sumo\_simulation.py

class SumoSimulation

Clase usada para controlar la ejecución de la simulaciones

@staticmethod

def net\_from\_nod\_edg(nodes, edges, dest)

Método usado para generar un archivo de red net.xml en base a un nod.xml y un edg.xml usando el comando netconvert del paquete de herramientas SUMO, devuelve la ruta de dicha red

Args:

nodes (str), ruta del archivo nod.xml

edges (str), ruta del archivo edg.xml

dest, ruta destino del archivo net.xml resultante, en caso de no especificar archivo se guarda como network.net.xml

Returns:

str, ruta de la red

@staticmethod

def net\_from\_osm(open\_street, dest)

Método usado para generar un archivo de red net.xml en base a un osm usando el comando netconvert del paquete de herramientas SUMO, devuelve la ruta de dicha red

Args:

open\_street (str), ruta del archivo osm

dest (str), ruta destino del archivo net.xml resultante, en caso de no especificar archivo se guarda como network.net.xml

Returns:

str, ruta de la red

@staticmethod

def random\_trips(network, random, dest)

Método usado para generar un archivo de viajes trips.xml en base a un la red net.xml usando el archivo randomTrips.py del paquete de herramientas SUMO, devuelve la ruta de dichos viajes

Args:

network (str), ruta del archivo de red net.xml

random (str), ruta del script de python randomTrips.py

dest (str), ruta destino del archivo trip.xml resultante, en caso de no especificar archivo se guarda como traffic.trips.xml

Returns:

str, ruta de los viajes

@staticmethod

def rou\_from\_trip(network, trips, dest)

Método usado para generar un archivo de viajes rou.xml en base a viajes en trip.xml usando el comando duarouter del paquete de herramientas SUMO, devuelve la ruta de dichas rutas

Args:

network (str), ruta del archivo de red net.xml

trips (str), ruta del archivo de viajes trip.xml

dest (str), ruta destino del archivo rou.xml resultante, en caso de no especificar archivo se guarda como routes.rou.xml

Returns:

str, ruta de las rutas

@staticmethod

def config\_from\_net\_rou(network, routes, dest)

Método usado para generar un archivo configuración de la simulación sumocfg en base a la red en net.xml y los viajes rutas en rou.xml, devuelve la ruta de dichas rutas

Args:

network (str), ruta del archivo de red net.xml

routes (str), ruta del archivo de rutas rou.xml

dest (str), ruta destino del archivo sumocfg resultante, en caso de no especificar archivo se guarda como simulation.sumocfg

Returns:

str, ruta de la configuración

def \_\_init\_\_(self, configuration)

Constructor de la clase SumoSimulation

Args:

sumo\_configuración (str), ruta del archivo de configuración sumocfg

def set\_files(self, config)

Método usado para asignar la ruta de un archivo de configuración sumocfg al objeto

Args:

sumo\_configuración (str), ruta del archivo de configuración sumocfg

def start\_simulation(self, visual)

Método usado para preparar la simulación de SUMO para su inicio en base al archivo de configuración

Args:

visual (bool), bandera que específica si la simulación a ejecutar se hará de forma visual mediante sumo-gui o directamente en sumo sin elementos visuales

def run\_simulation(self, steps)

Método usado para ejecutar la simulación especificada una cantidad de n pasos en SUMO, devuelve un diccionario con los datos resultantes de la misma

Args:

steps (int), cantidad de ciclos dentro de la simulación a ejecutar

Returns:

dict[str, float], diccionario con resultados de la simulación, contiene:

arrived\_number: número de vehículos que salieron de la red

departed\_number: número de vehículos que entraron a la red

average\_speed: velocidad promedio de los vehículos

average\_wait\_time: tiempo de espera promedio de los vehículos

average\_travel\_time: tiempo de viaje promedio de los vehículos

def end\_simulation(self)

Método que finaliza y cierra de manera correcta la simulación

def get\_lights(self)

Método que da a conocer las configuraciones de los semáforos en la simulación, devuelve

Returns:

dict[str, list], diccionario con la id y fases de cada uno de los semáforos

int, número total de fases de todos los semáforos

traffic\_demand.py

class TrafficDemand

Clase encargada de la lectura de datos relacionados al tráfico a partir de archivos cvs y la generación de tráfico a partir de matrices de demanda.

