Universidad Politécnica de Chiapas
Ingeniería en Software
Inteligencia Artificial
Corte 1 – Algoritmos Genéticos

Nombre:	**Daniel Peregrino Pérez**			**Matrícula:**	**223216**
Cuatrimestre	**8**	**Grupo:**	**B**	**Fecha**	**2025.06.12**

1 Emparejamiento

Estrategia

Los individuos se ordenan de mejor a peor, las parejas se forman con el contiguo, es decir, el primero con el segundo, el tercero con el cuarto y así.

Implementación

```
# selection_strategies.py
import numpy as np

def seleccion_por_orden(poblacion, fitness_list):
    """
    Selección por ordenamiento: los individuos se ordenan de mejor a peor fitness,
    las parejas se forman con el contiguo.
    Menor fitness es mejor (para este AG).
    """

    indices_ordenados = np.argsort(fitness_list)

    parejas = []
    num_individuos = len(poblacion)

    for i in range(0, num_individuos - 1, 2):
        idx1 = indices_ordenados[i]
        idx2 = indices_ordenados[i+1]
        parejas.append((poblacion[idx1], poblacion[idx2]))

    if num_individuos % 2 != 0 and num_individuos > 0:
        idx_ultimo = indices_ordenados[num_individuos - 1]
        idx_mejor = indices_ordenados[0]
        parejas.append((poblacion[idx_ultimo], poblacion[idx_mejor]))

    return parejas
```

2 Cruza

Estrategia

Cantidad aleatoria de puntos de cruza, la posición de los puntos de cruza es también aleatoria.

Implementación

```
# crossover_strategies.py
import numpy as np
import random
```

Universidad Politécnica de Chiapas
Ingeniería en Software
Inteligencia Artificial
Corte 1 – Algoritmos Genéticos

Nombre:	**Daniel Peregrino Pérez**			**Matrícula:**	**223216**
Cuatrimestre	**8**	**Grupo:**	**B**	**Fecha**	**2025.06.12**

```
def cruza_puntos_aleatorios(ga_instance, parejas):
    """
    Cruza con cantidad aleatoria de puntos en posiciones aleatorias.
    Todas las parejas seleccionadas se cruzan.
    Requiere ga_instance para acceder a _reparar_duplicados y n_asientos.
    """
    descendencia = []

    for padre1_crom, padre2_crom in parejas:
        max_puntos_posibles = ga_instance.n_asientos - 1

        if max_puntos_posibles < 1:
            num_puntos = 0
        else:
            num_puntos = random.randint(1, max_puntos_posibles)

        if num_puntos == 0 or ga_instance.n_asientos <= 1:
            hijo1_crom = list(padre1_crom)
            hijo2_crom = list(padre2_crom)
        else:
            puntos_cruza = sorted(random.sample(range(1, ga_instance.n_asientos), num_puntos))

            hijo1_crom = list(padre1_crom)
            hijo2_crom = list(padre2_crom)

            actual_padre_es_p1_para_hijo1 = True
            idx_anterior_corte = 0
            for punto_corte in puntos_cruza + [ga_instance.n_asientos]:
                segmento_p1 = padre1_crom[idx_anterior_corte:punto_corte]
                segmento_p2 = padre2_crom[idx_anterior_corte:punto_corte]

                if actual_padre_es_p1_para_hijo1:
                    hijo1_crom[idx_anterior_corte:punto_corte] = segmento_p1
                    hijo2_crom[idx_anterior_corte:punto_corte] = segmento_p2
                else:
                    hijo1_crom[idx_anterior_corte:punto_corte] = segmento_p2
                    hijo2_crom[idx_anterior_corte:punto_corte] = segmento_p1

                actual_padre_es_p1_para_hijo1 = not actual_padre_es_p1_para_hijo1
                idx_anterior_corte = punto_corte

        hijo1_reparado = ga_instance._reparar_duplicados(hijo1_crom, padre1_genes=padre1_crom,
padre2_genes=padre2_crom)
        hijo2_reparado = ga_instance._reparar_duplicados(hijo2_crom, padre1_genes=padre1_crom,
padre2_genes=padre2_crom)
```

Nombre:	**Daniel Peregrino Pérez**			**Matrícula:**	**223216**
Cuatrimestre	**8**	**Grupo:**	**B**	**Fecha**	**2025.06.12**

```
    descendencia.extend([hijo1_reparado, hijo2_reparado])

  return descendencia
```

3 Mutación

Estrategia

Sólo mutan los individuo que no superen el umbral de porcentaje de mutación del individuo PMI. La mutación consiste ordenar aleatoriamente un segmento. Elegir un segmento aleatorio (dado por una posición de inicio y un tamaño de segmento). Ordenar aleatoriamente los genes en el segmento.

Implementación

```
# mutation_strategies.py
import numpy as np
import random

def mutacion_barajar_segmento(ga_instance, poblacion_cromosomas): # Cambiado nombre para claridad
  """
  Mutación: para individuos seleccionados por PMI,
  elegir un segmento aleatorio (inicio y tamaño) y ordenar
  aleatoriamente (barajar) los genes en ese segmento.
  Requiere ga_instance para prob_mutacion_ind y n_asientos.
  """
  poblacion_mutada = []
  for cromosoma in poblacion_cromosomas:
    cromosoma_mutado = list(cromosoma)
    if random.random() < ga_instance.prob_mutacion_ind:
      if ga_instance.n_asientos >= 1:

        tam_segmento = random.randint(1, ga_instance.n_asientos)


        inicio_segmento = random.randint(0, ga_instance.n_asientos - tam_segmento)

        fin_segmento = inicio_segmento + tam_segmento

        segmento_a_barajar = cromosoma_mutado[inicio_segmento:fin_segmento]
        if len(segmento_a_barajar) > 1 :
          random.shuffle(segmento_a_barajar)
          cromosoma_mutado[inicio_segmento:fin_segmento] = segmento_a_barajar


    poblacion_mutada.append(cromosoma_mutado)
```

Nombre:	**Daniel Peregrino Pérez**			**Matrícula:**	**223216**
Cuatrimestre	**8**	**Grupo:**	**B**	**Fecha**	**2025.06.12**

```
return poblacion_mutada
```

4 Poda

Estrategia

Eliminar aleatoriamente, conservando al mejor.

