A PYTHON TOOL FOR SUMO TRAFFIC SIMULATION TO MODEL START-STOP SYSTEM FOR COMPREHENSIVE EMISSION ANALYSIS

Authors:
Tamás Tettamanti, Okolie Chinedu, Tamás Ormándi and Balázs Varga

Technical report for the open-source code of https://github.com/bmetrafficlab/SUMO_start_stop_system

BME Traffic Lab
Department of Control for Transportation and Vehicle Systems
Faculty of Transportation Engineering and Vehicle Engineering
Budapest University of Technology and Economics

MŰEGYETEM 1782

Budapest, 2024

Contents

1 Purpose of the Program

The Simulation of Urban Mobility (SUMO) [Lopez et al., 2018] is an advanced, open-source framework designed to facilitate detailed analysis and simulation of vehicular traffic in urban environments [Kovács et al., 2021, Gressai et al., 2021]. Developed by the German Aerospace Center, SUMO offers a versatile platform for microscopic modeling intermodal traffic systems, including but not limited to road vehicles, public transport, and pedestrians. SUMO's adaptability and extensive toolset, including the Traffic Control Interface (TraCI) [Wegener et al., 2008], enables users to dynamically interact with simulations, thereby providing an invaluable resource for traffic engineers, urban planners, and researchers aiming to optimize traffic flows and assess the environmental impacts of various traffic scenarios.

Leveraging the modular architecture of SUMO and the flexibility of TraCI, our program introduces an innovative approach to analyzing vehicular emissions under different traffic conditions. By simulating start-stop vehicle dynamics within urban traffic networks, the tool aims to quantify the emission reductions achievable through intelligent traffic management and vehicle operation strategies. This enhancement of SUMO's capabilities allows for granular analysis of emissions across a wide spectrum of pollutants, including CO_2, CO, HC, NO_x, and PM_x, offering deeper insights into the environmental benefits of deploying start-stop technologies in urban fleets.

2 Preliminaries

This section introduces Netedit, TraCI, and the HBEFA4 Emission Model [Keller et al., 2017] as key components of SUMO's traffic simulation toolkit. Netedit facilitates the creation of traffic networks, while TraCI enables real-time simulation interactions. The HBEFA4 Emission Model adds depth by providing detailed emission factors, essential for environmental impact assessments. Together, these tools equip users to design, analyze, and optimize urban traffic scenarios efficiently, supporting sustainable traffic management and planning efforts.

2.1 Designing and Configuring Traffic Networks in Sumo with Netedit

Netedit, a key component of the SUMO suite, offers an intuitive graphical interface for constructing and modifying traffic networks, from simple road layouts to complex urban infrastructures. It accommodates a wide range of inputs and outputs, facilitating the design, tweak, and implementation of various traffic scenarios, including the detailed management of vehicles, routes, and additional simulation elements. Its comprehensive toolset, aligned closely with sumo-gui's interface, ensures an accessible yet powerful platform for users at all levels, streamlining the process of network creation and adjustment through functionalities like undo/redo, hotkeys, and the management of additional data and shapes.

The process of network creation with Netedit is designed to be straightforward, enabling the swift assembly and customization of junctions, edges, and more, culminating in the generation of .net.xml files for simulation. This ease of use extends to the fine-tuning of network attributes and the application of advanced configurations, such as roundabout conversion and junction modifications, showcasing Netedit's versatility in network design. For those new to SUMO, Netedit demystifies network file creation, offering step-by-step guidance that leads to the successful development of functional traffic simulations.

Beyond network design, Netedit plays a crucial role in the broader simulation preparation process, aiding in the creation of SUMO configuration files (.sumo.cfg) which encapsulate the simulation parameters. This holistic approach to traffic simulation setup, combined with SUMO's extensive toolset, enables a deep exploration of traffic dynamics and provides a solid foundation for analyzing the impacts of various traffic management strategies and infrastructural changes within simulated environments, thus offering a rich resource for traffic studies and urban planning initiatives.

