HAL 9000 +
Algoritmo evolutivo

Abraham Morales Hernández 189507
Jorge Esteban Ramírez Sashida 201530
Andrea Martínez Muñoz 200777

Grupo: 002
Curso: Otoño 2023
Profesor: Andrés Gómez de Silva Garza

Introducción

El objetivo principal de este proyecto es poner en práctica lo aprendido en el curso de Inteligencia Artificial referente a los algoritmos evolutivos. Se hizo uso del lenguaje Python para escribir un programa que pudiera abrirse usando Spyder y que estuviera comentado adecuadamente para entender su funcionamiento y diseño. Asimismo, el programa escrito tenía que estar listo para ejecutarse.

Para diseñar e implementar un algoritmo evolutivo que asignara el calendario de partidos a una liga de baloncesto profesional (NBA).

La función heurística utilizada se definió con base al conocimiento que teníamos acerca de la NBA. De esa manera, decidimos que la mejor función heurística era aquella que balanceaba el número partidos de local y visitante de cada equipo. Por último, el algoritmo evolutivo se seleccionó para generar calendarios de mayor calidad

Desarrollo

Este código presenta una implementación orientada a objetos para modelar equipos de la NBA y sus respectivas ciudades. Para empezar, se definieron las clases principales que se iban a usar a lo largo del código: “Team” y “City” (junto con sus atributos como equipos), “Match” y “Calendar”.

La clase “Team`” encapsula la información fundamental de un equipo de la NBA, incluyendo su nombre y la ciudad a la que pertenece. Esta última está representada por la clase “City”, que no solo almacena el nombre de la ciudad, sino también sus coordenadas geográficas de latitud y longitud.

Por otro lado, la clase “Match” modela un enfrentamiento entre dos equipos, especificando tanto al equipo local como al visitante. La ubicación del partido se define por la ciudad del equipo local. La función “location” devuelve las coordenadas geográficas de la ciudad anfitriona del partido.

Además, el concepto del calendario de juegos se encapsula en la clase “Calendar”. Esta clase almacena una lista de objetos “Match” que representan la programación de los partidos. Además, se asocia un puntaje (“grade”) con el calendario, lo que permite la posibilidad de evaluar y clasificar la calidad del programa de juegos.

```python
class Match:
    """
    Represents a match between two teams.

    Attributes:
    - home (Team): The home team.
    - visitor (Team): The visiting team.
    - city (City): The City object where the match takes place.
    """

    def __init__(self, home, visitor):
        """Initialize a Match object with home and visitor teams."""
        self.home = home
        self.visitor = visitor
        self.city = home.city

    def location(self):
        """Get the coordinates of the match location."""
        return self.city.coords

    def __repr__(self):
        """Representation of the Match object."""
        return self.home.name + " at " + self.visitor.name


class Calendar:
    """
    Represents a schedule calendar.

    Attributes:
    - schedule (list): A schedule containing Match objects.
    - grade (float): The grade assigned to the schedule.
    """

    def __init__(self, schedule, grade):
        """Initialize a Calendar object with a schedule and grade."""
        self.schedule = schedule
        self.grade = grade
```

