

**UNIVERSIDAD NACIONAL DEL SANTA**  
**FACULTAD DE INGENIERÍA**  
**INGENIERÍA DE SISTEMAS E INFORMÁTICA**

---

**ALGORITMOS EVOLUTIVOS DE APRENDIZAJE**

Código: 1411-2278

**ACTIVIDAD EN AULA GRUPAL SEMANA 9**  
**CALIFICADA (0-20)**

**Operadores de Selección y Cruce en Algoritmos Genéticos**

---

**Docente:** Ms. Ing. Johan Max Alexander López Heredia

**Semestre:** 2025-I

**Duración:** 35 minutos

Código	Apellidos y Nombres	Firma
0202114033	Guevara Felipe Javier Valentino	
0202114044	Osorio Juaquin Edwin	
202114048	Portal Ibañez Anderzon	

**INSTRUCCIONES GENERALES**

- **Grupos:** Máximo 4-5 integrantes por grupo
- **Tiempo:** 35 minutos para completar todas las actividades
- **Material:** Solo lapicero (no calculadora, no laptop)
- **Entrega:** Al finalizar, entregar esta separata completa al docente
- **Calificación:** Evaluación grupal (0-20 puntos)

## CONTEXTO: Recordando la Semana Anterior

En la **Semana 8** trabajamos con **representaciones cromosómicas** para el problema de distribución de estudiantes. Teníamos 39 alumnos que debían distribuirse en 3 exámenes (A, B, C) con 13 alumnos cada uno, buscando equilibrio en las notas.

Implementamos tres representaciones:

- **Binaria:** 117 bits (39 alumnos × 3 bits c/u)
- **Real:** 117 valores normalizados
- **Permutacional:** Orden de 39 índices

Ahora en la **Semana 9**, estudiaremos cómo crear nuevas generaciones usando **operadores de selección y cruce**.

## CASO PRÁCTICO: Sistema de Distribución Inteligente

**Situación:** Eres parte del equipo que desarrolla un sistema para optimizar la distribución de estudiantes. Tu algoritmo genético ya tiene una población de 6 soluciones candidatas. Ahora debes aplicar operadores de selección y cruce para generar la siguiente generación.

### Población Actual (Generación 0)

Individuo	Fitness	Descripción
A	85	Muy buena distribución
B	45	Distribución regular
C	70	Buena distribución
D	20	Distribución deficiente
E	60	Distribución aceptable
F	90	Excelente distribución

## ACTIVIDAD 1: SELECCIÓN POR TORNEO (8 puntos)

La **selección por torneo** elige individuos comparando pequeños grupos al azar.

**Proceso:** Para cada torneo, selecciona 2 individuos al azar, compara sus fitness y el mejor "gana".

### Pregunta 1.1 (2 puntos)

Realiza 3 torneos de tamaño 2. Para cada torneo, indica:

- Los 2 individuos seleccionados al azar
- Sus respectivos fitness
- El ganador del torneo

#### Torneo 1:

Individuos seleccionados: **F** (fitness: **90**) y **C** (fitness: **70**)

Ganador: **F** ¿Por qué? **F** tiene un mayor fitness (90 frente a 70 de C), por lo tanto es el ganador del torneo.

**Torneo 2:**

Individuos seleccionados: B(fitness: 45) y E(fitness: 60)

Ganador: E ¿Por qué? E tiene un mayor fitness (60 frente a 45 de B), así que E es el ganador.

**Torneo 3:**

Individuos seleccionados: A(fitness: 85) y D(fitness: 20)

Ganador: A ¿Por qué? A tiene un fitness significativamente más alto (85 frente a 20 de D), por lo que A es el ganador.