2.2 Traffic Control Interface (TraCI) of SUMO

TraCI (Traffic Control Interface) of SUMO simulator is utilized to dynamically interact with simulations in SUMO from external applications, notably in Python, enabling real-time control and monitoring of simulation variables. This interface facilitates extensive manipulation of the simulation, including vehicle rerouting, traffic light control, and scenario adjustments based on evolving conditions. By scripting in Python, users can leverage TraCI to implement complex traffic management strategies and conduct detailed emissions analysis, integrating with the simulation to alter vehicle behavior, assess environmental impacts, and optimize traffic flows. The use of TraCI with Python enhances the flexibility and applicability of SUMO for custom scenario development and research purposes.

This lays the foundation for understanding how SUMO and TraCI, in conjunction with Python, form a powerful toolkit for the simulation and analysis of urban mobility and its environmental impacts. The integration of these tools supports a wide range of traffic studies, from basic network modeling to advanced traffic management and emissions studies.

2.3 HBEFA4 Emission Model

The Handbook Emission Factors for Road Transport (HBEFA) is a pivotal resource that provides emission factors for a wide range of vehicle categories across various traffic situations. Developed initially for the Environmental Protection Agencies of Germany, Switzerland, and Austria, HBEFA has expanded its influence with contributions from several other European countries. The model offers detailed emission factors not only for regulated pollutants but also for key non-regulated pollutants, fuel consumption, and CO_2 emissions. HBEFA is instrumental for accurate emission calculations at both macro and micro levels, from national inventories to specific road segments, making it an indispensable tool for environmental analysis and traffic planning.

HBEFA's comprehensive approach includes hot emissions, cold start excess emissions, and evaporative emissions, covering a broad spectrum of pollutants for passenger cars, light and heavy-duty vehicles, buses, and motorcycles. This model is crucial for evaluating the environmental impact of road transport, supporting a wide array of applications from national inventory reports to air quality modeling and impact assessments. Its data, derived from multinational measurement campaigns and statistical analyses, ensure the reliability and relevance of the emission factors provided.

Integrating HBEFA4/PC_petrol_Euro-4 or similar classes into SUMO simulations is straightforward, enhancing the accuracy of emission analysis. By including specifications such as

<vType emissionClass="HBEFA4/PC_petrol_Euro-4"/>

in the additional files, one can simulate specific vehicle types with precise emission profiles. This integration elevates the quality of environmental impact assessments, exploiting HBEFA as an essential component for traffic studies aiming to mitigate air pollution.

3 Overview of the Program

In this part, the main features of the developed program are introduced.

3.1 Starting the Program

The program, written in Python, begins with an import statement that brings in the essential *'os'* module and a custom module named *'startstop'*. The *'os'* module is a standard utility in Python that allows interaction with the operating system, such as checking for the existence of files, which is pertinent for validating the SUMO configuration path. The *'startstop'* module contains the specific functions required to run and manage the SUMO simulation.

```python
import os
import startstop as Stp
```

3.2 Customization Overview

The user-driven customization process is initiated in the *'main.py'* file, where the user can define the path to the SUMO configuration file *(sumocfg)*. This path is essential for locating the simulation setup files, including vehicle routes and network definitions.

```python
sumocfg = "C:/One_lane_signalized_full_green/cfg_70_capacity.sumocfg"
```

The user determines the simulation's operation mode by setting the *'ratio_based_simulation'* flag. If set to True, the simulation assigns vehicles to the start-stop category based on the percentage defined in *'start_stop_ratio'*. To set this ratio, the user specifies a value between 0 and 100 in the *'start_stop_ratio'* variable, which represents the percentage of vehicles in the simulation that will have start-stop functionality. Conversely, if set to False, the simulation identifies start-stop vehicles by their predefined vehicle type, ”start-stop-vehicle”, within the SUMO configuration files. This flexibility allows the user to choose between a dynamic, ratio-based assignment of start-stop functionality and a static, vehicle-type-based assignment, tailoring the simulation to their specific research needs. Users also now have the ability to define the duration of idle emissions through the idle_time_in_sec setting, enhancing the tool's adaptability to various start-stop behavior research. This setting, defaulting to 7 seconds, is informed by findings from research indicating this duration's relevance in accurately simulating the emissions impact of start-stop technology [Xie et al., 2016]. This customization enhances the simulation's precision, allowing for more tailored analysis reflective of real-world conditions and research insights.

```
ratio_based_simulation = True
start_stop_ratio = 80
idle_time_in_sec = 7
```