Implementación

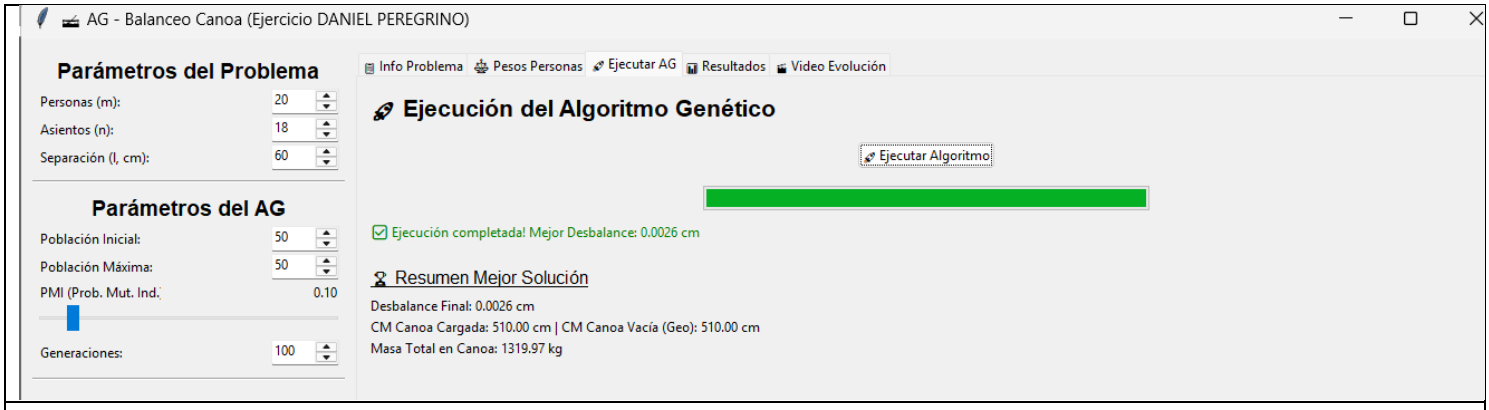
```
# pruning_strategies.py
import numpy as np
import random

def poda_aleatoria_conservando_mejor(poblacion_cromosomas, fitness_list, poblacion_maxima):
    """
    Poda eliminando aleatoriamente, conservando al mejor.
    Mayor fitness es mejor.
    """
    num_actual_individuos = len(poblacion_cromosomas)

    if num_actual_individuos <= poblacion_maxima:
        return list(poblacion_cromosomas)

    idx_mejor = np.argmax(fitness_list)
    mejor_individuo_crom = poblacion_cromosomas[idx_mejor]

    poblacion_sin_mejor = []
    for i, crom in enumerate(poblacion_cromosomas):
        if i != idx_mejor:
            poblacion_sin_mejor.append(crom)

    nueva_poblacion_crom = [mejor_individuo_crom]


    cantidad_a_seleccionar_aleatoriamente = poblacion_maxima - 1

    if cantidad_a_seleccionar_aleatoriamente > 0 and poblacion_sin_mejor:
        # Asegurarse de no pedir más muestras de las que hay
        num_muestras = min(cantidad_a_seleccionar_aleatoriamente, len(poblacion_sin_mejor))

        indices_seleccionados_aleatoriamente = random.sample(range(len(poblacion_sin_mejor)), num_muestras)

        for idx_sel in indices_seleccionados_aleatoriamente:
            nueva_poblacion_crom.append(poblacion_sin_mejor[idx_sel])
```

Nombre:	**Daniel Peregrino Pérez**			**Matrícula:**	**223216**
Cuatrimestre	**8**	**Grupo:**	**B**	**Fecha**	**2025.06.12**

```
return nueva_poblacion_crom
```

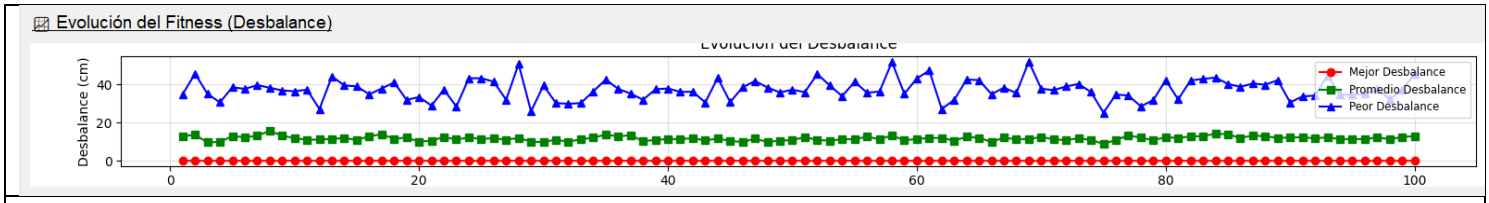
5 Salida

5.1 Reporte del individuo



Mejor individuo

5.2 Evolución de la aptitud



Evolución de la aptitud

5.3 Video de la evolución de la población

Web link al video de la evolución de la población
https://drive.google.com/drive/folders/1HvQ-xlXt-Mh4gVvYPP2wKl7280BydwVg?usp=drive_link