Trabaja principalmente en base a diccionarios con la llave representada por el id de una calle existente en la red y como valor la cantidad de vehículos que pasan por dicha calle

@staticmethod

def read\_csv(route)

Método usado para la lectura del tráfico vehicular dado desde un csv con una columna representado el id de las calles y otra la cantidad de vehículos, retorna un diccionario con los valores del mismo

Args:

route (str), ruta del archivo csv a extraer

Returns:

dict[str, int], diccionario con id de las calles y la cantidad de vehículos

@staticmethod

def match\_traffic(traffic\_1, traffic\_2)

Método usado para igualar el total de los valores de dos diccionarios de tráfico de manera proporcional, modifica los valores de los diccionarios in place

Args:

traffic\_1 (dict[str, int]), diccionario con id de las calles y cantidad de vehículos

traffic\_2 (dict[str, int]), diccionario con id de las calles y cantidad de vehículos

@staticmethod

def traffic\_demand(incoming\_traffic, outgoing\_traffic)

Método que distribuye el tráfico de manera proporcional en base a un diccionario con los datos de entrada y otros con los datos de salida, devuelve una lista de trios de datos que poseen el id origen, el id destino y la cantidad de vehículos asignados a dicho viaje

Args:

incoming\_traffic (dict[str, int]), diccionario con id de las calles y cantidad de vehículos que entran al sistema

outgoing\_traffic (dict[str, int]), diccionario con id de las calles y cantidad de vehículos que entran al sistema

Returns:

list[tuple(str, str, int)], Lista de trios que contienen el id de origen, el id de destino y la cantidad de vehículos con dicho viaje

@staticmethod

def random\_traffic\_demand(incoming\_traffic, outgoing\_traffic)

Método que distribuye el tráfico de manera aleatorial en base a un diccionario con los datos de entrada y otros con los datos de salida, devuelve una lista de trios de datos que poseen el id origen, el id destino y la cantidad de vehículos asignados a dicho viaje

Args:

incoming\_traffic (dict[str, int]), diccionario con id de las calles y cantidad de vehículos que entran al sistema

outgoing\_traffic (dict[str, int]), diccionario con id de las calles y cantidad de vehículos que entran al sistema

Returns:

list[tuple(str, str, int)], Lista de trios que contienen el id de origen, el id de destino y la cantidad de vehículos con dicho viaje

@staticmethod

def write\_taz\_od(incoming\_traffic, outgoing\_traffic, traffic, time, dest)

data\_writer.py

class DataWriter

Clase utilizada para registrar el resultado de las simulaciones a través de la clase LightsFunctions en archivos json, guarda los datos de todas las simulaciones en una lista de diccionarios y registra por separado el mejor de los casos

def \_\_init\_\_(self, file\_name, directory)

Constructor de las clase DataWriter

Args:

file\_name (str), nombre del archivo json a crear o modificar

directory: (str), ruta del archivo json a crear

def read\_file(self):

Método que lee el archivo json declarado en el objeto y devuelve un diccionario con dichos datos

Returns:

dict[str, dict], Diccionario que contiene los datos principales de cada una de las simulaciones, así como acceso directo al mejor de estos

def write\_file(self):

Método que escribe y añade los datos almacenados dentro del propio objeto en el archivo ya especificado con anterioridad

def overwrite\_best(self)

Método que sobrescribe los datos almacenados dentro del propio objeto en el archivo ya especificado con anterioridad

def add\_data(self, data)

Método que almacena dentro del objeto la información de cada una de las simulaciones otorgada por las mismas

Args:

data (dict), diccionario que posee la información resultante de cada simulación, siendo estas en su mayoría la ya mencionada en el método run\_simulation de SumoSimulation

def change\_file(self, file\_name, directory, write)

Método que cambia el actual archivo json con el que se está trabajando dentro del objeto

Args:

file\_name (str), nombre del nuevo archivo json a modificar

directory (str), directorio del nuevo archivo json a modificar, si no se pasa nada el directorio se mantiene igual

write (bool), booleano que da a conocer si los datos registrados hasta el momento serán añadidos al anterior archivo o descartados

def set\_best(self)

Método que actualiza el valor de la mejor simulación dentro del objeto

lights\_functions.py

class LightsFunctions:

Clase encargada de funcionar como mediador entre la clase SumoSimulation y los algoritmos de optimización, convirtiendo la entrada x en algo que pueda ser usado dentro de la simulación para después usar los resultados de la misma en algo legible para los algoritmos de optimización.

def \_\_init\_\_(self, file, cars, steps, data\_writer)

Crea una instancia de la clase LightsFunctions

Args:

file (str), dirección del archivo de configuración a usar en la simulación

cars (int), total de vehículos en la simulación

steps (int), Lista de valores máximos para cada una de las dimensiones de x

data\_writer (int), número de iteraciones de búsqueda del mejor resultado

def get\_metrics\_function(data)

Transforma las variables obtenidas de la simulación como velocidad promedio en una única salida para los algoritmos de optimización

Args:

data (dict), datos de la simulación obtenida de SumoSimulation

Returns:

float, resultado de las operaciones

def get\_ligths\_phases(self, simulation)

Obtiene la lista de todas las fases de cada uno de los semáforos de la simulación actual y lo guarda dentro del propio objeto

Args:

simulation (SumoSimulation), simulación de donde se obtienen las fases

def get\_min\_max(self, min\_time\_rg, max\_time\_rg, min\_time\_y, max\_time\_y)

Función que en base al tiempo asignado en los argumentos, devuelve dos listas de tamaño n con dichos valores, siendo n el total de etapas de cada uno de los semáforos.