Incorporating vehicle-type-based customization, users can switch to a specific vehicle behavior model by setting *ratio_based_simulation = False* in the main.py file. This approach necessitates defining vehicle types in SUMO's additional configuration file, such as *input_additional.add.xml*, where vehicle types like ”start-stop-vehicle” are explicitly declared with attributes including car follow model and emission class.

```
<additional>
<edgeData id="1" type="emissions" freq="1800" file="emissions.xml" excludeEmpty="true
    " end="1800" withInternal="true"/>
<vType id="intelligent_driver" carFollowModel="EIDM" laneChangeModel="LC2013" minGap=
    "1.5" emissionClass="HBEFA4/PC_petrol_Euro-4"/>
<vType id="start-stop-vehicle" carFollowModel="EIDM" laneChangeModel="LC2013" minGap=
    "1.5" emissionClass="HBEFA4/PC_petrol_Euro-4"/>
</additional>
```

Defining vehicle flows in the route file *route_free.rou.xml* allows for the simulation of traffic with these specific vehicle types, enabling a detailed analysis of their behavior and emissions under simulated traffic conditions. This customization layer adds depth to the simulation, allowing for nuanced studies of different vehicle technologies and their impact on traffic dynamics and emissions.

```
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/routes_file.xsd">
<flow id="0" from="1" to="-3" type="start-stop-vehicle" begin="0" end="1800"
    probability="0.03"/>
<flow id="4" from="1" to="-2" type="intelligent_driver" begin="0" end="1800"
    probability="0.03"/>
#.....
#.....
<flow id="14" from="4" to="-3" type="intelligent_driver" begin="0" end="1800"
    probability="0.03"/>
</routes>
```

In addition to these settings, users can customize idle emission values through the *idle_values* dictionary. By default, these values are based on the HBEFA4/PC_petrol_Euro-4 model, but they can be overwritten with user-specified parameters, allowing for precise control over the emission factors used in the simulation.

```
idle_values = None
# Example parameter selection:
#idle_values = {'CO2': 1.8, 'CO': 3.0126e-12}
```

This customization overview emphasizes the configurability of the simulation tool, ensuring that the user can closely tailor the simulation parameters to suit specific research questions or scenarios before executing the main.py script.

3.3 Parameter Validation

Upon execution, the *main()* function within *'main.py'* is responsible for ensuring that the simulation settings adhere to logical constraints. It first examines the *'start_stop_ratio'*, a critical value that determines the proportion of vehicles with start-stop functionality. The function checks whether this ratio is within the acceptable bounds, greater than 0% but less than or equal to 100%. If the *'start_stop_ratio'* falls outside this range, the program halts and notifies the user with a printed message, prompting for a valid ratio within the specified limits.

```
if start_stop_ratio > 100 or start_stop_ratio == 0:
    print("The start_stop_ratio must be larger than 0% and lower or equal to 100%!")
    return
```

Next, the program turns its attention to the *sumocfg* variable, which should contain the path to the SUMO configuration file. This file is the blueprint for the simulation, containing all the necessary settings and parameters. The existence of this file at the specified location is crucial. The *os* module's *'exists'* function is invoked to verify this. If the file is missing, or if *sumocfg* is not set, the program again stops and alerts the user, requesting a valid SUMO configuration file path. Only once these parameters are validated does the program proceed to the subsequent stages of the simulation process.

```
if sumocfg:
    if os.path.exists(sumocfg):
        print("Extracting SUMO configuration settings...")

        # Extract the step length size from the given SUMO configuration
        # .....
        # .....

    else:
        print("The given SUMO configuration file does not exist!")
        return
else:
    print("Please provide a SUMO configuration file for the simulation.")
    return
```

After parameter validation, the program proceeds to extract crucial simulation settings from the SUMO configuration file. It uses the *startstop* module to call two specific functions: *extract_step_length* and *extract_duration*. These functions read the SUMO configuration file specified by the *sumocfg* variable to determine the simulation's step length and duration. The step length defines the time interval between each simulation step, while the duration specifies the total length of the simulation. These extracted values are essential for accurately timing the simulation and ensuring that the start-stop functionality is assessed over the correct period. With these settings retrieved and converted to floating-point numbers for precision, the program signals readiness to initiate the SUMO simulation, transitioning to the execution phase.

```
stepsize = float(Stp.extract_step_length(sumocfg))
duration = float(Stp.extract_duration(sumocfg))
```

