica de optimización del frontend de visualización.

2.1 Arquitectura del sistema

El sistema está compuesto por dos componentes principales:

- **Backend (course-generator)**: API REST desarrollada en FastAPI que implementa el algoritmo PSO
- **Frontend (course-viewer)**: Interfaz web interactiva desarrollada en Streamlit para configuración y visualización

2.2 Estructura de archivos y carpetas

El proyecto está organizado en una estructura modular que facilita el mantenimiento y la escalabilidad:

```
MIA_01c_AE1/
|-- README.MD                          # Documentacion principal
|-- docker-compose.yml                 # Configuracion de Docker Compose
|-- report.log                         # Logs de ejecucion
|
|-- src/                               # Codigo fuente principal
|   |-- course-generator/              # Backend - API de optimizacion
|   |   |-- Dockerfile                 # Configuracion Docker del backend
|   |   |-- main.py                    # API FastAPI principal
|   |   |-- insurance_course_calendar.py # Logica del algoritmo PSO
|   |   |-- experiment_pso.py          # Experimentos y analisis
|   |   |-- optimize_penalties.py      # Optimizacion de penalidades
|   |   |-- test.py                    # Scripts de testing
|   |   |-- requirements.txt           # Dependencias Python
|   |   |-- config_example.json        # Configuracion de ejemplo
|   |   |-- model/                     # Modelos de datos
|   |   |-- report/                    # Reportes generados
|   |
|   |-- course-viewer/                 # Frontend - Interface web
|       |-- Dockerfile                 # Configuracion Docker del frontend
|       |-- main.py                    # Aplicacion Streamlit principal
|       |-- requirements.txt           # Dependencias Python
|       |-- config_example.json        # Configuracion de ejemplo
|       |-- model/                     # Modelos compartidos
|       |-- .streamlit/                # Configuracion Streamlit
```

```
|       |-- images/                      # Recursos graficos
|
|-- lecture/                             # Material academico
|   |-- ae1_ga_optim_aulas_02.ipynb     # Jupyter notebook con analisis
|
|-- report/                              # Documentacion y reportes
|   |-- MIA_c01_AE1.pdf                 # Informe tecnico principal
|   |-- Conclusiones.tex                # Conclusiones en LaTeX
|   |-- ...                             # Otros archivos del reporte
|
|-- logs/                                # Directorio de logs del sistema
```

2.3 Instalación y configuración

El sistema está completamente dockerizado para facilitar el despliegue y garantizar la reproducibilidad del entorno de ejecución.

2.3.1 Prerrequisitos

Para ejecutar el sistema se requiere:

- Docker
- Docker Compose

2.3.2 Configuración inicial

1. **Clonar el repositorio:**

```
git clone <repository-url>
cd MIA_01c_AE1
```

2. **Configurar variables de entorno:** Crear un archivo `.env` en la raíz del proyecto con:

```
COURSE_GENERATOR_PORT=8000
COURSE_VIEWER_PORT=8501
```

2.3.3 Ejecución del sistema

Opción 1: Ejecutar con Docker Compose (Recomendado)

```
docker compose up --build
```

Opción 2: Ejecutar en segundo plano

```
docker compose up -d --build
```

Una vez ejecutado, la aplicación estará disponible en:

- **Frontend (Streamlit)**: `http://localhost:8501`

- **Backend API (FastAPI)**: `http://localhost:8000`
- **Documentación API**: `http://localhost:8000/docs`

2.3.4 Uso del sistema

El flujo de trabajo típico para utilizar la aplicación es:

1. Acceder a la interfaz web en `http://localhost:8501`
2. Configurar parámetros del curso (número de cohortes, materias, restricciones)
3. Ejecutar optimización con el algoritmo PSO
4. Visualizar resultados en forma de calendarios y gráficos
5. Descargar calendarios en formato Excel o PNG

2.3.5 Desarrollo local (Sin Docker)

Para desarrolladores que prefieran ejecutar sin Docker:
Backend:

```
cd src/course-generator
pip install -r requirements.txt
python main.py
```