Con respecto a la representación de las ciudades y sus equipos primero se definió un diccionario llamado “cities_info”, que asocia nombres de ciudades con sus respectivas coordenadas geográficas de latitud y longitud. Luego, cada ciudad se asocia con un objeto “City” que almacena su nombre y coordenadas. Se definió esta estructura para facilitar la asociación de ciudades con equipos. Al mismo tiempo, se definió la variable “nba_teams_info” en donde cada elemento de esta lista es un diccionario que contiene el nombre del equipo y una referencia a la ciudad correspondiente.

```
cities = {city_name: City(city_name, lat, long) for city_name, (lat, long) in cities_info.items()}

# Define NBA teams with their information and respective cities
nba_teams_info = [
    {"name": "Hawks", "city": cities["Atlanta"]},
    {"name": "Celtics", "city": cities["Boston"]},
    {"name": "Nets", "city": cities["Brooklyn"]},
    {"name": "Hornets", "city": cities["Charlotte"]},
    {"name": "Bulls", "city": cities["Chicago"]},
    {"name": "Cavaliers", "city": cities["Cleveland"]},
    {"name": "Pistons", "city": cities["Detroit"]},
    {"name": "Pacers", "city": cities["Indianapolis"]},
    {"name": "76ers", "city": cities["Philadelphia"]},
    {"name": "Raptors", "city": cities["Toronto"]},
    {"name": "Wizards", "city": cities["Washington"]},
    {"name": "Mavericks", "city": cities["Dallas"]},
    {"name": "Nuggets", "city": cities["Denver"]},
    {"name": "Warriors", "city": cities["San Francisco"]},
    {"name": "Rockets", "city": cities["Houston"]},
    {"name": "Clippers", "city": cities["Los Angeles"]},
    {"name": "Lakers", "city": cities["Los Angeles"]},
    {"name": "Grizzlies", "city": cities["Memphis"]},
    {"name": "Pelicans", "city": cities["New Orleans"]},
    {"name": "Thunder", "city": cities["Oklahoma City"]},
    {"name": "Suns", "city": cities["Phoenix"]},
    {"name": "Trail Blazers", "city": cities["Portland"]},
    {"name": "Kings", "city": cities["Sacramento"]},
    {"name": "Spurs", "city": cities["San Antonio"]},
    {"name": "Jazz", "city": cities["Salt Lake City"]},
    {"name": "Bucks", "city": cities["Milwaukee"]},
    {"name": "Heat", "city": cities["Miami"]},
    {"name": "Knicks", "city": cities["New York"]},
    {"name": "Magic", "city": cities["Orlando"]},
    {"name": "Timberwolves", "city": cities["Minneapolis"]}

]

# Create Team objects
nba_teams = [Team(info["name"], info["city"]) for info in nba_teams_info]
```

Posteriormente se utilizaron las funciones “get_column” y “transpose_matrix” para manipular matrices y obtener columnas específicas.

La función “get_column” devuelve una columna específica de una matriz. Se hizo uso de esta variable para extraer columnas de la matriz del calendario de partidos. Además, la función “transpose_matrix” toma una matriz como entrada y devuelve su transpuesta. Esta función se utiliza para transformar el calendario de partidos generado inicialmente en un formato más manejable, donde cada columna representa los partidos de una semana.

Sin embargo, la función principal es “generate_schedule”, que recibe una lista de objetos “Team” y para garantizar que el número de equipos sea par, se agrega un equipo ficticio llamado “BYE”’ si el número original es impar. Luego, los equipos se mezclan aleatoriamente para garantizar variabilidad en la programación.

La función utiliza combinaciones de equipos para generar la primera mitad del calendario de partidos. Luego, crea la segunda mitad invirtiendo los equipos en cada partido. Esto garantiza que cada equipo se enfrente a todos los demás exactamente una vez.

Se crea una matriz de objetos “Match” para representar los partidos de cada semana, y se mezclan aleatoriamente. Finalmente, la matriz se transpone para organizar los partidos por semana en lugar de por equipo.

```python
def generate_schedule(teams):
    """
    Generates a schedule for given NBA teams.

    Args:
    - teams (list): List of Team objects representing NBA teams.

    Returns:
    - list of lists: Transposed schedule matrix of Match objects.
    """

    # In case of an odd number of teams, add a fictional 'BYE' team
    if len(teams) % 2 == 1:
        teams.append(Team("BYE", None))

    random.shuffle(teams)  # Shuffle the teams randomly

    # Generate combinations of teams for the first and second halves of the schedule
    first_half = [list(comb) for comb in combinations(teams, 2)]
    second_half = [[match[1], match[0]] for match in first_half]
    matches = first_half + second_half  # Combine the first and second half matches

    num_weeks = int(len(teams)-1) * 2  # Calculate the number of weeks in the schedule
    num_teams = len(teams)  # Total number of teams

    schedule = []  # Initialize the schedule list to hold matches for each week

    for j in range(num_weeks):
        week = []
        for i in range(num_teams // 2):
            # Calculate home and visitor teams for each week
            home = (j + i) % (num_teams - 1)
            visitor = (j + num_teams - i - 1) % (num_teams - 1)
            if i == 0:
                visitor = num_teams - 1

            # Assign teams to matches based on the week
            if j < num_weeks // 2:
                week.append((teams[home], teams[visitor]))
            else:
                week.append((teams[visitor], teams[home]))

        schedule.append(week)  # Append the week's matches to the schedule
```