3.4 Emission Data Handling

Once the SUMO simulation is initiated, the program focuses on collecting and processing vehicle emission data. It distinguishes between *start-stop vehicles* and *regular vehicles*, applying user-defined or default idle emission values specifically to *start-stop vehicles*. This differentiation is crucial for assessing the impact of the start-stop functionality on emissions accurately. The simulation loop iterates through each simulation step, capturing emission data for all vehicles.

Upon the *run_simulation* function's invocation, it first ensures a designated directory for storing results is present, creating one if absent, which is foundational for organizing the simulation outputs.

```
if not os.path.exists("results"):
    os.makedirs("results")
    print("Results folder was created.")
```

The function sets up default idle emission values, offering the option to use user-specified values CO_2, CO, HC, NO_x, PM_x), thereby tailoring the simulation to various environmental conditions. It then kickstarts the SUMO simulation, employing user-defined configurations and emission output settings for precise control.

Throughout the simulation, vehicles are dynamically classified as either start-stop or regular, based on predefined criteria such as the *start_stop_ratio* or their specific types. This classification leverages sets for vehicle IDs to guarantee uniqueness and facilitate swift access, which is vital for managing large volumes of vehicle data efficiently.

In the simulation, *start_stop_data* and *regular_data* dictionaries store emission details for start-stop and regular vehicles, respectively, allowing for in-depth analysis of their emission impacts. The *start_stop_vehicles* and *assigned_vehicles* sets track vehicles with start-stop functionality and all vehicles processed, ensuring efficient management and no duplication. *total_vehicles_processed* counts every vehicle encountered, aiding in the dynamic assignment of start-stop functionality based on the ratio. *vehicle_emission_data_per_step* dictionary captures emissions at each simulation step, providing a detailed temporal view of emissions for further analysis:

```
# Data storages
start_stop_data = {}
regular_data = {}
start_stop_vehicles = set()
assigned_vehicles = set()
total_vehicles_processed = 0
vehicle_emission_data_per_step = {}
```

As the simulation progresses, it meticulously collects emission data at each step. Notably, for vehicles with start-stop functionality, the program adjusts their emissions to include idle emissions for a 7-second duration if they've been stationary for more than 2 seconds. This adjustment is crucial for accurately simulating the emission reductions achieved through start-stop technology.

```
if start_stop_data[vehicle_id]['time'] >= 2:
    adjusted_co2 = current_co2 + idle_co2_7s
```

As highlighted earlier, emission data is stored in structured dictionaries, segregating start-stop and regular vehicles, which simplifies the comparative analysis of their emissions. The operational logic for implementing start-stop functionality, underscored by the *start_stop_threshold*, reflects the simulation's ability to mimic real-world scenarios closely. The *start_stop_threshold* is calculated as the *start_stop_ratio* divided by 100, converting the percentage into a decimal, i.e., $StartStopThreshold = \frac{StartStopRatio}{100}$.

```
start_stop_threshold = start_stop_ratio / 100.0
```

This threshold is used to dynamically determine whether a vehicle should be assigned start-stop functionality during the simulation, based on the specified ratio. If the ratio of currently identified start-stop vehicles to the total number of processed vehicles falls below this threshold, new vehicles will be assigned start-stop capabilities until the ratio is met. This mechanism ensures that the proportion of start-stop vehicles in the simulation accurately reflects the user's specified ratio.

```
if ((len(start_stop_vehicles) / total_vehicles_processed) < start_stop_threshold and
    ratio_based_simulation):
    start_stop_vehicles.add(vehicle_id)
```

The simulation's timeline is carefully managed, ensuring that data collection and adjustments are precisely timed and recorded, highlighting the program's sophisticated approach to simulating and analyzing the impact of start-stop functionality on vehicle emissions.