**CÓDIGO PYTHON:**

```
import random
import tkinter as tk
from tkinter import ttk, messagebox

class TorneoApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Algoritmo de Selección por Torneo Genético")
        self.root.geometry("900x650")
        self.root.configure(bg="#f0f8ff")

        # Configurar paleta de colores
        self.colores = {
            'fondo': '#f0f8ff',
            'fondo_titulo': '#4682b4',
            'texto_titulo': 'white',
            'fondo_frame': '#e6f2ff',
            'participante': '#5f9ea0',
            'ganador': '#2e8b57',
            'perdedor': '#cd5c5c',
            'boton': '#6495ed',
            'boton_hover': '#4169e1',
            'texto_boton': 'white',
            'vs_color': '#9932cc'
        }

        # Datos iniciales
        self.individuos = {
            "A": 85,
            "B": 45,
            "C": 70,
            "D": 20,
            "E": 60,
            "F": 90
        }

        self.ganadores = []
        self.current_torneo = 0

        # Configurar estilo
        self.setup_style()
```

```
# Crear interfaz
self.create_widgets()

def setup_style(self):
    """Configura los estilos visuales con colores"""
    style = ttk.Style()

    # Configurar tema
    style.theme_use('clam')

    # Estilos personalizados
    style.configure('Titulo.TLabel',
                   font=('Helvetica', 18, 'bold'),
                   background=self.colores['fondo_titulo'],
                   foreground=self.colores['texto_titulo'],
                   padding=10)

    style.configure('Subtitulo.TLabel',
                   font=('Helvetica', 12, 'italic'),
                   background=self.colores['fondo'],
                   foreground='#333333')

    style.configure('Participante.TLabel',
                   font=('Helvetica', 11, 'bold'),
                   background=self.colores['fondo_frame'],
                   foreground=self.colores['participante'])

    style.configure('Ganador.TLabel',
                   font=('Helvetica', 12, 'bold'),
                   background=self.colores['fondo_frame'],
                   foreground=self.colores['ganador'])

    style.configure('Perdedor.TLabel',
                   font=('Helvetica', 11),
                   background=self.colores['fondo_frame'],
                   foreground=self.colores['perdedor'])

    style.configure('TButton',
                   font=('Helvetica', 10, 'bold'),
                   background=self.colores['boton'],
                   foreground=self.colores['texto_boton'],
                   borderwidth=1)

    style.map('TButton',
              background=[('active', self.colores['boton_hover'])])

    style.configure('TFrame', background=self.colores['fondo'])
    style.configure('TLabelframe', background=self.colores['fondo_frame'],
borderwidth=2)
    style.configure('TLabelframe.Label',
background=self.colores['fondo_frame'])

def create_widgets(self):
    """Crea todos los elementos de la interfaz"""
    # Frame principal
    main_frame = ttk.Frame(self.root, padding="20")
    main_frame.pack(fill=tk.BOTH, expand=True)

    # Titulo con color
    titulo_frame = ttk.Frame(main_frame, style='Titulo.TFrame')
```

```

    titulo_frame.grid(row=0, column=0, columnspan=3, sticky='ew', pady=(0,
15))

    ttk.Label(titulo_frame,
              text="🃏 SELECCIÓN POR TORNEO GENÉTICO 🃏",
              style='Titulo.TLabel').pack(fill=tk.X)

    # Panel de configuración
    config_frame = ttk.LabelFrame(main_frame, text="⚙️ Configuración",
padding=15)
    config_frame.grid(row=1, column=0, sticky='nsew', padx=5, pady=5,
ipadx=10, ipady=5)

    ttk.Label(config_frame,
              text="Número de torneos:",
              style='Participante.TLabel').grid(row=0, column=0, padx=5)

    self.num_torneos = tk.IntVar(value=3)
    ttk.Spinbox(config_frame,
                from_=1, to=10,
                textvariable=self.num_torneos,
                width=5,
                font=('Helvetica', 10)).grid(row=0, column=1, padx=5)

    start_btn = ttk.Button(config_frame,
                           text="⟳ Iniciar Torneos",
                           command=self.iniciar_torneos)
    start_btn.grid(row=0, column=2, padx=10)

    # Panel de participantes
    part_frame = ttk.LabelFrame(main_frame, text="👤 Participantes",
padding=15)
    part_frame.grid(row=2, column=0, sticky='nsew', padx=5, pady=5, ipadx=10,
ipady=5)

    # Crear una tabla visual para los participantes
    ttk.Label(part_frame,
              text="Individuo",
              style='Participante.TLabel').grid(row=0, column=0, padx=10,
pady=2)
    ttk.Label(part_frame,
              text="Fitness",
              style='Participante.TLabel').grid(row=0, column=1, padx=10,
pady=2)

    row = 1
    for ind, fit in self.individuos.items():
        ttk.Label(part_frame,
                  text=f'{ind}',
                  style='Participante.TLabel').grid(row=row, column=0,
sticky='w', padx=10)
        ttk.Label(part_frame,
                  text=f'{fit}',
                  style='Participante.TLabel').grid(row=row, column=1,
sticky='w', padx=10)
        row += 1

    # Panel del torneo actual
    self.torneo_frame = ttk.LabelFrame(main_frame, text="⚔️ Torneo Actual",
padding=20)

```

```

        self.torneo_frame.grid(row=1, column=1, rowspan=2, sticky='nsew', padx=5,
pady=5, ipadx=10, ipady=5)

        self.torneo_label = ttk.Label(self.torneo_frame,
                                      text="Presione 'Iniciar Torneos' para
comenzar",
                                      style='Subtitulo.TLabel')
        self.torneo_label.pack(pady=10)

# Contenedor para los participantes del torneo
self.participantes_frame = ttk.Frame(self.torneo_frame)
self.participantes_frame.pack(pady=10)

self.part1_frame = ttk.Frame(self.participantes_frame)
self.part1_frame.pack(side=tk.LEFT, padx=20)

        self.part1_label = ttk.Label(self.part1_frame, text="",
style='Participante.TLabel')
        self.part1_label.pack()

self.vs_label = ttk.Label(self.participantes_frame,
                        text="",
                        font=('Helvetica', 24, 'bold'),
                        foreground=self.colores['vs_color'])
self.vs_label.pack(side=tk.LEFT, padx=10)

self.part2_frame = ttk.Frame(self.participantes_frame)
self.part2_frame.pack(side=tk.LEFT, padx=20)

        self.part2_label = ttk.Label(self.part2_frame, text="",
style='Participante.TLabel')
        self.part2_label.pack()

# Separador
ttk.Separator(self.torneo_frame, orient='horizontal').pack(fill=tk.X,
pady=15)

# Ganador
self.ganador_frame = ttk.Frame(self.torneo_frame)
self.ganador_frame.pack()

ttk.Label(self.ganador_frame,
          text="Ganador:",
          style='Participante.TLabel').pack(side=tk.LEFT)

self.ganador_label = ttk.Label(self.ganador_frame,
                               text="",
                               style='Ganador.TLabel')
self.ganador_label.pack(side=tk.LEFT, padx=10)

# Botón siguiente
self.next_button = ttk.Button(self.torneo_frame,
                             text="▶ Siguiente Torneo",
                             command=self.siguiente_torneo,
                             state=tk.DISABLED)
self.next_button.pack(pady=15)

# Panel de resultados
self.resultados_frame = ttk.LabelFrame(main_frame, text="▣ Resultados",
padding=15)