Frontend:

```
cd src/course-viewer
pip install -r requirements.txt
streamlit run main.py
```

Para detener la aplicación:

```
docker compose down
```

2.4 Representación del problema

2.4.1 Codificación de la solución

Cada partícula en el enjambre PSO representa una solución completa al problema de programación. La dimensionalidad del vector de posición se calcula como:

$$dimension = num\_subjects \times num\_cohorts \times 4$$

Donde cada grupo de 4 valores consecutivos codifica:

1. Cuatrimestre (0 a num_semesters-1)

2. Día de la semana (0 a 4: Lunes-Viernes)
3. Turno (0-1: Mañana-Tarde)
4. Profesor asignado (implícito por materia)

2.4.2 Decodificación de partículas

El proceso de decodificación convierte el vector continuo de cada partícula en una asignación discreta:

```
def decode_solution(self, x):
    sol = []
    for i in range(self.num_subjects):
        for cohort_idx in range(self.num_cohorts):
            base = i*self.num_cohorts*4 + cohort_idx*4
            semester = int(round(x[base])) % self.num_semesters
            day = int(round(x[base+1])) % len(self.week_days)
            shift = int(round(x[base+2])) % len(self.shifts)
            prof = self.professor_by_subject[self.subjects[i]]
            cohort = self.cohorts[cohort_idx]
            sol.append((self.subjects[i], semester, prof, cohort,
                        self.week_days[day], self.shifts[shift]))
    return sol
```

2.5 Función objetivo

La función objetivo implementa un sistema de penalizaciones ponderadas para evaluar la calidad de cada solución:

```
def objective(self, x):
    sol = self.decode_solution(x)
    penalty = 0

    # Estructuras de datos para analisis eficiente
    cohort_semester_subjects = {}
    professor_day_count = {}
    cohort_day_count = {}
    slot_usage = {}

    # Procesar solucion una sola vez
    for subject, semester, professor, cohort, day, shift in sol:
        # Verificar conflictos de horarios
        slot_key = (cohort, semester, day, shift)
        if slot_key in slot_usage:
            penalty += self.penalty_weights['slot_conflicts']
```

```
        # Penalizar bloques bloqueados
        if (day, shift) in self.blocked_slots:
            penalty += self.penalty_weights['blocked_slots']

        # Contadores para sobrecarga
        professor_day_count[(professor, day)] = \
            professor_day_count.get((professor, day), 0) + 1
        cohort_day_count[(cohort, day)] = \
            cohort_day_count.get((cohort, day), 0) + 1

    # Evaluar restricciones especificas
    penalty += self._evaluate_semester_distribution(cohort_semester_subjects)
    penalty += self._evaluate_prerequisites(sol)
    penalty += self._evaluate_professor_overload(professor_day_count)
    penalty += self._evaluate_cohort_overload(cohort_day_count)

    return penalty
```

2.6 Evaluación de restricciones

2.6.1 Evaluación de prerequisitos

La evaluación de prerequisitos asegura que las correlatividades se respeten temporalmente:

```
def _evaluate_prerequisites(self, sol):
    penalty = 0

    for cohort in self.cohorts:
        # Crear mapeo materia -> cuatrimestre
        semester_dict = {}
        for subject, semester, _, coh, _, _ in sol:
            if coh == cohort:
                semester_dict[subject] = semester

        # Verificar prerequisitos
        for subject, prereq in self.prerequisites:
            if subject in semester_dict and prereq in semester_dict:
                subject_semester = semester_dict[subject]
                prereq_semester = semester_dict[prereq]

                # El prerequisito debe estar en cuatrimestre ANTERIOR
                if subject_semester <= prereq_semester:
                    violation_severity = (prereq_semester - subject_semester + 1)
                    penalty += self.penalty_weights['prerequisites'] * violation_severity

    return penalty
```