Args:

min\_time\_rg(float), tiempo mínimo para las etapas únicamente con semáforos rojos y verdes

max\_time\_rg(float), tiempo máximo para las etapas únicamente con semáforos rojos y verdes

min\_time\_y(float), tiempo mínimo para las etapas con semáforos amarillos, en caso de no específicarse toma el tiempo de min\_time\_rg

max\_time\_y(float), tiempo máxmio para las etapas con semáforos amarillos, en caso de no específicarse toma el tiempo de max\_time\_rg

Returns:

list[float], lista de el tiempo mínimo de cada una de las etapas

list[float], lista del tiempo máximo de cada una de las etapas

def all\_lights(self, x, visual)

Ejecuta la simulación modificando el tiempo de cada una de las etapas de los semáforos en base a la lista de número x, devuelve el resultado de get\_metrics\_function como valor a minimizar

Args:

x (list[float]), lista de tiempos para cada una de las fases de los semáforos

visual (bool), booleano que dicta si la simulación será ejecutada con interfaz gráfica

Returns:

float, resultado de las operaciones de get\_metrics\_function a minimizar

optimization.py

def random\_simulation(func, xLow, xHigh, n, data)

Método que permite la minimización de funciones de n dimensiones mediante el algoritmo random en un rango limitado, devuelve el valor de X y Y con los mejores resultados

Args:

func, referencia a la función a optimizar

xLow (list[float]), Lista de valores mínimos para cada una de las dimensiones de x

xHigh (list[float]), Lista de valores máximos para cada una de las dimensiones de x

n (int), número de iteraciones de búsqueda del mejor resultado

data, (dict[list[float], float]), datos del mejor resultado de iteraciones pasadas.

Returns:

list[float], valor x del mejor resultado de la simulación

float, valor y del mejor resultado de la simulación

def hill\_simulation(func, xLow, xHigh, n, data)

Método que permite la minimización de funciones de n dimensiones mediante el algoritmo hill climbing en un rango limitado, devuelve el valor de X y Y con los mejores resultados

Args:

func, referencia a la función a optimizar

xLow (list[float]), Lista de valores mínimos para cada una de las dimensiones de x

xHigh (list[float]), Lista de valores máximos para cada una de las dimensiones de x

n (int), Número de iteraciones de búsqueda del mejor resultado

data, (dict[list[float], float]), datos del mejor resultado de iteraciones pasadas.

Returns:

list[float], valor x del mejor resultado de la simulación

float, valor y del mejor resultado de la simulación

def swarm\_simulation(func, xLow, xHigh, m, n, data)

Método que permite la minimización de funciones de n dimensiones mediante el algoritmo swarm en un rango limitado, devuelve el valor de X y Y con los mejores resultados

Args:

func, referencia a la función a optimizar

xLow (list[float]), Lista de valores mínimos para cada una de las dimensiones de x

xHigh (list[float]), Lista de valores máximos para cada una de las dimensiones de x

m (int), Número de partículas deseadas dentro de la simulación

n (int), Número de iteraciones de búsqueda del mejor resultado

data, (dict[list[float], float]), datos del mejor resultado de iteraciones pasadas.

Returns:

Paticle, partícula con los datos del mejor resultado de la simulación

def genetic\_simulation(func, xl, xh, N, M, fxl, fxh)

Método que permite la minimización de funciones de n dimensiones mediante el algoritmo swarm en un rango limitado, devuelve el valor de X y Y con los mejores resultados

Args:

func, referencia a la función a optimizar

xl (list[float]), Lista de valores mínimos para cada una de las dimensiones de x

xh (list[float]), Lista de valores máximos para cada una de las dimensiones de x

N (int), Número de iteraciones de búsqueda del mejor resultado

M (int), Número de individuos deseadas dentro de la simulación

fxl, Límite inferior estimado del valor de Y

fxh, Límite superior estimado del valor de Y

Returns:

Individual, individuo con los datos del mejor resultado de la simulación