```

```

        self.resultados_frame.grid(row=3, column=0, columnspan=2, sticky='nsew',
padx=5, pady=5, ipadx=10, ipady=5)

    # Añadir scrollbar
    scrollbar = ttk.Scrollbar(self.resultados_frame)
    scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

    self.resultados_text = tk.Text(self.resultados_frame,
                                    height=8,
                                    wrap=tk.WORD,
                                    font=('Helvetica', 10),
                                    yscrollcommand=scrollbar.set)
    self.resultados_text.pack(fill=tk.BOTH, expand=True)

    scrollbar.config(command=self.resultados_text.yview)

    # Configurar tags para colores en el texto
    self.resultados_text.tag_config('ganador',
foreground=self.colores['ganador'])
    self.resultados_text.tag_config('perdedor',
foreground=self.colores['perdedor'])
    self.resultados_text.tag_config('titulo', font=('Helvetica', 11, 'bold'))

    self.resultados_text.insert(tk.END, "Resultados de los torneos:\n",
'titulo')
    self.resultados_text.config(state=tk.DISABLED)

    # Configurar pesos de filas/columnas
    main_frame.columnconfigure(0, weight=1)
    main_frame.columnconfigure(1, weight=1)
    main_frame.rowconfigure(3, weight=1)

def iniciar_torneos(self):
    """Inicia el proceso de torneos"""
    self.ganadores = []
    self.current_torneo = 0
    self.resultados_text.config(state=tk.NORMAL)
    self.resultados_text.delete(1.0, tk.END)
    self.resultados_text.insert(tk.END, "⚡ Iniciando torneos...\n\n",
'titulo')
    self.resultados_text.config(state=tk.DISABLED)
    self.next_button.config(state=tk.NORMAL)
    self.siguiente_torneo()

def siguiente_torneo(self):
    """Realiza el siguiente torneo"""
    if self.current_torneo >= self.num_torneos.get():
        messagebox.showinfo("🏁 Fin del Torneo", "; Todos los torneos han sido
completados!")
        self.next_button.config(state=tk.DISABLED)
        return

    self.current_torneo += 1
    self.torneo_label.config(text=f"Torneo {self.current_torneo} de
{self.num_torneos.get()}")

    # Seleccionar dos individuos aleatorios
    participantes = random.sample(list(self.individuos.items()), 2)
    ind1, fit1 = participantes[0]
    ind2, fit2 = participantes[1]

```

```

# Mostrar participantes
self.part1_label.config(text=f"Individuo {ind1}\nFitness: {fit1}")
self.part2_label.config(text=f"Individuo {ind2}\nFitness: {fit2}")
self.vs_label.config(text="VS")

# Determinar ganador
if fit1 > fit2:
    ganador = ind1
    perdedor = ind2
elif fit2 > fit1:
    ganador = ind2
    perdedor = ind1
else:
    ganador = random.choice([ind1, ind2])
    perdedor = ind1 if ganador == ind2 else ind2

# Resaltar ganador y perdedor
self.part1_label.config(style='Ganador.TLabel' if ganador == ind1 else
'Perdedor.TLabel')
self.part2_label.config(style='Ganador.TLabel' if ganador == ind2 else
'Perdedor.TLabel')

self.ganadores.append(ganador)
self.ganador_label.config(text=f"{ganador}")

# Actualizar resultados
self.resultados_text.config(state=tk.NORMAL)

self.resultados_text.insert(tk.END, f"Torneo {self.current_torneo}:\n")
self.resultados_text.insert(tk.END, f"  {ind1}({fit1}) vs {ind2}({fit2})\n",
'titulo')
self.resultados_text.insert(tk.END, f"Ganador: {ganador}\n\n", 'ganador')

self.resultados_text.see(tk.END)
self.resultados_text.config(state=tk.DISABLED)

# Deshabilitar botón si es el último torneo
if self.current_torneo == self.num_torneos.get():
    self.next_button.config(state=tk.DISABLED)
    self.resultados_text.config(state=tk.NORMAL)
    self.resultados_text.insert(tk.END, "◆ Proceso completado!\n",
'titulo')
    self.resultados_text.insert(tk.END, f"\nGANADORES FINALES: {',
'.join(self.ganadores)}\n", 'ganador')
    self.resultados_text.config(state=tk.DISABLED)

if __name__ == "__main__":
    root = tk.Tk()
    app = TorneoApp(root)
    root.mainloop()