3.5 Post-Simulation Processing

After the simulation concludes, the program processes the collected emission data. This involves the *create_start_stop_emissions* function, which adapts the original SUMO emissions data to reflect adjustments for vehicles with start-stop functionality. The *create_start_stop_emissions* function plays a pivotal role in adjusting the emissions data post-simulation. It starts by ensuring both the start-stop vehicle data and stop times are available:

```
if os.path.exists(original_emissions_file):
```

Then, it copies the original SUMO emissions file to a new file, maintaining data integrity. The *copy_file* function is a crucial part of the post-simulation processing that ensures the integrity of the original emissions data. It copies the original emissions file to a new file before any modifications are made. This step is essential for maintaining a backup of the original data, allowing the program to safely update emissions for start-stop vehicles without risking loss or corruption of the original data set. Successful execution of this function is confirmed to the user, reinforcing the program's reliability and attention to data preservation.

```
def copy_file(original_file, new_file):
    try:
        with open(original_file, 'rb') as f_original:
            with open(new_file, 'wb') as f_new:
                # Read from the original file and write to the new file
                f_new.write(f_original.read())
        print(f"File '{original_file}' copied to '{new_file}' successfully.")
        return True
    except FileNotFoundError:
        print("One of the files doesn't exist.")
        return False
```

This new file updates the CO_2, CO, HC, NO_x, and PM_x values for start-stop vehicles, based on their stop times and idle emissions. This process involves parsing the XML structure to locate and adjust specific vehicle emissions:

```
if os.path.exists(original_emissions_file):
    if copy_file(original_emissions_file, start_stop_emissions_file):
        tree = ET.parse(start_stop_emissions_file)
        root = tree.getroot()
```

Successful completion is confirmed through a message, whereas failure due to missing files or data results in an error message, highlighting the importance of complete and accurate data for this operation. This post-processing step is crucial for accurately representing the impact of start-stop technology on vehicle emissions within the simulation's results.

3.6 Cumulative Emissions and Output

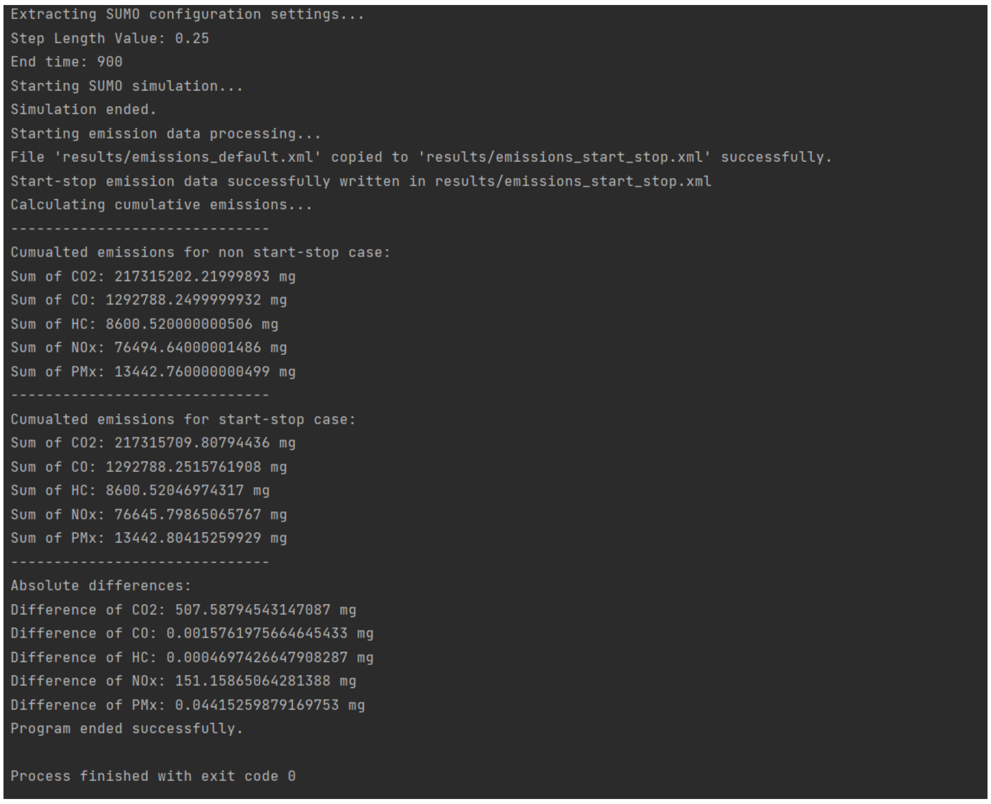
The *calculate_cumulative_emissions* function processes emission data after the simulation, comparing emissions between standard and start-stop vehicles. It reads XML files for both vehicle types, tallying CO_2, CO, HC, NO_x, and PM_x emissions. The function calculates total emissions and their absolute differences, highlighting the start-stop technology's impact.

```
print("------------------------------")
print("Cumulated emissions for start-stop case:")
print("Sum of CO2:", sum_CO2_start_stop, "mg")
print("Sum of CO:", sum_CO_start_stop, "mg")
print("Sum of HC:", sum_HC_start_stop, "mg")
print("Sum of NOx:", sum_NOx_start_stop, "mg")
print("Sum of PMx:", sum_PMx_start_stop, "mg")
print("------------------------------")
print("Absolute differences:")
print("Difference of CO2:", abs(sum_CO2_start_stop - sum_CO2), "mg")
print("Difference of CO:", abs(sum_CO_start_stop - sum_CO), "mg")
print("Difference of HC:", abs(sum_HC_start_stop - sum_HC), "mg")
print("Difference of NOx:", abs(sum_NOx_start_stop - sum_NOx), "mg")
print("Difference of PMx:", abs(sum_PMx_start_stop - sum_PMx), "mg")
```

