Universidad Politécnica de Chiapas

IA.C3.A1b Reporte de proyecto aplicativo AG

Inteligencia Artificial

2023.C1.08B.IA.C3.A1b

P R E S E N T A N :

203407 Raúl Alejandro Álvarez Calvo
203420 Hiram Emilio Lache Toledo

Docentes

Carlos Alberto Díaz Hernández
Sirgei García Ballinas

Chiapas, cuatrimestre enero - abril de 2023

Resumen

Este reporte documenta las actividades realizadas durante el desarrollo del proyecto de implementación de un algoritmo genetico, incluyendo aspectos como la descripción de la problematica problemática elegida, y las diferentes estrategias derminadas, también se documenta la implementación de las mismas para la resolución del problema, y los resultados obtenidos tras la ejecución del algoritmo.

N.equipo	Nombre	Matrícula
6	Hiram Emilio Lache Toledo	203420
6	Raúl Alejandro Álvarez Calvo	203407

Tabla 1: Integrantes del equipo

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

IA.C3 Proyecto AG

1.1. Problemática

Se diseñó e implementó un algoritmo genético para la resolución de un problema de optimización de costos para un viaje vacacional. El algoritmo tiene como principal objetivo, obtener las mejores combinaciones posibles de vuelos y hoteles, esto tomando en cuenta aspectos como los costos de hotel y boletos de avión, calificaciones y duración del vuelo, realizando este proceso para los distintos destinos que se visitaran.

Esto se realizó mediante una página web, en la cual el usuario puede ingresar valores como: fecha de inicio del viaje, fecha de finalización del viaje, cantidad de personas en el viaje, punto de origen y destinos a visitar, a partir de esto, se obtienen los datos de vuelos y hoteles para la ejecución del algoritmo y la obtención de la combinación más óptima.

Para este problema, se definieron algunos valores como una cantidad de generaciones de 60, una población máxima de 8, y probabilidades de cruza y mutación de 0.6 y 0.4 respectivamente.

1.2. Modelado

entidades

Para la resolución del problema y ejecución del algoritmo se definieron 3 clases de objetos, estos son: Hotel, Transporte y Viaje.

1.2.1. Clase Hotel

¿Por qué la aptitud está asociada al hotel?

Utilizada para almacenar los atributos de un hotel, y a partir de estos, realizar un cálculo de la aptitud del hotel. Los parámetros que contiene esta clase son:

- **ID:** Este valor se utiliza para identificar al hotel en la lista obtenida.
- **Nombre:** Nombre del hotel.
- **Precio:** Costo del hotel de acuerdo al total de días.
- **Calificación:** Calificación del hotel de acuerdo a opiniones de usuarios.
- **Aptitud:** Valor obtenido a partir de un cálculo con los valores de calificación y precio, así como los pesos asignados a cada valor (0.7 y 0.3 para precio y calificación). De forma mas detallada, la función de aptitud consiste en la multiplicación del peso del precio por el valor inverso del precio (de modo que un menor precio signifique mayor aptitud), sumado a la multiplicación del peso de la calificación por la calificación.

```python
class Hotel:
    def __init__(self, id, nombre, calificacion, precio):
        self.id = id
        self.nombre = nombre
        self.calificacion = calificacion
        self.precio = precio
        self.aptitud = (0.7 * (1/float(self.precio)) * (0.3 * float(self.calificacion))) * 10000
```

Figura 1.1: Modelado clase hotel

¿Qué son las constantes 0.3 y 0.7?