```

## EJECUCIÓN:

Individuo	Fitness
A	85
B	45
C	70
D	20
E	60
F	90

### Pregunta 1.2 (3 puntos)

**Análisis: ¿Qué individuos tienen mayor probabilidad de ser seleccionados? Explica la relación entre fitness y probabilidad de selección.**

En la **selección por torneo**, la probabilidad de que un individuo sea seleccionado depende de su **fitness** relativo en comparación con los otros individuos en el torneo. Los individuos con un fitness más alto tienen una mayor probabilidad de ser seleccionados, pero no necesariamente con certeza. A continuación se explica con mayor detalle:

- En un **torneo**, dos individuos son seleccionados al azar y se comparan. El individuo con mayor fitness es elegido como ganador, lo que implica que los individuos con un **fitness alto** tienen **mayor probabilidad** de ser seleccionados, ya que tienen más posibilidades de ganar en la comparación.
- Sin embargo, este proceso no garantiza que siempre se seleccione al mejor individuo (por ejemplo, en un torneo con individuos de fitness similar, incluso un individuo con fitness bajo podría ganar por azar).
- **Relación entre fitness y probabilidad de selección:** A mayor fitness, más alta es la probabilidad de que un individuo sea elegido, ya que, al tener un mayor rendimiento (fitness), es más probable que "gane" frente a un individuo con fitness más bajo.

**Pregunta 1.3 (3 puntos)**

**Comparación:** ¿Qué ventajas tiene la selección por torneo sobre simplemente elegir siempre al mejor individuo?

La **selección por torneo** tiene varias ventajas respecto a elegir siempre al mejor individuo, y algunas de las más importantes son:

**1. Mantenimiento de la diversidad genética:**

- Si seleccionamos siempre al mejor individuo, el algoritmo se puede quedar atrapado en una **solución local**. La selección por torneo permite que no siempre se elija al mejor, lo que facilita la **diversidad genética** en la población.
- Al permitir que individuos con fitness inferior puedan ser seleccionados por azar, se fomenta la diversidad y se evita la **convergencia prematura** a una solución subóptima.

**2. Eficiencia computacional:**

- Elegir siempre al mejor individuo requiere evaluar a toda la población en cada iteración. Con la selección por torneo, solo se comparan pares de individuos, lo que es más **rápido y eficiente** en términos computacionales, especialmente en poblaciones grandes.

**3. Exploración más amplia del espacio de soluciones:**

- Al no siempre elegir al mejor individuo, la selección por torneo permite que el algoritmo explore una **mayor variedad de soluciones**. Esto es útil para encontrar nuevas combinaciones de genes que puedan resultar en una mejor solución en el futuro.

**4. Evita la sobreexploración de un solo individuo:**

- Si se elige siempre al mejor individuo, ese individuo puede ser explotado en exceso, limitando la exploración del espacio de soluciones. Con la selección por torneo, incluso los mejores individuos no son seleccionados con certeza, lo que permite una **mejor explotación y exploración** balanceada del espacio de búsqueda.

## ACTIVIDAD 2: CRUCE PMX PARA PERMUTACIONES (8 puntos)

El PMX (Partially-Mapped Crossover) es ideal para problemas donde el orden importa y no se pueden repetir elementos.

**Contexto:** En nuestro problema de distribución, la representación permutacional ordena a los estudiantes: posiciones [0-12] → Examen A, [13-25] → Examen B, [26-38] → Examen C.

### Ejemplo de Cruce PMX

**Padres (8 elementos para simplificar):**

Padre 1:	1	2	3	4	5	6	7	8
Padre 2:	8	7	6	5	4	3	2	1

**Puntos de cruce:** Entre posiciones 3 y 6 (segmento resaltado en rojo)

### Pregunta 2.1 (3 puntos)

**Paso 1:** Identifica el mapeo del segmento intercambiado.

Segmento del Padre 1: 4-5 ↔ Segmento del Padre 2: 5 - 4

Mapeo generado: 4 (P1) ↔ 5 (P2)  
5 (P1) ↔ 4 (P2)

### Pregunta 2.2 (5 puntos)

**Paso 2:** Construye el Hijo 1 aplicando PMX.

a) Copia el segmento del Padre 2 al Hijo 1:

Hijo 1:	-	-	-	5	4	-	-	-
---------	---	---	---	---	---	---	---	---

b) Completa las posiciones restantes usando el mapeo y el Padre 1:

Posición 0: Valor del Padre 1 = 1, ¿está en el segmento? NO

Si está, usar mapeo: - \_\_\_\_\_, sino copiar directamente: - \_\_\_\_\_

Posición 1: Valor del Padre 1 = 2, ¿está en el segmento? NO

Si está, usar mapeo: - \_\_\_\_\_, sino copiar directamente: - \_\_\_\_\_

Posición 2: Valor del Padre 1 = 3, ¿está en el segmento? NO

Si está, usar mapeo: - \_\_\_\_\_, sino copiar directamente: - \_\_\_\_\_

- *Repetimos el mismo patrón para completar todo el Hijo 1*
- *ningún valor reemplazado está en el segmento*

c) **Hijo 1 completo:**

HIJO 1:	1	2	3	5	4	6	7	8
---------	---	---	---	---	---	---	---	---

## CÓDIGO PYTHON:

```

import tkinter as tk
from tkinter import ttk, messagebox

class OXApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Algoritmo de Cruce OX")
        self.root.geometry("950x650")
        self.root.configure(bg="#f5f5f5")

        # Configurar paleta de colores
        self.colores = {
            'fondo': '#f5f5f5',
            'titulo': '#3f51b5',
            'subtitulo': '#2196f3',
            'padre1': '#4caf50',
            'padre2': '#ff5722',
            'hijo': '#9c27b0',
            'segmento': '#ffeb3b',
            'texto': '#212121',
            'borde': '#bdbdbd'
        }

        # Datos iniciales
        self.padre_1 = [1, 2, 3, 4, 5, 6, 7, 8]
        self.padre_2 = [8, 7, 6, 5, 4, 3, 2, 1]
        self.inicio_cruce = 3
        self.fin_cruce = 6

        # Configurar estilo
        self.setup_style()

        # Crear interfaz
        self.create_widgets()

        # Ejecutar cruce inicial
        self.ejecutar_cruce()

    def setup_style(self):
        """Configura los estilos visuales"""
        style = ttk.Style()
        style.theme_use('clam')

        # Estilos personalizados
        style.configure('Titulo.TLabel',
                       font=('Roboto', 16, 'bold'),
                       foreground='white',
                       background=self.colores['titulo'],
                       padding=10)

        style.configure('Subtitulo.TLabel',
                       font=('Roboto', 12, 'bold'),
                       foreground=self.colores['subtitulo'])

        style.configure('Gen.TLabel',
                       font=('Roboto', 14, 'bold'),
                       padding=10,
                       relief='solid',
                       borderwidth=1)

```

```

style.configure('GenPadre1.TLabel',
               background='white',
               foreground=self.colores['padre1'])

style.configure('GenPadre2.TLabel',
               background='white',
               foreground=self.colores['padre2'])

style.configure('GenHijo.TLabel',
               background='white',
               foreground=self.colores['hijo'])

style.configure('Segmento.TLabel',
               background=self.colores['segmento'],
               font=('Roboto', 14, 'bold'))

style.configure('TButton',
               font=('Roboto', 10, 'bold'),
               padding=6)

style.map('TButton',
         background=[('active', '#e0e0e0')])

def create_widgets(self):
    """Crea todos los elementos de la interfaz"""
    # Frame principal
    main_frame = ttk.Frame(self.root, padding="15")
    main_frame.pack(fill=tk.BOTH, expand=True)

    # Título
    ttk.Label(main_frame,
              text="Algoritmo de Cruce OX (Order Crossover)",
              style='Titulo.TLabel').grid(row=0, column=0, columnspan=3,
                                          sticky='ew', pady=(0, 15))

    # Panel de configuración
    config_frame = ttk.LabelFrame(main_frame, text="Configuración",
                                  padding=10)
    config_frame.grid(row=1, column=0, sticky='nsew', padx=5, pady=5)

    ttk.Label(config_frame, text="Padre 1:").grid(row=0, column=0, padx=5,
                                                 pady=5)
    self.padre1_entry = ttk.Entry(config_frame, width=30)
    self.padre1_entry.grid(row=0, column=1, padx=5, pady=5)
    self.padre1_entry.insert(0, ",".join(map(str, self.padre_1)))

    ttk.Label(config_frame, text="Padre 2:").grid(row=1, column=0, padx=5,
                                                 pady=5)
    self.padre2_entry = ttk.Entry(config_frame, width=30)
    self.padre2_entry.grid(row=1, column=1, padx=5, pady=5)
    self.padre2_entry.insert(0, ",".join(map(str, self.padre_2)))

    ttk.Label(config_frame, text="Inicio segmento:").grid(row=2, column=0,
                                                       padx=5, pady=5)
    self.inicio_entry = ttk.Spinbox(config_frame, from_=0,
                                    to=len(self.padre_1)-1, width=5)
    self.inicio_entry.grid(row=2, column=1, padx=5, pady=5, sticky='w')
    self.inicio_entry.set(self.inicio_cruce)

    ttk.Label(config_frame, text="Fin segmento:").grid(row=3, column=0,
                                                       padx=5, pady=5)

