Tecnológico de Monterrey
Campus Guadalajara

Optimización de distribución de bicicletas para MiBici

Diseño de algoritmos matemáticos bioinspirados Grupo 201

Alejandra Velasco Zárate A01635453
José Antonio Juárez Pacheco A00572186
José Carlos Yamuni Contreras A01740285
Juan Manuel Hernández Solano A00572208

Octubre 2023

1. Introducción

El crecimiento de las ciudades modernas ha planteado desafíos significativos en términos de movilidad sostenible y eficiencia en el transporte público. En este contexto, los sistemas de bicicletas compartidas han surgido como una solución prometedora para fomentar la movilidad urbana sostenible al proporcionar un medio de transporte ecológico, económico y saludable. Sin embargo, el éxito de estos sistemas depende en gran medida de la disponibilidad y accesibilidad de las bicicletas en las estaciones de servicio, lo que a su vez requiere una gestión óptima de la distribución de bicicletas en toda la red.

En particular, el sistema de bicicletas compartidas "MiBici"ha experimentado un rápido crecimiento en los últimos años, sirviendo como un modelo ejemplar de movilidad urbana sostenible en nuestras ciudades. Para garantizar la disponibilidad constante de bicicletas en las estaciones y, al mismo tiempo, minimizar los costos operativos, es esencial desarrollar estrategias eficientes de redistribución de bicicletas. Este desafío se complica aún más debido a la variabilidad de la demanda en diferentes áreas de la ciudad y las limitaciones de recursos.

2. Objetivo

Implementar diferentes algoritmos bioinspirados para optimizar la distribución de bicicletas por estación en el servicio MiBici del Gobierno de Jalisco.

3. Descripción de los datos

Los datos utilizados provienen del servicio de transporte público MiBici. Sus datos están públicos, por lo que son accesibles para toda la población. Constan de 9 bases de datos donde en todas viene la información de la fecha y hora que empezó un viaje en bicicleta, de la estación de origen, la estación de destino y la hora y fecha en la que terminó el recorrido. Cada base de datos mostraba los registros por mes, por lo que la primera base de datos es de enero del 2023, la segunda de febrero del 2023, así hasta llegar a septiembre del 2023. Lo que se hizo para manejar las distintas bases de datos, fue concatenarlas para tener todos los viajes de los 9 meses en una sola base de datos.

4. Demanda por estación

Para calcular la demanda, primeramente se agrupó por mes y se filtró por estación. Después por cada hora de todos los días del mes, se obtuvo un promedio de los viajes de origen de cada estación así como un promedio de los viajes destino por estación. Teniendo ambos promedios se realizó un promedio de promedios para saber la demanda de cada estación de cada mes. Teniendo la demanda, se pueden aplicar distintos algoritmos genéticos para optimizar la distribución.

5. Algoritmos bioinspirados

Se llevaron a cabo 3 algoritmos genéticos y con base en sus resultados se seleccionó el que tuviera mejor rendimiento a la hora de asignar bicicletas.