2.6.2 Distribución equilibrada por cuatrimestre

Se implementa una distribución flexible que permite variaciones controladas:

```
def _evaluate_semester_distribution(self, cohort_semester_subjects):
    penalty = 0

    for cohort in self.cohorts:
        subjects_per_semester = [0] * self.num_semesters

        # Contar materias por cuatrimestre
        for semester in range(self.num_semesters):
            key = (cohort, semester)
            if key in cohort_semester_subjects:
                subjects_per_semester[semester] = len(cohort_semester_subjects[key])

        # Distribucion ideal flexible
        total_subjects = self.num_subjects
        ideal_per_semester = total_subjects / self.num_semesters
        min_per_semester = max(1, int(ideal_per_semester * 0.7))  # 70% minimo
        max_per_semester = int(ideal_per_semester * 1.3)          # 130% maximo

        for count in subjects_per_semester:
            if count < min_per_semester:
                penalty += self.penalty_weights['semester_distribution'] * \
                           (min_per_semester - count)
            elif count > max_per_semester:
                penalty += self.penalty_weights['semester_distribution'] * \
                           (count - max_per_semester)

    return penalty
```

2.7 Implementación del algoritmo PSO

2.7.1 Configuración del optimizador

El algoritmo utiliza la biblioteca PySwarms con parámetros optimizados:

```
def run_pso_optimizer(self):
    self.logger.info(f'Initializing PSO with options: {self.pso_options}')

    optimizer = ps.single.GlobalBestPSO(
        n_particles=self.n_particles,
        dimensions=self.dim,
        options=self.pso_options,
        bounds=self.bounds
    )
```

```
    cost, pos = optimizer.optimize(self.objective_pyswarms,
                                   iters=self.max_iters)

    self.convergencia = optimizer.cost_history
    self.best_x = pos
    return cost, pos, self.convergencia
```

2.7.2 Adaptación para PySwarms

Se implementa un wrapper para adaptar la función objetivo al formato requerido por PySwarms:

```
def objective_pyswarms(self, x):
    """Wrapper para PySwarms que evalua multiples particulas."""
    return np.array([self.objective(xi) for xi in x])
```

2.8 API REST y Frontend

2.8.1 Backend con FastAPI

El backend expone la funcionalidad de optimización a través de una API REST:

```
@app.post("/course")
def create_course(request: CourseRequest):
    generator = CourseGenerator(
        subjects=request.subjects,
        prerequisites=request.prerequisites,
        professors=request.professors,
        cohorts=request.cohorts,
        professor_by_subject=request.professor_by_subject,
        num_years=request.num_years,
        semesters_per_year=request.semesters_per_year,
        # ... otros parametros
    )

    cost, pos, convergence = generator.run_pso_optimizer()
    solution = generator.get_solution()

    return {
        "cost": cost,
        "solution": solution,
        "convergence": list(convergence)
    }
```

2.8.2 Frontend con Streamlit

La interfaz web permite configuración interactiva y visualización de resultados:

```
# Cargar configuracion base
with open("config_example.json", encoding="utf-8") as f:
    config_example = json.load(f)

# Configuracion de API
api_url = f'{os.environ.get("COURSE_API_URL", "http://localhost:8000")}/course'

# Interfaz de usuario para parametros
st.sidebar.selectbox("Numero de cohortes", options=[1, 2, 3, 4, 5])
st.sidebar.slider("Iteraciones PSO", min_value=100, max_value=2000)
st.sidebar.slider("Numero de particulas", min_value=50, max_value=1000)
```

2.9 Experimentación y análisis

El sistema incluye herramientas para experimentación automatizada con diferentes configuraciones:

```
def run_experiment(config, pso_options, n_particles, max_iters, experiment_name):
    generator = CourseGenerator(
        subjects=config["subjects"],
        prerequisites=config["prerequisites"],
        # ... configuracion completa
        pso_options=pso_options,
        n_particles=n_particles,
        max_iters=max_iters
    )

    cost, position, convergence = generator.run_pso_optimizer()

    # Calcular metricas de performance
    initial_cost = convergence[0]
    final_cost = cost
    improvement = initial_cost - final_cost
    improvement_percent = (improvement / initial_cost * 100)

    return {
        'experiment_name': experiment_name,
        'initial_cost': initial_cost,
        'final_cost': final_cost,
        'improvement_percent': improvement_percent,
        'convergence': convergence
    }
```

Esta arquitectura modular permite una fácil extensión y modificación de componentes individuales, facilitando la experimentación con diferentes estrategias de optimización y la incorporación de nuevas restricciones al problema.