```

```

        self.fin_entry = ttk.Spinbox(config_frame, from_=1, to=len(self.padre_1),
width=5)
        self.fin_entry.grid(row=3, column=1, padx=5, pady=5, sticky='w')
        self.fin_entry.set(self.fin_cruce)

        ttk.Button(config_frame,
                   text="Ejecutar Cruce",
                   command=self.ejecutar_cruce).grid(row=4, column=0,
columnspan=2, pady=10)

    # Panel de visualización
    vis_frame = ttk.Frame(main_frame)
    vis_frame.grid(row=2, column=0, sticky='nsew', padx=5, pady=5)

    # Visualización del Padre 1
    ttk.Label(vis_frame, text="Padre 1",
style='Subtitulo.TLabel').grid(row=0, column=0, pady=(0, 10))
    self.padre1_frame = ttk.Frame(vis_frame)
    self.padre1_frame.grid(row=1, column=0)

    # Visualización del Padre 2
    ttk.Label(vis_frame, text="Padre 2",
style='Subtitulo.TLabel').grid(row=0, column=1, pady=(0, 10))
    self.padre2_frame = ttk.Frame(vis_frame)
    self.padre2_frame.grid(row=1, column=1)

    # Visualización del Hijo
    ttk.Label(vis_frame, text="Hijo", style='Subtitulo.TLabel').grid(row=0,
column=2, pady=(0, 10))
    self.hijo_frame = ttk.Frame(vis_frame)
    self.hijo_frame.grid(row=1, column=2)

    # Panel de explicación
    expl_frame = ttk.LabelFrame(main_frame, text="Proceso de Cruce OX",
padding=15)
    expl_frame.grid(row=1, column=1, rowspan=2, sticky='nsew', padx=5,
pady=5)

    self.expl_text = tk.Text(expl_frame, wrap=tk.WORD, height=15, width=40,
font=('Roboto', 10), padx=10, pady=10)
    self.expl_text.pack(fill=tk.BOTH, expand=True)

    # Configurar scrollbar
    scrollbar = ttk.Scrollbar(expl_frame)
    scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
    self.expl_text.config(yscrollcommand=scrollbar.set)
    scrollbar.config(command=self.expl_text.yview)

    # Configurar pesos de filas/columnas
    main_frame.columnconfigure(0, weight=1)
    main_frame.columnconfigure(1, weight=2)
    main_frame.rowconfigure(2, weight=1)

def ejecutar_cruce(self):
    """Ejecuta el algoritmo de cruce OX y actualiza la visualización"""
    try:
        # Obtener valores de entrada
        self.padre_1 = list(map(int, self.padre1_entry.get().split(',')))
        self.padre_2 = list(map(int, self.padre2_entry.get().split(',')))
        self.inicio_cruce = int(self.inicio_entry.get())
        self.fin_cruce = int(self.fin_entry.get())

```

```

# Validar entradas
if len(self.padre_1) != len(self.padre_2):
    raise ValueError("Los padres deben tener la misma longitud")

if self.inicio_cruce >= self.fin_cruce:
    raise ValueError("El inicio del segmento debe ser menor que el
fin")

if self.fin_cruce > len(self.padre_1):
    raise ValueError("El fin del segmento excede la longitud de los
padres")

# Paso 1: Copiar el segmento del Padre 2 al Hijo 1
hijo_1 = [-1] * len(self.padre_1)
hijo_1[self.inicio_cruce:self.fin_cruce] =
self.padre_2[self.inicio_cruce:self.fin_cruce]

# Crear el mapeo entre los genes intercambiados
mapeo = {}
for i in range(self.inicio_cruce, self.fin_cruce):
    mapeo[self.padre_1[i]] = self.padre_2[i]
    mapeo[self.padre_2[i]] = self.padre_1[i]

# Paso 2: Completar el resto del hijo con los genes del Padre 1
for i in range(len(self.padre_1)):
    if self.inicio_cruce <= i < self.fin_cruce:
        continue # Saltar el segmento de cruce

    gen = self.padre_1[i]

    # Resolver conflictos
    while gen in hijo_1[self.inicio_cruce:self.fin_cruce]:
        gen = mapeo[gen]

    hijo_1[i] = gen

# Actualizar visualización
self.actualizar_visualizacion(self.padre_1, self.padre_2, hijo_1)

# Actualizar explicación
self.actualizar_explicacion(hijo_1, mapeo)

except Exception as e:
    messagebox.showerror("Error", f"Entrada inválida: {str(e)}")

def actualizar_visualizacion(self, padre1, padre2, hijo):
    """Actualiza la visualización gráfica de los cromosomas"""
    # Limpiar frames anteriores
    for widget in self.padre1_frame.winfo_children():
        widget.destroy()

    for widget in self.padre2_frame.winfo_children():
        widget.destroy()

    for widget in self.hijo_frame.winfo_children():
        widget.destroy()

    # Mostrar Padre 1
    for i, gen in enumerate(padre1):
        style = 'Gen.TLabel'
        if self.inicio_cruce <= i < self.fin_cruce:
            style = 'Segmento.TLabel'
        ...