5.1. Algoritmos genéticos (GA)

```
# -*- coding: utf-8 -*-
#-------------------------------------------------------------------------

# ALGORITMO GENTICO

#-------------------------------------------------------------------------
```

```
demanda = [5,5,5,5,5,5,5,5,12,5,5,5,5,5,5,5,5,5,6,5,5,5,5,5,5,6,5,5,5,5,5,7,11,5,
5,7,5,5,5,5,5,5,5,5,5,5,5,5,10,19,11,5,10,5,6,5,7,5,5,5,5,5,5,5,6,6,5,5,6,5,5,5,5,5,
8,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,6,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,8,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,7,5,5,5,5,5,5,5,5,5,5,11,5,6,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5]

import pandas as pd
from math import ceil

n = 300
miu = 5.5

def generar_individuos(n, miu):
    numeros_aleatorios = np.random.normal(miu, 1.0, n)
    #numeros_aleatorios = np.clip(numeros_aleatorios, lower_bound, upper_bound)
    return list(map(int, numeros_aleatorios))

import random
import numpy as np

# Datos de ejemplo: estaciones y demanda
num_estaciones = 300
demanda_estaciones = demanda[:] #[random.randint(1, 15) for _ in range(num_estaciones

print(demanda_estaciones)
print(sum(demanda_estaciones))
# Par metros del algoritmo gen tico
num_generaciones = 100
tamano_poblacion = 50
prob_mutacion = 0.1

limite_bicicletas = 1500#1600#3200  # L mite de bicicletas

elitismo_ratio = 0.1  # Porcentaje de individuos el tistas a conservar

def inicializar_poblacion(tamano_poblacion, num_estaciones, limite_bicicletas):
    poblacion = []

    for _ in range(tamano_poblacion):
        asignacion = generar_individuos(300, 5.5)
        while sum(asignacion) != limite_bicicletas:
            asignacion = generar_individuos(300, 5.5)
        poblacion.append(asignacion)
    return poblacion

def evaluar_poblacion(poblacion, demanda_estaciones, limite_bicicletas):
    evaluaciones = []
    for asignacion in poblacion:
        exceso_bicicletas = sum(asignacion) - limite_bicicletas
        evaluacion = sum(np.abs(np.array(asignacion) - np.array(demanda_estaciones)))
        if exceso_bicicletas > 0:
            evaluacion += exceso_bicicletas  # Penaliza el exceso de bicicletas
```

```
        evaluaciones.append(evaluacion)
    return evaluaciones


def seleccionar_mejores_padres(poblacion, evaluaciones, num_padres):
    padres = [poblacion[i] for i in np.argsort(evaluaciones)[:num_padres]]
    return padres

def cruzar_padres(padre1, padre2):
    punto_cruce = random.randint(1, len(padre1) - 1)
    hijo1 = padre1[:punto_cruce] + padre2[punto_cruce:]
    hijo2 = padre2[:punto_cruce] + padre1[punto_cruce:]
    return hijo1, hijo2

def mutar(individuo, prob_mutacion):
    for i in range(len(individuo)):
        if random.random() < prob_mutacion:
            individuo[i] = random.randint(1, 20)
    return individuo

# Algoritmo gen tico
poblacion = inicializar_poblacion(tamano_poblacion, num_estaciones, limite_bicicletas)

for generacion in range(num_generaciones):
    evaluaciones = evaluar_poblacion(poblacion, demanda_estaciones, limite_bicicletas)
    mejores_padres = seleccionar_mejores_padres(poblacion, evaluaciones, tamano_pobla

    # Conservar una parte de los mejores individuos (elitismo)
    num_elitistas = int(tamano_poblacion * elitismo_ratio)
    poblacion_elitista = poblacion[:num_elitistas]

    nueva_generacion = []
    while len(nueva_generacion) < tamano_poblacion - num_elitistas:
        padre1, padre2 = random.sample(mejores_padres, 2)
        hijo1, hijo2 = cruzar_padres(padre1, padre2)
        hijo1 = mutar(hijo1, prob_mutacion)
        hijo2 = mutar(hijo2, prob_mutacion)

        if sum(hijo1) > limite_bicicletas:
            hijo1 = mutar(hijo1, prob_mutacion)
        if sum(hijo2) > limite_bicicletas:
            hijo2 = mutar(hijo2, prob_mutacion)

        nueva_generacion.extend([hijo1, hijo2])

    # Combinar la poblaci n elitista con la nueva generaci n
    poblacion = poblacion_elitista + nueva_generacion

mejor_asignacion = poblacion[np.argmin(evaluaciones)]
print("Mejor asignaci n de bicicletas en estaciones:", mejor_asignacion)

print(sum(demanda_estaciones) - sum(mejor_asignacion))

print(sum(mejor_asignacion))
```