Finally, it visualizes these differences with a bar graph using *matplotlib*, offering a clear representation of the emissions reduction achieved through start-stop functionality. This step is crucial for analyzing and demonstrating the environmental benefits of incorporating start-stop systems in vehicles.

3.7 Program Ending

Upon completion, the program assesses the successful execution of tasks, ensuring no errors have occurred. This phase marks the end of the simulation process, highlighting the effectiveness and reliability of the program in analyzing the impact of start-stop technology on vehicle emissions. The conclusion serves as a checkpoint, confirming the integrity and completion of the simulation and its objectives.



```
Extracting SUMO configuration settings...
Step Length Value: 0.25
End time: 900
Starting SUMO simulation...
Simulation ended.
Starting emission data processing...
File 'results/emissions_default.xml' copied to 'results/emissions_start_stop.xml' successfully.
Start-stop emission data successfully written in results/emissions_start_stop.xml
Calculating cumulative emissions...
------------------------------
Cumualted emissions for non start-stop case:
Sum of CO2: 217315202.21999893 mg
Sum of CO: 1292788.2499999932 mg
Sum of HC: 8600.520000000506 mg
Sum of NOx: 76494.64000001486 mg
Sum of PMx: 13442.760000000499 mg
------------------------------
Cumualted emissions for start-stop case:
Sum of CO2: 217315709.80794436 mg
Sum of CO: 1292788.2515761908 mg
Sum of HC: 8600.52046974317 mg
Sum of NOx: 76645.79865065767 mg
Sum of PMx: 13442.80415259929 mg
------------------------------
Absolute differences:
Difference of CO2: 507.58794543147087 mg
Difference of CO: 0.0015761975664645433 mg
Difference of HC: 0.0004697426647908287 mg
Difference of NOx: 151.15865064281388 mg
Difference of PMx: 0.04415259879169753 mg
Program ended successfully.

Process finished with exit code 0
```

Figure 1: Ratio based output example

4 A Working Example

In this part, a working example is presented with example outputs generated by the tool.

4.1 Assessing the Emission Effect of a Ratio of Vehicles with Start-Stop System

In setting up a SUMO simulation environment to explore the impact of start-stop technology, we're focusing on a scenario where 80% of the vehicles are equipped with start-stop systems, as indicated by the setting *start_stop_ratio = 80* in the *main.py* file. This configuration, applied within a free traffic network scenario such as *"C:/One_lane_signalized_full_green/cfg_70_free.sumocfg,"* allows for a detailed examination of how widespread adoption of start-stop technology can influence emissions without necessitating changes to additional or route files. By adjusting the simulation to reflect an 80% adoption rate or whatever ratio deemed fit, users can gain insights into the potential environmental benefits of implementing start-stop systems in urban mobility planning.