```

```

        lbl = ttk.Label(self.padre1_frame, text=str(gen), style=style)
        lbl.grid(row=0, column=i, padx=2)

    # Mostrar Padre 2
    for i, gen in enumerate(padre2):
        style = 'Gen TLabel'
        if self.inicio_cruce <= i < self.fin_cruce:
            style = 'Segmento TLabel'

        lbl = ttk.Label(self.padre2_frame, text=str(gen), style=style)
        lbl.grid(row=0, column=i, padx=2)

    # Mostrar Hijo
    for i, gen in enumerate(hijo):
        style = 'GenHijo TLabel'
        if self.inicio_cruce <= i < self.fin_cruce:
            style = 'Segmento TLabel'

        lbl = ttk.Label(self.hijo_frame, text=str(gen), style=style)
        lbl.grid(row=0, column=i, padx=2)

    def actualizar_explicacion(self, hijo, mapeo):
        """Actualiza el panel de explicación del proceso"""
        self.expl_text.config(state=tk.NORMAL)
        self.expl_text.delete(1.0, tk.END)

        # Paso 1
        self.expl_text.insert(tk.END, "Paso 1: Copiar segmento del Padre 2 al Hijo\n", 'bold')
        self.expl_text.insert(tk.END,
                             f"Se copian los genes del Padre 2 en las posiciones {self.inicio_cruce} a {self.fin_cruce-1}:\n")
        self.expl_text.insert(tk.END, f"Segmento copiado:\n{self.padre_2[self.inicio_cruce:self.fin_cruce]}\n\n")

        # Mapeo
        self.expl_text.insert(tk.END, "Mapeo de genes intercambiados:\n", 'bold')
        for k, v in mapeo.items():
            self.expl_text.insert(tk.END, f"{k} ↔ {v}\n")
        self.expl_text.insert(tk.END, "\n")

        # Paso 2
        self.expl_text.insert(tk.END, "Paso 2: Completar con genes del Padre 1\n", 'bold')
        self.expl_text.insert(tk.END, "Para cada posición fuera del segmento:\n")
        self.expl_text.insert(tk.END, "1. Tomar el gen del Padre 1\n")
        self.expl_text.insert(tk.END, "2. Si el gen ya está en el segmento copiado, usar el mapeo\n")
        self.expl_text.insert(tk.END, "3. Repetir hasta encontrar un gen no presente\n\n")

        # Resultado final
        self.expl_text.insert(tk.END, "Resultado final del cruce OX:\n", 'bold')
        self.expl_text.insert(tk.END, f"{hijo}\n")

    # Configurar tags para formato
    self.expl_text.tag_config('bold', font=('Roboto', 10, 'bold'))

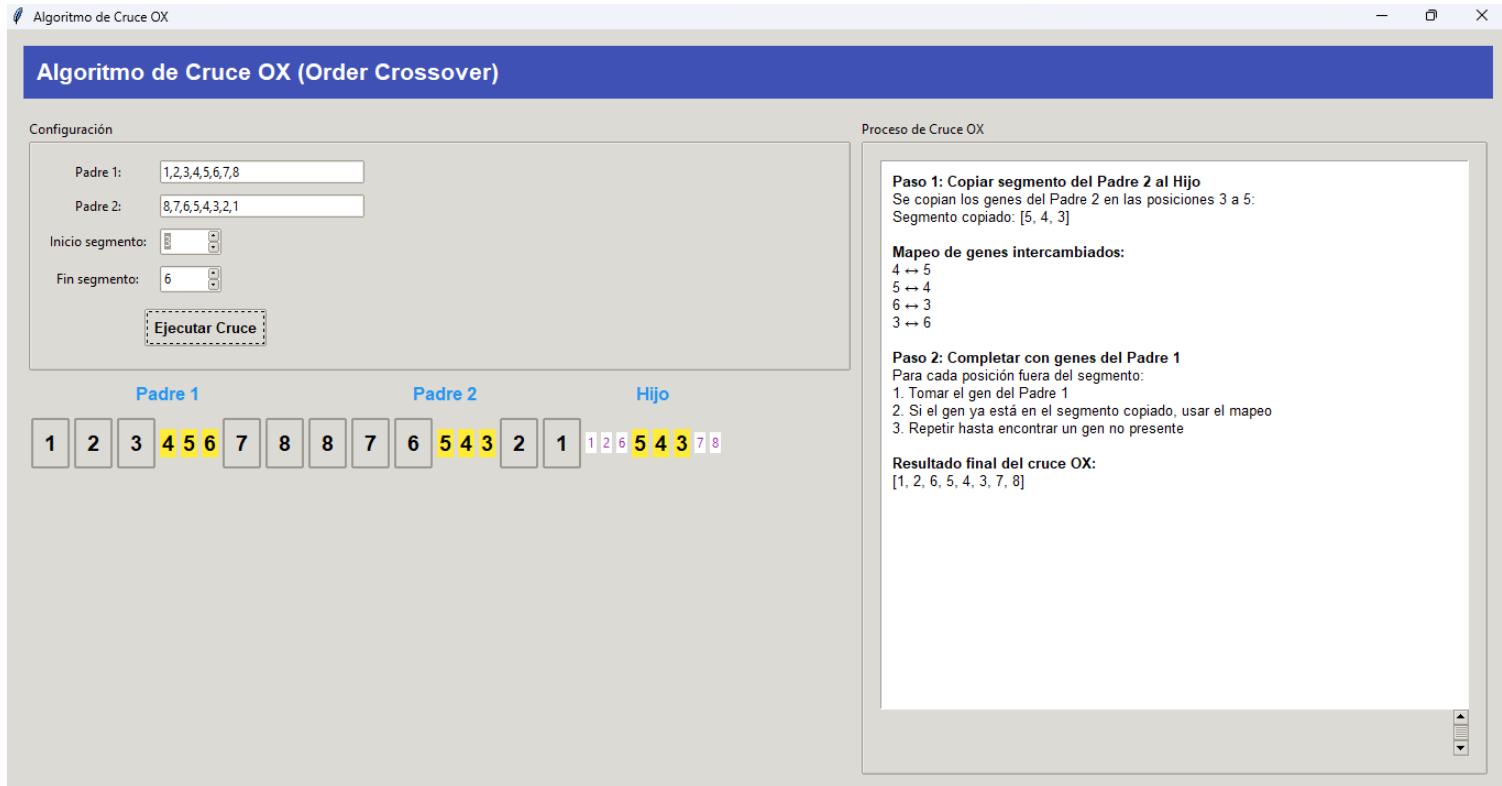
    self.expl_text.config(state=tk.DISABLED)

if __name__ == "__main__":