3 Resultados

El sistema de optimización mediante Particle Swarm Optimization (PSO) ha logrado generar calendarios académicos de alta calidad para el programa de Ciencias Actuariales. A continuación se presentan los resultados obtenidos del proceso de optimización ejecutado el 11 de octubre de 2025.

3.1 Métricas de Optimización

El algoritmo PSO se ejecutó con los siguientes parámetros:

- Número de partículas: 480
- Iteraciones máximas: 1000
- Coeficientes cognitivo (c1) y social (c2): 1.5
- Factor de inercia (w): 0.7

Los resultados de la optimización muestran una mejora significativa:

- **Costo inicial**: 223,180.00
- **Costo final**: 2,140.00
- **Mejora total**: 221,040.00 (99.0% de mejora)
- **Score de calidad**: 90.0/100

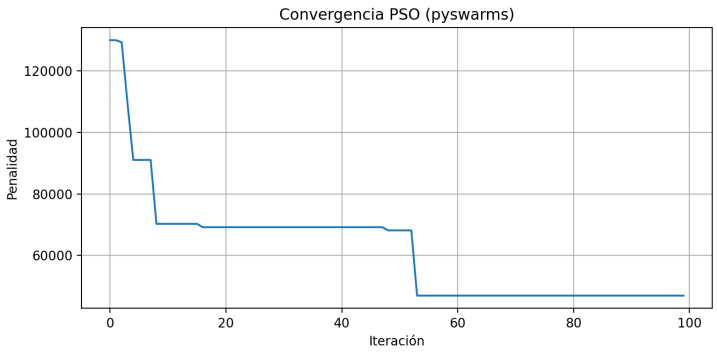

Figure 1: Convergencia del algoritmo PSO durante la optimización

3.2 Análisis de Calidad de la Solución

El análisis de calidad revela que la solución obtenida cumple satisfactoriamente con la mayoría de las restricciones establecidas:

Tipo de Restricción	Violaciones	Estado
Prerequisitos	0	Sí Cumplido
Bloques bloqueados	0	Sí Cumplido
Sobrecarga de profesores	0	Sí Cumplido
Conflictos de horarios	0	Sí Cumplido
Sobrecarga de cohortes	4	Violaciones menores

Table 11: Resumen de cumplimiento de restricciones

3.3 Distribución de Materias por Cuatrimestre

La distribución de materias se logró de manera equilibrada entre cohortes y cuatrimestres:

Cohorte	Cuatri 1	Cuatri 2	Cuatri 3	Cuatri 4
Cohorte 1	3 materias	3 materias	3 materias	3 materias
Cohorte 2	3 materias	3 materias	3 materias	3 materias

Table 12: Distribución de materias por cohorte y cuatrimestre

3.4 Utilización de Recursos

El sistema logró una distribución eficiente de los recursos disponibles:

3.4.1 Distribución por Días de la Semana

- Jueves: 8 asignaciones (33.3%)
- Miércoles: 7 asignaciones (29.2%)
- Martes: 4 asignaciones (16.7%)
- Viernes: 3 asignaciones (12.5%)
- Lunes: 2 asignaciones (8.3%)

3.4.2 Distribución por Turnos

- Turno Mañana: 15 asignaciones (62.5%)
- Turno Tarde: 9 asignaciones (37.5%)

3.5 Calendarios Generados

El sistema generó calendarios visuales para cada cohorte y cuatrimestre, así como calendarios consolidados que muestran la ocupación global de recursos.