When running a SUMO simulation with an 80% start-stop ratio in a free traffic scenario, users can expect to see a detailed output of cumulative emissions comparing vehicles with and without start-stop functionality. As seen in the provided results, the simulation calculates and presents the sum of various pollutants, such as CO_2, CO, HC, NO_x, and PM_x.

The output includes absolute differences in emissions between the two cases, signifying the changes brought about by the start-stop technology as shown in Fig. 1. For instance, a notable difference in CO_2 and NO_x emissions indicates the effectiveness of start-stop systems in reducing these specific pollutants.

The accompanying plot visualizes these differences, enabling users to quickly discern the environmental impact of implementing start-stop technology at the specified ratio (Fig. 2).

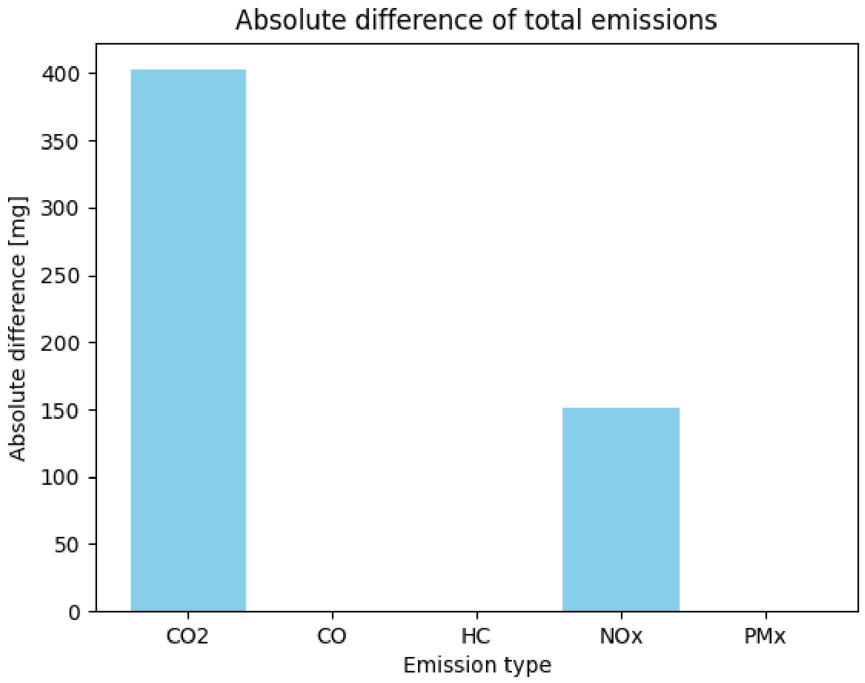


Figure 2: Plot output example showcasing the absolute difference of total emissions measured

Users should analyze these results to understand the potential emission reductions for various pollutants when start-stop technology is applied to a significant portion of the vehicle fleet. This data is most crucial to urban planners and researchers in evaluating the benefits of integrating such technologies into traffic management systems.