1.2.2. Clase Transporte

Utilizada para almacenar los atributos de un vuelo, y a partir de estos, realizar un cálculo de la aptitud del mismo. Los parámetros que contiene esta clase son:

- **ID:** Este valor se utiliza para identificar al vuelo en la lista obtenida.
- **Nombre:** Nombre de la aerolínea.
- **Precio:** Costo del vuelo.
- **Duración:** Duración del vuelo.
- **Aptitud:** Valor obtenido a partir de un cálculo con los valores de duración y precio. De forma mas detallada, la función de aptitud consiste en la multiplicación del peso del precio por el valor inverso del precio (de modo que un menor precio signifique mayor aptitud), sumado a la multiplicación del peso de la duración de viaje por el valor inverso duración del viaje.

```python
class Transporte:
    def __init__(self, id, nombre, duracion, precio):
        self.id = id
        self.nombre = nombre
        self.duracion = duracion
        self.precio = precio
        self.aptitud = (0.675 * (1/float(self.precio)) + 0.325 * (1/(float(self.duracion)))) * 100
```

Figura 1.2: Modelado clase transporte

1.2.3. Clase Viaje

Utilizada para almacenar los atributos de un viaje, y a partir de estos, realizar un cálculo de la aptitud del mismo. Esta clase hace uso de las 2 anteriores, teniendo como parámetros:

- **ID:** Este valor se utiliza para identificar al viaje.
- **Nombre:** Nombre de la ciudad visitada.
- **Transporte:** Objeto de la clase transporte elegido para este viaje.
- **Hospedaje:** Objeto de la clase hotel elegido para este viaje.
- **Generación:** Generación del algoritmo en la que se creó este viaje.
- **Aptitud:** Valor obtenido a partir de un cálculo con los valores de aptitud del hotel y vuelo. Específicamente, la suma de las aptitutes del hotel y transporte, multiplicadas por determinados pesos (0.6 y 0.4 respectivamente)

```python
class Viaje:
    def __init__(self, id, ciudad, transporte, hospedaje, generacion):
        self.id = id
        self.ciudad = ciudad
        self.transporte = transporte
        self.hospedaje = hospedaje
        self.generacion = generacion
        self.aptitud = self.get_aptitud()

    def get_aptitud(self):
        aptitud_transporte = self.transporte.aptitud * 0.4
        aptitud_hotel = self.hospedaje.aptitud * 0.6

        return aptitud_transporte + aptitud_hotel
```

Figura 1.3: Modelado clase viaje

Estas clases se utilizaron en la ejecución del algoritmo, siendo la clase viaje la representación del ïndividuo", y elementos de la clase Hotel y Transporte, siendo propiedades del individuo, utilizados para procesos como cruza y mutación.

1.3. Estrategias

1.3.1. Creación de parejas

Para esta sección, tras probar diferentes estrategias y su funcionamiento, no se definió un método para la selección de parejas muy sofisticado, sino que, simplemente se emparejó a los individuos en la lista de forma secuencia de acuerdo al orden, es decir, el primer individuo se empareja con el segundo y así sucesivamente. Esto se implementó de la siguiente manera:

Que pasa si la cantidad de individuos es impar

```python
def get_parejas(poblacion):
    parejas = []
    for i in range(len(poblacion)):
        if i < len(poblacion) -1:
            parejas.append([poblacion[i], poblacion[i+1]])

    return parejas
```

Figura 1.4: Creación de parejas

1.3.2. Cruza

Para este problema, los elementos a cruzar en cada pareja (en la que los individuos son objetos de la clase viaje) son los valores de hotel y transporte, pues se busca la combinación más óptima de estos para cada destino, por ello, la cruza se definió de modo que estos valores se intercambien y combinen de acuerdo a la pareja elegida, por ejemplo, obteniendo el valor de hotel del primer individuo y el valor de transporte del segundo. La cruza se definió de esta manera debido a que las propiedades de los objetos Hotel y Transporte no pueden cruzarse, pues son propios de cada objeto, tampoco se cruza el orden de los viajes debido a que los datos del transporte dependen directamente de un punto de origen y destino, cambiar el orden no sería adecuado porque, por ejemplo, los precios de vuelos entre ciudades no se corresponderían, pues fueron obtenidos de acuerdo a información distinta. La implementación de la cruza fue la siguiente:

```python
def cruzar(parejas, destino, gen):
    global VIAJES_ID
    hijos = []

    if (random.randint(0, 100)/100) < 0.6:
        for p in parejas:
            hijo1 =  Viaje(VIAJES_ID, destino, p[1].transporte, p[0].hospedaje, gen)
            VIAJES_ID +=1
            hijo2 =  Viaje(VIAJES_ID, destino, p[0].transporte, p[1].hospedaje, gen)
            VIAJES_ID +=1

            hijos.append(hijo1)
            hijos.append(hijo2)

    return hijos
```