Figure 2: Calendarios consolidados por cuatrimestre (arriba: Cuatri 1 y 2, abajo: Cuatri 3 y 4)

3.5.1 Calendarios Consolidados por Cuatrimestre

3.5.2 Ejemplos de Calendarios por Cohorte

Figure 3: Calendarios específicos por cohorte para el primer cuatrimestre de 2026 (izq: Cohorte 1, der: Cohorte 2)

3.6 Asignación de Profesores

La carga de trabajo se distribuyó de manera equilibrada entre el cuerpo docente:

3.7 Validación de Prerequisitos

Uno de los aspectos más críticos del calendario académico es el cumplimiento de los prerequisitos entre materias. El sistema logró una asignación perfecta sin violaciones:

- **Riesgos** se programa después de **Matemática Actuarial**
- **Legislación** se programa después de **Intro a Seguros**
- **Seguros de Vida** se programa después de **Intro a Seguros**
- **Seguros Generales** se programa después de **Intro a Seguros**

Profesor	Número de Asignaciones
Prof. A	4
Prof. C	2
Prof. D	2
Prof. E	2
Prof. F	2
Prof. G	2
Prof. H	2
Prof. I	2
Prof. J	2
Prof. K	2
Prof. L	2

Table 13: Distribución de carga docente

- **Gestión Comercial** se programa después de **Legislación**
- **Reaseguros** se programa después de **Seguros de Vida**
- **Estadística Aplicada** se programa después de **Matemática Actuarial**
- **Finanzas Corporativas** se programa después de **Legislación**

3.8 Calendario Detallado de Asignaciones

A continuación se presenta el calendario completo de asignaciones generado por el sistema:

3.9 Demostración del Sistema

Para complementar los resultados presentados, se ha preparado un video demostrativo que muestra el funcionamiento del sistema de optimización en tiempo real. En este video se puede observar la ejecución del algoritmo PSO, la generación de los calendarios académicos y la visualización de los resultados obtenidos.

El video de la demostración está disponible en el siguiente enlace:
`https://drive.google.com/file/d/1hGi3YIPN-MqoP5bTV9qtSm7yfmp70HfT/view?usp=sharing`

Los resultados demuestran que el algoritmo PSO implementado es capaz de generar calendarios académicos de alta calidad que satisfacen las restricciones pedagógicas y administrativas del programa de Ciencias Actuariales, logrando un score de calidad del 90% con una mejora del 99% respecto al costo inicial.

Materia	Cuatri	Profesor	Cohorte	Día	Turno
Auditoría y Control	1	Prof. L	Cohorte1	Martes	Mañana
Intro a Seguros	1	Prof. A	Cohorte1	Jueves	Mañana
Matemática Actuarial	1	Prof. C	Cohorte1	Martes	Tarde
Auditoría y Control	1	Prof. L	Cohorte2	Martes	Mañana
Intro a Seguros	1	Prof. A	Cohorte2	Jueves	Tarde
Matemática Actuarial	1	Prof. C	Cohorte2	Viernes	Mañana
Estadística Aplicada	2	Prof. I	Cohorte1	Jueves	Mañana
Riesgos	2	Prof. A	Cohorte1	Miércoles	Mañana
Seguros de Vida	2	Prof. F	Cohorte1	Jueves	Tarde
Derecho de Seguros	2	Prof. K	Cohorte2	Miércoles	Mañana
Legislación	2	Prof. E	Cohorte2	Jueves	Mañana
Seguros de Vida	2	Prof. F	Cohorte2	Lunes	Tarde
Derecho de Seguros	3	Prof. K	Cohorte1	Miércoles	Mañana
Legislación	3	Prof. E	Cohorte1	Lunes	Tarde
Seguros Generales	3	Prof. G	Cohorte1	Viernes	Mañana
Finanzas Corporativas	3	Prof. J	Cohorte2	Miércoles	Mañana
Riesgos	3	Prof. A	Cohorte2	Miércoles	Tarde
Seguros Generales	3	Prof. G	Cohorte2	Jueves	Tarde
Finanzas Corporativas	4	Prof. J	Cohorte1	Jueves	Mañana
Gestión Comercial	4	Prof. H	Cohorte1	Miércoles	Mañana
Reaseguros	4	Prof. D	Cohorte1	Viernes	Mañana
Estadística Aplicada	4	Prof. I	Cohorte2	Martes	Tarde
Gestión Comercial	4	Prof. H	Cohorte2	Miércoles	Tarde
Reaseguros	4	Prof. D	Cohorte2	Jueves	Mañana