```

```
root = tk.Tk()
app = OXApp(root)
root.mainloop()
```

## EJECUCIÓN



## ACTIVIDAD 3: ANÁLISIS COMPARATIVO (4 puntos)

### Pregunta 3.1 (2 puntos)

**Selección por Ruleta vs Torneo:** ¿En qué situación preferirías usar selección por torneo en lugar de selección por ruleta?

**Selección por Torneo** tiene ciertas ventajas en situaciones donde:

- Evitar la convergencia prematura:** La selección por torneo permite una mayor diversidad genética en la población al seleccionar a los mejores dentro de un grupo pequeño pero sin elegir siempre al mejor individuo. Esto evita que el algoritmo se quede atrapado en una solución local rápidamente, algo que puede suceder con la selección por ruleta, donde individuos con un fitness muy alto tienen una probabilidad muy grande de ser seleccionados repetidamente.
- Manejo de poblaciones pequeñas:** Si tienes una población pequeña, la selección por torneo puede ser más efectiva para garantizar una buena exploración del espacio de soluciones, ya que no depende exclusivamente de un rango de probabilidades basado en el fitness total, como ocurre en la selección por ruleta.
- Menor sensibilidad a las diferencias de fitness:** En situaciones donde las diferencias de fitness entre los individuos no son muy grandes, la selección por ruleta podría ser muy sensible, favoreciendo demasiado a los individuos de fitness más alto. La selección por torneo es más

robusta en este caso, ya que el proceso de selección está basado solo en pequeños grupos de individuos.

### Pregunta 3.2 (2 puntos)

**Aplicación práctica:** Si tuvieras que resolver un problema del viajante de comercio (TSP) con 20 ciudades, ¿qué representación cromosómica y qué operador de cruce usarías? Justifica tu respuesta.

**Representación:** Permutacional **Cruce:** Cruce OX (Order Crossover) o Cruce PMX (Partially Mapped Crossover)

#### Representación cromosómica: Permutacional

**Justificación:** En el problema del viajante de comercio (TSP), la solución se representa como un recorrido que pasa por cada ciudad una sola vez. Este recorrido tiene un orden específico de las ciudades. La representación permutacional es ideal porque conserva el orden de las ciudades y no permite repeticiones, lo cual es esencial en este problema. Un cromosoma puede ser una secuencia de índices que representan el orden en el que se visitan las ciudades.

#### Operador de cruce: Cruce OX (Order Crossover) o Cruce PMX (Partially Mapped Crossover)

**Justificación:** Para problemas como el TSP, donde el orden de los elementos (en este caso, las ciudades) es crucial, los operadores de cruce deben preservar el orden y evitar repeticiones.

- El cruce OX funciona bien en problemas como el TSP porque intercambia segmentos de los padres mientras mantiene el orden relativo de las ciudades en la descendencia.
- El cruce PMX también es adecuado para el TSP, ya que preserva la validez de las soluciones asegurando que no se repitan ciudades.

## GLOSARIO

**Algoritmo Genético (AG)**

Técnica de optimización inspirada en la evolución natural que usa operadores como selección, cruce y mutación.

**Cromosoma**

Representación codificada de una solución al problema (equivale a un individuo).

**Fitness**      Valor numérico que indica qué tan buena es una solución (aptitud).**Generación**

Conjunto de individuos en un momento específico del algoritmo.

**PMX (Partially-Mapped Crossover)**

Operador de cruce específico para representaciones permutacionales que mantiene la validez de la permutación.

**Población**

Conjunto de todas las soluciones candidatas en una generación.

**Presión Selectiva**

Intensidad con la que se favorece a los individuos más aptos durante la selección.

**Representación Permutacional**

Codificación donde la solución es un ordenamiento de elementos sin repetición.

**Selección por Torneo**

Método que elige padres comparando pequeños grupos de individuos seleccionados al azar.

**Selección por Ruleta**

Método que asigna probabilidades de selección proporcionales al fitness de cada individuo.

**¡Éxito en la actividad!**

*Recuerden: La evolución artificial requiere los mismos principios que la natural: selección y recombinación.*