La función “schedule_to_dataframe” utiliza “pandas” para convertir la matriz del calendario de partidos en un “DataFrame”. Cada columna del “DataFrame” representa una semana y contiene la información de los partidos programados para esa semana.

En cambio, la función “write_schedule_to_file” toma la matriz del calendario de partidos y la escribe en un archivo de texto y la función “total_distance” calcula la distancia total recorrida por los equipos en el calendario de partidos. Así, la función “home_visitor_differential” evalúa la diferencia entre los juegos en casa y los juegos como visitantes para cada equipo en el calendario. Examina cada semana del calendario para contar la cantidad de juegos en casa y como visitante, luego calcula la diferencia total.

La función heurística “heuristic” está basada en atributos del calendario, todo con el objetivo de mejorar la calidad del calendario generado. “Heuristic” calcula un valor heurístico para

un calendario dado y este valor se basa en dos componentes principales: la distancia total recorrida por los equipos (“distance”) y la diferencia entre los juegos en casa y como visitante (“dif”). Ambos valores se normalizan y se combinan para formar el valor heurístico final.
Aunado a esto, se realizó la función “mutation1” que realiza una operación de mutación en la matriz del calendario de partidos. Esta mutación implica la selección aleatoria de un subconjunto de columnas (semanas) y su posterior mezcla. La cantidad de columnas a intercambiar se controla mediante el parámetro “max_columns_to_swap”.
En cambio, la función “mutation2” realiza otra operación de mutación en la matriz del calendario de partidos. En este caso, se selecciona aleatoriamente un número de columnas y filas para intercambiar los atributos “visitor” y “home” en los objetos “Match”.

Por último, para lograr un algoritmo evolutivo funcional que encontrará el mejor calendario de partidos posible para equipos de la NBA se creó la función “evolutionary_algorithm”. Se decidido que esta función generara una población inicial de calendarios de partidos utilizando la función “generate_schedule” y se almacenara en la lista “population”. A través de la función “heuristic” se itera sobre la población inicial y se evalúa el grado de cada calendario. Se realizan diferentes nivel de iteraciones del algoritmo evolutivo para mejorar gradualmente la calidad del calendario y en cada iteración:

- Se seleccionan aleatoriamente dos subconjuntos de la población (“m1” y “m2”).
- Se aplica “mutation1” a los calendarios de los subconjuntos “m1”.
- Se aplica “mutation2” a los calendarios de los subconjuntos “m2”.
- Se genera un nuevo calendario aleatorio.
- Los nuevos calendarios se evalúan utilizando la heurística y se crean objetos “Calendar” para cada uno.
- Los nuevos calendarios y los calendarios existentes se combinan en la población.
- La población se ordena en función de los grados en orden descendente, y se seleccionan los mejores calendarios.

A continuación, el mejor calendario final se selecciona de la población y se almacena en “best_schedule”, se imprime el número de iteración y el mejor grado en cada iteración y se retorna una tupla que contiene la matriz del mejor calendario de partidos y su grado asociado. Finalmente, el mejor calendario se convierte a un DataFrame.