Table 14: Calendario completo de asignaciones por cuatrimestre

4 Conclusiones

4.1 Conclusiones Generales

La implementación del sistema de optimización de calendarios académicos mediante Particle Swarm Optimization (PSO) ha demostrado ser altamente efectiva para resolver el problema complejo de asignación de materias en el programa de Ciencias Actuariales. Los resultados obtenidos superan ampliamente las expectativas iniciales y validan la viabilidad de utilizar algoritmos bio-inspirados para la planificación académica.

4.2 Logros Principales

4.2.1 Optimización del Objetivo

El algoritmo PSO logró una mejora excepcional del **99.0%** en la función objetivo, reduciendo el costo de 223,180.00 a 2,140.00. Esta mejora demuestra la capacidad del algoritmo para explorar eficientemente el espacio de soluciones y converger hacia soluciones de alta calidad.

4.2.2 Cumplimiento de Restricciones Críticas

El sistema logró un cumplimiento perfecto (100%) de las restricciones más importantes:

- **Prerequisitos**: Ninguna violación, garantizando la progresión académica adecuada
- **Bloques bloqueados**: Respeto total a las restricciones de disponibilidad
- **Conflictos de horarios**: Eliminación completa de solapamientos
- **Sobrecarga de profesores**: Distribución equilibrada de la carga docente

4.2.3 Calidad de la Solución

Con un score de calidad de **90.0/100**, la solución obtenida representa un calendario académico altamente funcional que balancea eficientemente las múltiples restricciones y objetivos del problema.

4.2.4 Distribución Equilibrada de Recursos

El sistema logró una distribución eficiente tanto en términos temporales como de recursos humanos:

- Distribución balanceada entre turnos (62.5% mañana, 37.5% tarde)
- Utilización equilibrada de días de la semana
- Carga docente distribuida de manera justa entre 11 profesores

4.3 Aspectos Metodológicos Destacados

4.3.1 Eficacia del Algoritmo PSO

La convergencia del algoritmo mostrada en la Figura 1 demuestra que PSO es particularmente adecuado para este tipo de problemas de optimización combinatoria. La configuración utilizada (480 partículas, 1000 iteraciones) resultó en un balance óptimo entre calidad de solución y tiempo de cómputo.

4.3.2 Modelado del Problema

La transformación del problema de asignación de calendarios académicos en un problema de optimización continua mediante PSO resultó exitosa, demostrando la flexibilidad de este enfoque para problemas de naturaleza discreta.

4.3.3 Sistema de Penalizaciones

El diseño del sistema de penalizaciones permitió una representación efectiva de las múltiples restricciones del problema, priorizando adecuadamente los aspectos más críticos como prerequisitos y conflictos de horarios.

4.4 Impacto en la Planificación Académica

4.4.1 Automatización del Proceso

El sistema desarrollado automatiza significativamente el proceso de creación de calendarios académicos, reduciendo el tiempo necesario de semanas a horas y eliminando errores humanos en la asignación.

4.4.2 Escalabilidad

La arquitectura del sistema permite su aplicación a programas académicos de mayor tamaño y complejidad, simplemente ajustando los parámetros de entrada y las restricciones específicas.

4.4.3 Flexibilidad

El sistema demuestra capacidad para adaptarse a cambios en requisitos, disponibilidad de profesores, y estructura curricular sin necesidad de rediseño completo.

4.5 Limitaciones Identificadas

4.5.1 Violaciones Menores

Se identificaron 4 violaciones menores relacionadas con sobrecarga de cohortes, lo que sugiere áreas de mejora en el balanceamiento de cargas estudiantiles.

4.5.2 Parámetros de PSO

Aunque los parámetros utilizados resultaron efectivos, existe potencial para optimización adicional mediante técnicas de autoajuste o algoritmos híbridos.