Figura 1.5: Cruza

Solo hay un destino

1.3.3. Mutación

En los casos en los que un individuo debía mutar (de acuerdo a la probabilidad de mutación), se definió un valor aleatorio correspondiente a un hotel y transporte en el destino elegido, a partir de este valor, se modifican los valores de hotel y transporte del individuo mutado (el cual es un objeto de la clase Viaje), esto con el objetivo de que puedan existir más combinaciones entre transportes y hoteles. La implementación en código fue la siguiente:

```python
def mutar(hijos, num_datos, vuelos, hoteles):
    global VUELOS_ID, HOTELES_ID
    hijosM = []

    for h in hijos:
        if (random.randint(0, 100)/100) < 0.4:
            num = random.randint(0, num_datos-1)
            t = Transporte(VUELOS_ID, vuelos[0][num], vuelos[2][num], vuelos[1][num])
            VUELOS_ID += 1
            h.set_transporte(t)
            ho = Hotel(HOTELES_ID, hoteles[0][num], hoteles[2][num], hoteles[1][num])
            HOTELES_ID += 1
            h.set_hospedaje(ho)

        hijosM.append(h)

    return hijosM
```

Figura 1.6: Mutación

1.3.4. Poda

Por último, para la poda, como estrategia, se determinó una eliminación aleatoria de los individuos en la población hasta cumplir con el límite de población máxima establecido (siempre manteniendo al individuo más apto, esto se realiza determinando un rango para la eliminación de individuos en el cual se evita la primera posición de la lista de población). La implementación de la poda se realizó de la siguiente manera:

```python
def poda(poblacion):
    while len(poblacion) > MAX_POB:
        pos = random.randint(1, len(poblacion)-1)
        poblacion.pop(pos)

    return poblacion
```

Figura 1.7: Poda

1.4. Resultados

Como principal resultado de la ejecución del algoritmo se obtiene el resultado del viaje con las combinaciones más óptimas, incluyendo la mejor opción de transporte y hotel para cada uno de los destinos del viaje, mostrando sus respectivos datos (precio de hotel y transporte, calificación del hotel, duración del viaje, y la generación en la cual fue obtenido el resultado). Un ejemplo de resultado obtenido tras una ejecución del programa fue el siguiente:

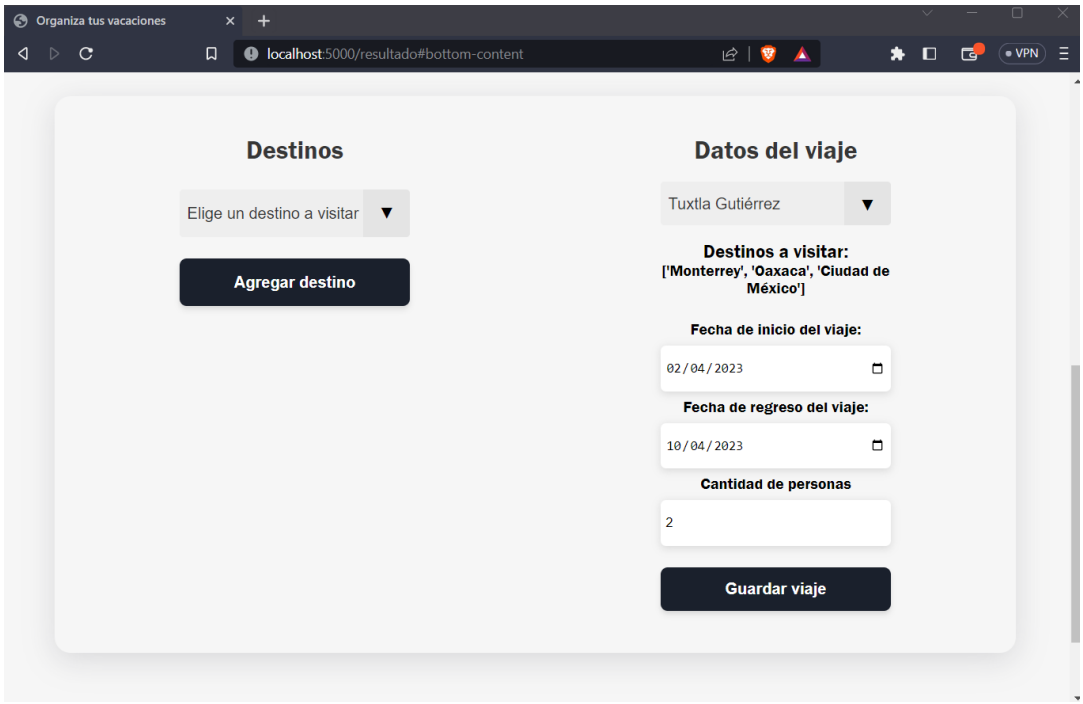

Figura 1.8: Datos de entrada

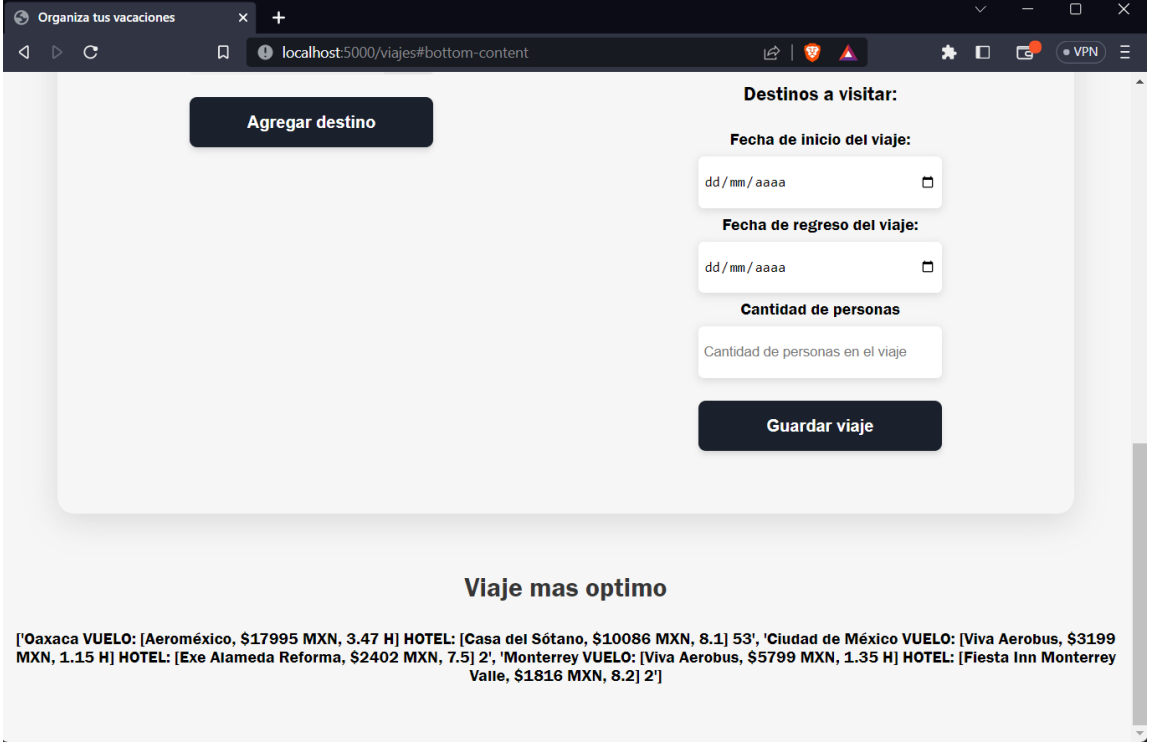

Figura 1.9: Resultado de viaje más apto

Además de la información sobre las mejores combinaciones para el viaje, se generan gráficas de evolución, de aptitud y distribución de individuos por generación (para la generación inicial, intermedia y final) para cada uno de los destinos. Un ejemplo de las gráficas obtenidas para un destino en esta ejecución del programa es el siguiente:

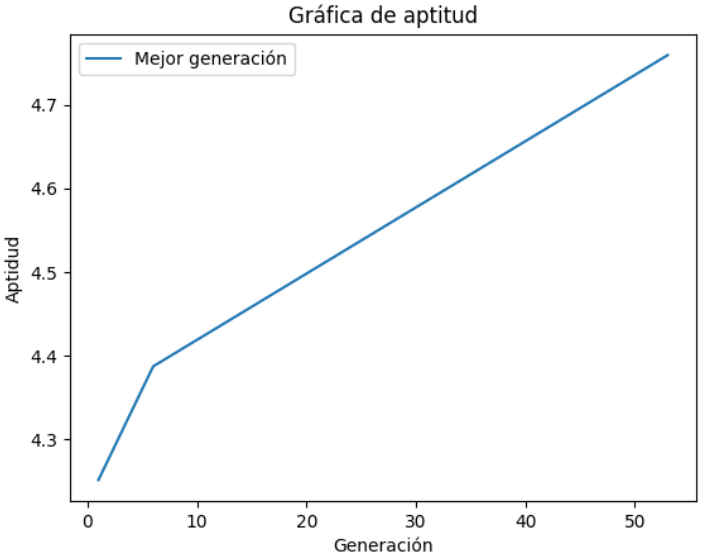

Figura 1.10: Gráfica evolución de aptitud

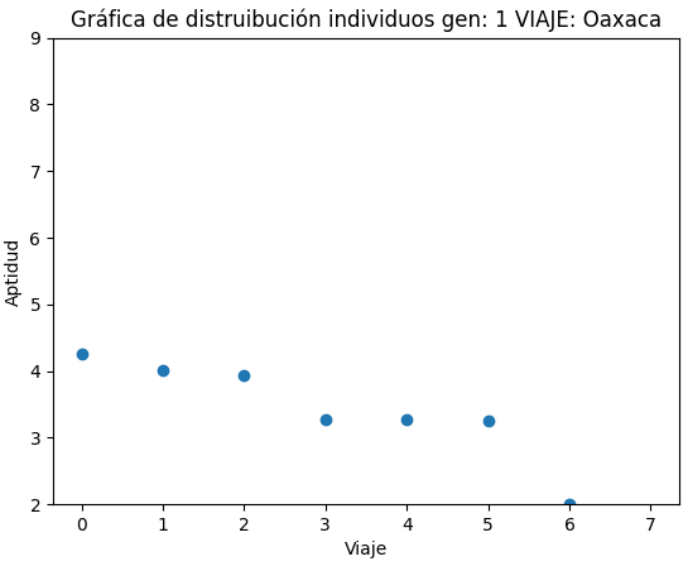

Figura 1.11: Gráfica distribución individuos generación 1

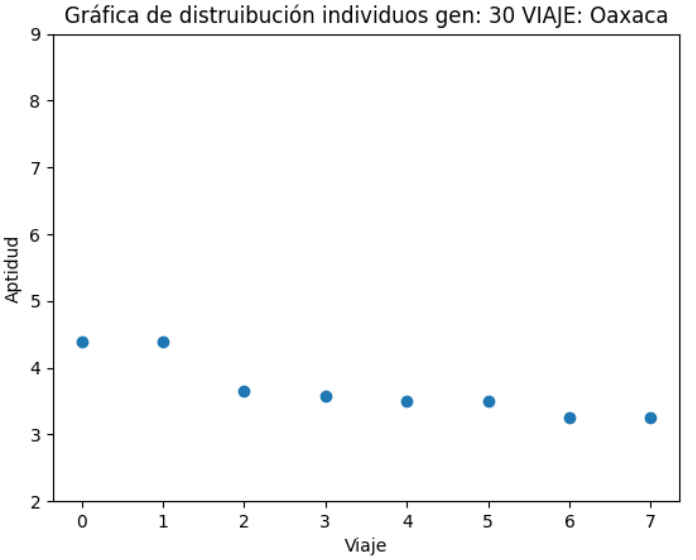

Figura 1.12: Gráfica distribución individuos generación 30

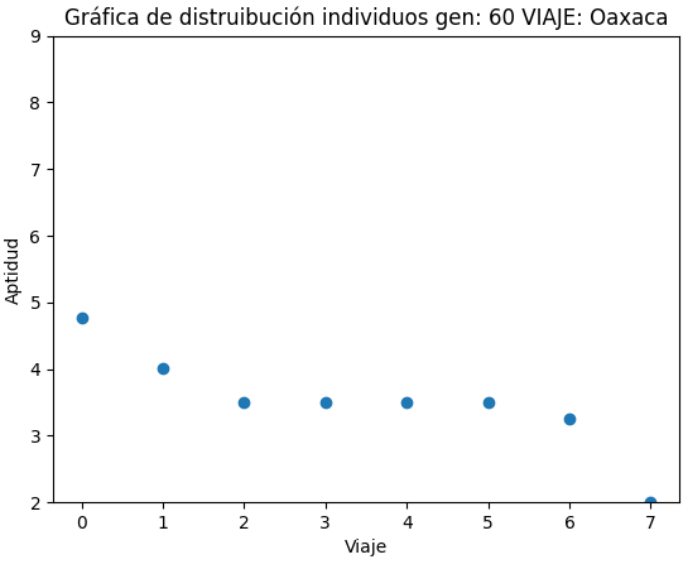

Figura 1.13: Gráfica distribución individuos generación 60

debe ser antes de la poda

Estas últimas gráficas presentan esta distribución debido a que se obtienen tras la poda, y teniendo una población ordenada por aptitud. Todas estas gráficas se generan para cada ciudad, por lo que, ante una lista más grande de ciudades, los resultados serán más extensos.

Un aspecto más a considerar en los obtención de resultados, es que debido a que los datos se obtienen con cada ejecución, y la población inicial se encuentra determinada de forma aleatoria, los resultados pueden tener diversas variaciones, por ejemplo, en algunos casos encontrando el mejor resultado en generaciones muy tempranas, o en algunas de las últimas.

1.5. Comentarios finales

Como conclusión de esta práctica, podriamos mencionar la utilidad de la misma pues, al definir nuestra propia propuesta, nos permitió identificar por nuestra cuenta una aplicación útil para un algoritmo genetico, y a partir de esto, aplicar nuestros conocimientos para la implementación del mismo, lo cual fue una practica útil, pues si bien ya se habia realizado anteriormente, el proceso de razonar e identificar las mejores estrategias posibles y adaptarlas de acuerdo al problema elegido nos ha ayudado a tener mejor entendimiento y habilidad para el futuro desarrollo de estos proyectos.

A lo largo del desarrollo se presentaron algunos problemas, principalmente para la obtención de los datos, o incluso, el uso de estrategias que resultaban en un funcionamiento menos optimo, sin embargo, estos problemas consiguieron solucionarse y se consiguió el resultado deseado en la implementación.