Index	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
0	Kings at Grizzlies	Rockets at Grizzlies	76ers at Grizzlies	Mavericks at Grizzlies	Pelicans at Grizzlies	Pistons at Grizzlies	Hornets at Grizzlies
1	Warriors at Hornets	Lakers at Timberwolves	Bulls at Knicks	Hornets at Bucks	Trail Blazers at Pacers	Jazz at Raptors	Kings at Mavericks
2	Celtics at Mavericks	Nets at Nuggets	Heat at Spurs	Kings at Wizards	Jazz at Magic	Trail Blazers at Wizards	Warriors at Bucks
3	Bucks at Cavaliers	Spurs at Suns	Hawks at Nets	Warriors at Raptors	Clippers at Pistons	Bucks at Pelicans	Wizards at Celtics
4	Hawks at Wizards	Bulls at Thunder	Timberwolves at Cavaliers	Celtics at Pistons	Thunder at Raptors	Mavericks at Pacers	Raptors at Cavaliers
5	Heat at Raptors	76ers at Clippers	Rockets at Celtics	Jazz at Cavaliers	Suns at Wizards	Magic at Hornets	Hawks at Pistons
6	Knicks at Pistons	Magic at Knicks	Lakers at Warriors	Hawks at Trail Blazers	Nuggets at Bucks	Clippers at Kings	Heat at Jazz
7	Jazz at 76ers	Heat at Pacers	Nuggets at Kings	Heat at Pelicans	Mavericks at Lakers	Thunder at Warriors	Trail Blazers at Knicks
8	Trail Blazers at Bulls	Hawks at Pelicans	Hornets at Suns	Pacers at Knicks	Rockets at Hornets	Celtics at Suns	Pelicans at 76ers
9	Pelicans at Spurs	Trail Blazers at Cavaliers	Thunder at Mavericks	76ers at Magic	Timberwolves at Kings	Cavaliers at Nuggets	Pacers at Bulls
10	Pacers at Nets	Celtics at Jazz	Clippers at Bucks	Bulls at Clippers	Warriors at Nets	Lakers at Hawks	Spurs at Magic
11	Magic at Timberwolves	Pistons at Warriors	Wizards at Magic	Spurs at Thunder	Spurs at Celtics	Heat at Rockets	Clippers at Nets
12	Rockets at Clippers	Kings at Raptors	Raptors at Pacers	Nets at Suns	Bulls at Cavaliers	Knicks at Timberwolves	Thunder at Timberwolves
13	Lakers at Thunder	Wizards at Hornets	Pistons at Pelicans	Nuggets at Timberwolves	76ers at Hawks	76ers at Nets	Rockets at Suns
14	Nuggets at Suns	Mavericks at Bucks	Trail Blazers at Jazz	Rockets at Lakers	Heat at Knicks	Spurs at Bulls	Nuggets at Lakers

Conclusión

La implementación de este proyecto nos reiteró la importancia de planear antes de ejecutar la parte práctica. El estructurar y organizar la problemática desde un inicio nos dio una visión clara de cómo queríamos hacer el proyecto y cuál era la mejor estrategia para lograrlo. También, reconocimos la importancia de planificar las tareas correspondientes a cada integrante.

En lo referente al algoritmo creemos que nos faltó tiempo para poder desarrollarlo mejor. De esa manera, el proyecto ser mucho más compacto. Sin embargo, estamos orgullosos y satisfechos con nuestro resultado.

Las limitaciones de nuestro algoritmo están muy apegadas al poder computacional al que tenemos acceso. El programa tarda mucho en correr lo que dificulta hacer muchas pruebas. Además, trabajamos con una liga de muchos equipos y por ende muchas rondas (30 equipos y 58 jornadas). Este algoritmo nada más funciona con ligas que generan sus calendarios en formato todos contra todos y con un número constante de las veces que cada equipo tiene que jugar contra los otros. Es decir, no funciona con ligas como la NFL, en la que algunos equipos deben jugar dos veces contra sus rivales divisionales pero nada más una o cero veces contra otros equipos.

Por último, reconocemos que lo aprendido en el curso de Inteligencia Artificial nos ayudará para el resto de nuestra carrera académica, tanto como experiencia para vida laboral. Tanto este proyecto, como los anteriores, han sido de bastante aprendizaje. De igual manera, han brindado formación y preparación a cada uno de nosotros.

Bibliografías

https://www.sportingnews.com/mx/nba?gr=www

https://es.wikipedia.org/wiki/National_Basketball_Association