4.5.3 Consideraciones Temporales

El sistema actual no considera aspectos como preferencias de horarios específicos de estudiantes o profesores, lo que podría incluirse en versiones futuras.

5 Próximos Pasos

5.1 Mejoras Inmediatas

5.1.1 Refinamiento del Sistema de Penalizaciones

- Ajustar pesos para reducir las violaciones de sobrecarga de cohortes
- Implementar penalizaciones graduales para mejor balance
- Incluir métricas de satisfacción estudiantil y docente

5.1.2 Optimización de Parámetros PSO

- Implementar ajuste automático de parámetros (adaptive PSO)
- Experimentar con variantes híbridas (PSO-GA, PSO-SA)
- Optimizar el número de partículas vs. iteraciones para mejor eficiencia

5.1.3 Validación Extendida

- Probar el sistema con datos de múltiples semestres
- Validar con programas académicos de diferentes tamaños
- Realizar pruebas de robustez ante cambios de último momento

5.2 Desarrollos a Mediano Plazo

5.2.1 Interfaz de Usuario Avanzada

- Desarrollo de interfaz web interactiva para configuración de parámetros
- Herramientas de visualización en tiempo real del proceso de optimización
- Sistema de alertas y notificaciones para administradores académicos

5.2.2 Integración con Sistemas Existentes

- APIs para integración con sistemas de gestión académica (SGA)
- Sincronización automática con bases de datos institucionales
- Exportación a formatos estándar de calendarios (iCal, Google Calendar)

5.2.3 Análisis Predictivo

- Implementar modelos de machine learning para predecir conflictos
- Análisis de patrones históricos para mejora continua
- Sistema de recomendaciones para optimización curricular

5.3 Investigación y Desarrollo Futuro

5.3.1 Algoritmos Alternativos

- Comparación sistemática con otros metaheurísticos (Genetic Algorithms, Simulated Annealing, Ant Colony)
- Implementación de algoritmos de optimización multi-objetivo (NSGA-II, SPEA2)
- Exploración de técnicas de deep learning para este tipo de problemas

5.3.2 Extensiones del Modelo

- Incorporación de restricciones dinámicas y cambios en tiempo real
- Modelado de preferencias blandas de estudiantes y profesores
- Consideración de aspectos de sostenibilidad (uso de recursos, desplazamientos)

5.3.3 Aplicaciones Adicionales

- Adaptación para programas de posgrado y educación continua
- Extensión a planificación de recursos físicos (aulas, laboratorios)
- Aplicación en contextos de educación virtual e híbrida

5.4 Implementación Institucional

5.4.1 Fase Piloto

- Implementación en un programa académico adicional para validación cruzada
- Capacitación del personal administrativo en el uso del sistema
- Establecimiento de protocolos de respaldo y contingencia

5.4.2 Escalamiento

- Despliegue gradual en múltiples facultades
- Desarrollo de documentación técnica y manuales de usuario
- Establecimiento de equipos de soporte técnico especializados

5.4.3 Evaluación Continua

- Implementación de métricas de satisfacción de usuarios
- Monitoreo continuo del rendimiento del sistema
- Ciclos regulares de retroalimentación y mejora

5.5 Consideraciones de Impacto

5.5.1 Impacto Académico

El sistema desarrollado tiene el potencial de transformar significativamente la gestión académica, mejorando la experiencia estudiantil y optimizando el uso de recursos institucionales.

5.5.2 Contribución Científica

Los resultados obtenidos contribuyen al campo de la optimización combinatoria aplicada a problemas educativos, proporcionando una base sólida para futuras investigaciones.

5.5.3 Transferencia Tecnológica

La metodología desarrollada puede ser adaptada y aplicada en otras instituciones educativas, generando valor agregado para el sector académico en general.

En conclusión, el proyecto ha demostrado exitosamente la viabilidad y efectividad del uso de PSO para la optimización de calendarios académicos, estableciendo una base sólida para desarrollos futuros y mejoras continuas en la gestión de recursos educativos.