5.2. Optimización por enjambre de partículas (PSO)

```
# -*- coding: utf-8 -*-
#-------------------------------------------------------------------------------

# PSO

#-------------------------------------------------------------------------------

demanda = [5,5,5,5,5,5,5,5,12,5,5,5,5,5,5,5,5,5,6,5,5,5,5,5,6,5,5,5,5,5,7,11,5,
5,7,5,5,5,5,5,5,5,5,5,5,5,10,19,11,5,10,5,6,5,7,5,5,5,5,5,5,6,6,5,5,6,5,5,5,5,5,
8,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,7,5,5,5,5,5,5,5,5,5,5,
5,5,5,6,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,8,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,7,5,5,5,5,5,5,5,5,5,5,11,5,6,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5]

import random
import numpy as np

# Número de estaciones y límite de bicicletas
num_estaciones = 300
limite_bicicletas = 1500

# Crear una lista de demanda de estaciones (reemplaza los valores)
demanda_estaciones = demanda

# Parámetros de PSO
num_particulas = 50
num_dimensiones = num_estaciones
max_iteraciones = 100
inercia = 0.5
cognitivo = 2
social = 2

# Inicialización de partículas
particulas = np.random.randint(0, 100, size=(num_particulas, num_dimensiones))
mejor_posicion_local = particulas.copy()
valor_objetivo_local = [float('inf')] * num_particulas
valor_objetivo_global = float('inf')
mejor_posicion_global = None

# Función de evaluación
def evaluar_asignacion(asignacion):
    # Calcular el total de bicicletas asignadas y el exceso
    total_bicicletas = sum(asignacion)
    exceso = max(0, total_bicicletas - limite_bicicletas)

    # Calcular la demanda insatisfecha
    demanda_total = sum(demanda_estaciones)
    demanda_insatisfecha = max(0, demanda_total - total_bicicletas)

    # Calcular el valor de la función objetivo (penalización por exceso y demanda i
    valor_objetivo = exceso * 100 + demanda_insatisfecha

    return valor_objetivo
```

```
# Algoritmo PSO
for iteracion in range(max_iteraciones):
    for i in range(num_particulas):
        # Asegurar que todas las estaciones tengan al menos 2 bicicletas
        particulas[i] = np.maximum(particulas[i], 2)

        valor_objetivo = evaluar_asignacion(particulas[i])
        if valor_objetivo < valor_objetivo_local[i]:
            valor_objetivo_local[i] = valor_objetivo
            mejor_posicion_local[i] = particulas[i].copy()

        if valor_objetivo < valor_objetivo_global:
            valor_objetivo_global = valor_objetivo
            mejor_posicion_global = particulas[i].copy()

    for i in range(num_particulas):
        velocidad = inercia * particulas[i] + \
                    cognitivo * random.random() * (mejor_posicion_local[i] - particula
                    social * random.random() * (mejor_posicion_global - particulas[i])
        particulas[i] = np.clip(particulas[i] + velocidad, 2, 100)  # Asegurar que to

# Mostrar la asignaci n  ptima
print("Asignaci n  ptima de bicicletas por estaci n:", mejor_posicion_global)
print("Valor  ptimo de la funci n objetivo:", valor_objetivo_global)

#-----------------------------------------------------------------------

# PSO con librer a Pyswarm

#-----------------------------------------------------------------------

import numpy as np
from pyswarm import pso

# Definir la funci n objetivo a minimizar
def objective(x):
    # x representa la asignaci n de bicicletas a estaciones
    total_bikes = np.sum(x)
    excess_bikes = max(0, total_bikes - 1500)  # Restricci n de l mite de bicicletas
    penalty = excess_bikes * 100  # Penalizaci n por exceso de bicicletas
    demand = np.array(demanda)  # Lista de demanda de cada estaci n (reemplaza los v
    total_demand = np.sum(demand)
    unmet_demand = max(0, total_demand - total_bikes)  # Penalizaci n por demanda no
    return penalty + unmet_demand

# Definir los l mites para la asignaci n de bicicletas (por ejemplo, de 0 a 100)
lb = [0] * 300
ub = [100] * 300

# Ejecutar PSO para minimizar la funci n objetivo
xopt, fopt = pso(objective, lb, ub)

# Mostrar la asignaci n  ptima de bicicletas por estaci n
print("Asignaci n  ptima de bicicletas por estaci n:", xopt)
print("Valor  ptimo de la funci n objetivo:", fopt)
```

5.3. Ant Colony Optimization (ACO)

```
# -*- coding: utf-8 -*-
#-------------------------------------------------------------------------------

# ACO

#-------------------------------------------------------------------------------

demanda = [5,5,5,5,5,5,5,5,12,5,5,5,5,5,5,5,5,5,6,5,5,5,5,5,6,5,5,5,5,5,7,11,5,
5,7,5,5,5,5,5,5,5,5,5,5,5,10,19,11,5,10,5,6,5,7,5,5,5,5,5,5,6,6,5,5,6,5,5,5,5,5,
8,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,6,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,8,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,7,5,5,5,5,5,5,5,5,5,5,11,5,6,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,
5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5]

import random
import numpy as np

# Datos de ejemplo: estaciones y demanda
num_estaciones = 300
demanda_estaciones = demanda[:]
bicicletas_disponibles = 1500

# Par metros de ACO
num_hormigas = 50
num_iteraciones = 100
alfa = 1.0  # Influencia de feromonas
beta = 1.0  # Influencia heur stica
ro = 0.1  # Tasa de evaporaci n de feromonas
Q = 100  # Cantidad de feromonas depositadas por cada hormiga
feromonas = np.ones((num_estaciones, num_estaciones))

# Funci n para calcular la probabilidad de transferencia de bicicletas
def calcular_probabilidad_transferencia(hormiga, estacion_actual, disponibles, demanc
    probabilidad = []
    for estacion in disponibles:
        if demanda_estaciones[estacion] > 0:
            prob = (feromonas[estacion_actual][estacion] ** alfa) * ((1.0 / demanda_e
            probabilidad.append(prob)
    total_probabilidad = sum(probabilidad)
    return [p / total_probabilidad for p in probabilidad]

# Funci n para evaluar la calidad de la asignaci n
def evaluar_asignacion(asignacion, demanda_estaciones):
    costo_total = 0
    exceso_total = 0

    for estacion, bicicletas_asignadas in enumerate(asignacion):
        demanda = demanda_estaciones[estacion]
        exceso = bicicletas_asignadas - demanda  # Calcula el exceso de bicicletas en

        # Penalizaci n por exceso de bicicletas
        if exceso >= 1:
```

```
            costo_total += exceso

        exceso_total += exceso

    return costo_total, exceso_total

# Inicializa el costo, el exceso y el n mero de bicicletas asignadas
mejor_costo = float('inf')
mejor_exceso = float('inf')
mejor_asignacion = np.zeros(num_estaciones)

# Algoritmo de optimizaci n de colonia de hormigas
for _ in range(num_iteraciones):
    for hormiga in range(num_hormigas):
        estacion_actual = random.randint(0, num_estaciones - 1)
        disponibles = list(range(num_estaciones))
        disponibles.remove(estacion_actual)
        asignacion_hormiga = [0] * num_estaciones

        while disponibles:
            probabilidad = calcular_probabilidad_transferencia(hormiga, estacion_actua

            # Verifica si hay probabilidades v lidas (mayores a cero)
            probabilidades_validas = [p for p in probabilidad if p > 0]

            if len(probabilidades_validas) == 0:
                # Si no hay probabilidades v lidas, selecciona una estaci n aleator
                siguiente_estacion = np.random.choice(disponibles)
            else:
                # Si hay probabilidades v lidas, selecciona con base en las probabili
                probabilidad_normalizada = [p / sum(probabilidades_validas) for p in
                siguiente_estacion = np.random.choice([e for e, p in zip(disponibles,

            # Aseg rate de no asignar m s bicicletas de las disponibles en la estac
            bicicletas_disponibles_estacion = min(bicicletas_disponibles - sum(asignac
            bicicletas_transferidas = min(bicicletas_disponibles_estacion, demanda_es

            # Comprobaci n: La suma de bicicletas asignadas por la hormiga no debe e
            if sum(asignacion_hormiga) + bicicletas_transferidas <= bicicletas_dispon
                asignacion_hormiga[siguiente_estacion] = bicicletas_transferidas

            # Actualizar feromonas basadas en la transferencia
            for i in range(num_estaciones):
                estacion1 = estacion_actual
                estacion2 = siguiente_estacion
                feromonas[estacion1][estacion2] = (1 - ro) * feromonas[estacion1][esta

            estacion_actual = siguiente_estacion
            disponibles.remove(siguiente_estacion)

        # Evaluar la calidad de la asignaci n de la hormiga
        asignacion_actual = np.argmax(feromonas, axis=1)
        costo, exceso = evaluar_asignacion(asignacion_actual, demanda_estaciones)

        # Comparar con la mejor asignaci n y costo anterior
        if costo < mejor_costo and exceso >= 0:
```

```
            mejor_asignacion = asignacion_actual
            mejor_costo = costo
            mejor_exceso = exceso

        # Si el costo y el exceso son cercanos a cero, det n el algoritmo
        if costo <= 0.001 and exceso >= 0 and exceso <= 0.001:
            break

# Resultado final
print(" Mejor  asignaci n encontrada!")
print("Mejor asignaci n de bicicletas:", mejor_asignacion.tolist())
print("Costo final:", mejor_costo)
print("Exceso total:", mejor_exceso)
```

6. Aplicación

Después de realizar los tres algoritmos, se observo claramente que el que mejor asignaba las bicicletas por estación fue el algoritmo genético. Por lo que se hizo una aplicación para automatizar el algoritmo y hacer que el usuario pueda acceder fácilmente a esta valiosa información. La interfaz de la aplicación tiene el siguiente diseño:

Figura 1: Interfaz de la aplicación

Además, la aplicación cuenta con un mapa interactivo donde en cada estación viene la demanda óptima según el algoritmo genértico. Este mapa se ve así:

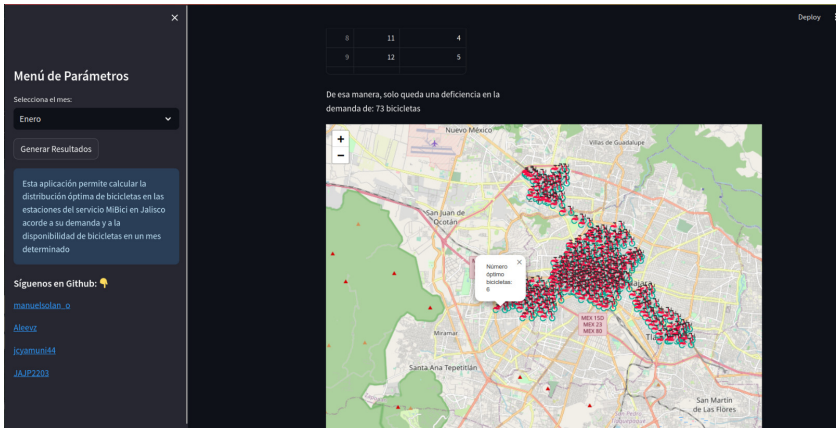

Figura 2: Mapa: demanda óptima por estación

7. Reflexiones

7.1. Alejandra

Mi experiencia en el trabajo en equipo en nuestro proyecto fue altamente positiva. Nuestro equipo demostró una excepcional capacidad para construir acuerdos, interactuar con respeto y generó resultados notables a través de un esfuerzo colaborativo. La disposición al diálogo y la resolución de conflictos de manera constructiva fueron claves. Cada miembro mostró un compromiso excepcional con la toma de decisiones y la generación de valor. En resumen, nuestro equipo demostró que, con organización y aprovechando las habilidades de todos, se pueden alcanzar logros destacados. Fue una experiencia gratificante y espero futuras oportunidades de colaboración.

7.2. José Antonio

Considero que nuestro grupo ha sobresalido en la construcción de acuerdos y en la interacción entre sus miembros. Hubo una notable colaboración, lo que se tradujo en resultados sobresalientes y un alto compromiso de todos. Lo que más valoré fue la dedicación de cada miembro al proceso de toma de decisiones y la generación de valor. Hubo un compromiso evidente en elevar la calidad de nuestro trabajo y en superar las expectativas. Estoy seguro de que, con la organización y la combinación de las habilidades de todos, pudimos alcanzar un nivel excepcional de desempeño.

7.3. José Carlos

Para este proyecto fue muy importante la paciencia a la hora de escoger lo que queríamos hacer como equipo ya que era nuestra responsabilidad escoger el tema. Una vez escogido el tema, fue necesaria una buena comunicación para que todos los miembros del equipo entendieran el planteamiento del problema y el objetivo. Además, para la elaboración del proyecto fue necesaria humildad para reconocer las limitantes que tenía el proyecto para así poder entregar algo sólido.

7.4. Juan Manuel

Mi experiencia en el trabajo en equipo durante el desarrollo de este proyecto ha sido enriquecedora en muchos aspectos. A través de la construcción de acuerdos, interacciones y relaciones sociales sólidas, la generación de resultados y compromisos, la utilización de estrategias de negociación y el compromiso con la toma de decisiones, he aprendido que el trabajo colaborativo va más allá de la obtención de resultados tangibles; es una oportunidad para crecer como individuo y como parte de un equipo. La capacidad de adaptarse, aprender de los desacuerdos y valorar la diversidad de opiniones ha sido la base de nuestro éxito y ha dejado una impresión duradera en mi enfoque hacia el trabajo en equipo en el futuro.

8. Conclusiones

Los algoritmos bioinspirados, inspirados en procesos naturales y fenómenos evolutivos, han demostrado ser poderosas herramientas para abordar desafíos logísticos y de optimización en contextos urbanos. El algoritmo genético, en particular, se ha destacado por su capacidad para adaptarse a entornos dinámicos y encontrar soluciones óptimas en escenarios complejos. En el contexto de la distribución de bicicletas para sistemas de bicicletas compartidas, esta flexibilidad es esencial debido a las variaciones en la demanda y las condiciones cambiantes de operación a lo largo del día.

La relevancia de los algoritmos bioinspirados no se limita únicamente a la optimización de la distribución de bicicletas, sino que se extiende a áreas como la reducción de costos operativos, la promoción de la movilidad sostenible y la mejora de la calidad de vida en las ciudades. La implementación exitosa del algoritmo genético en este estudio resalta su importancia como una solución valiosa para abordar la movilidad urbana del siglo XXI.

9. Referencias

MiBici. (2023). Recuperado de https://www.mibici.net/