4.2 Vehicle Type Selection

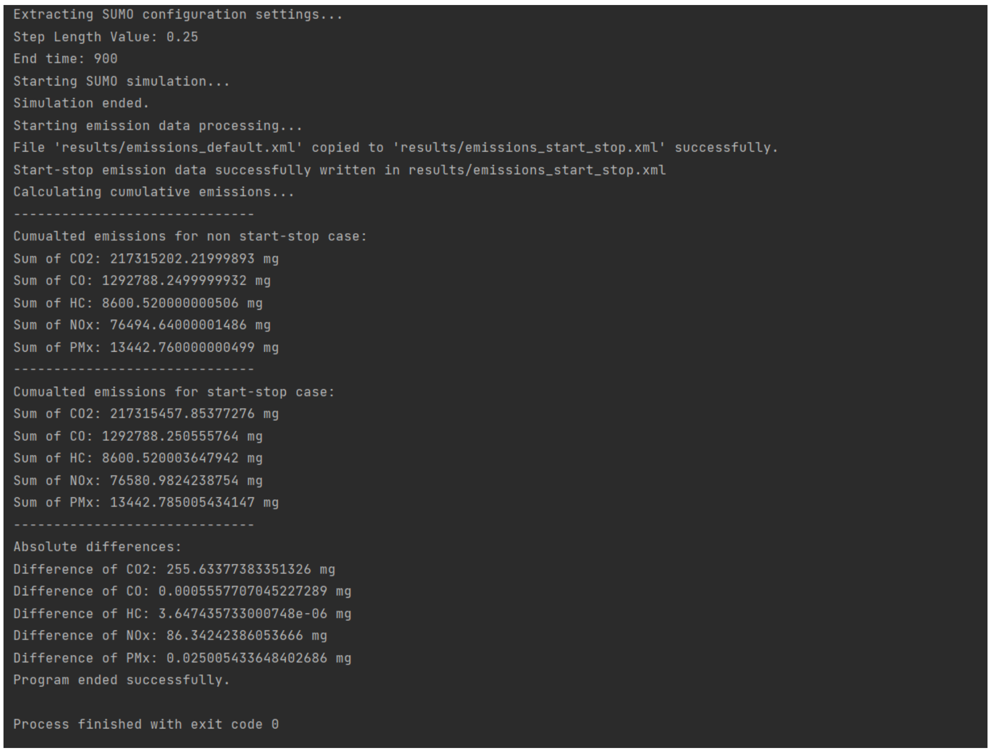
To set up a SUMO simulation using vehicle type selection for start-stop functionality, users will need to refer to the Customization Overview section of the provided documentation. In that section, it explains how to define vehicle types within SUMO's additional configuration files, specifying attributes, such as car-following model, lane change model, minimum headway (*minGap*), and emission class for each vehicle type, including those with start-stop capabilities.

For this simulation scenario using *"C:/One_lane_signalized_full_green/cfg_70_free.sumocfg,"* ensure that the *ratio_based_simulation* is set to False in the *main.py* script. This configuration will signal the program to identify start-stop vehicles by their predefined vehicle type within the SUMO configuration files, rather than by a percentage ratio. This method is suitable for users who wish to simulate traffic with specific vehicle types, enabling a detailed analysis of behavior and emissions under simulated traffic conditions with explicit start-stop vehicle definitions.

The results showcase the impact of start-stop technology on predefined vehicle types within the simulation. In the free traffic scenario using vehicle-type selection, the output (Fig. 3) will reflect the emissions data similar to the previous example but with nuances based on the distinct vehicle behaviors and characteristics.

The cumulative emissions data for the start-stop case, as seen in the provided results, will typically present a breakdown of various pollutants. The absolute differences in emissions between the standard and start-stop cases will be evident, allowing users to assess the efficacy of start-stop technology in reducing specific types of emissions.

Fig. 4 visualizes the absolute differences in emissions, which is an effective way for users to quickly compare the environmental performance of vehicles with start-stop systems against regular vehicles. Users can interpret these results to evaluate the potential of start-stop technology in contributing to emission reduction strategies and to inform traffic management policies.



```
Extracting SUMO configuration settings...
Step Length Value: 0.25
End time: 900
Starting SUMO simulation...
Simulation ended.
Starting emission data processing...
File 'results/emissions_default.xml' copied to 'results/emissions_start_stop.xml' successfully.
Start-stop emission data successfully written in results/emissions_start_stop.xml
Calculating cumulative emissions...
------------------------------
Cumualted emissions for non start-stop case:
Sum of CO2: 217315202.21999893 mg
Sum of CO: 1292788.2499999932 mg
Sum of HC: 8600.520000000506 mg
Sum of NOx: 76494.64000001486 mg
Sum of PMx: 13442.760000000499 mg
------------------------------
Cumualted emissions for start-stop case:
Sum of CO2: 217315457.85377276 mg
Sum of CO: 1292788.250555764 mg
Sum of HC: 8600.520003647942 mg
Sum of NOx: 76580.9824238754 mg
Sum of PMx: 13442.785005434147 mg
------------------------------
Absolute differences:
Difference of CO2: 255.63377383351326 mg
Difference of CO: 0.0005557707045227289 mg
Difference of HC: 3.647435733000748e-06 mg
Difference of NOx: 86.34242386053666 mg
Difference of PMx: 0.025005433648402686 mg
Program ended successfully.

Process finished with exit code 0
```

Figure 3: Ratio based output results

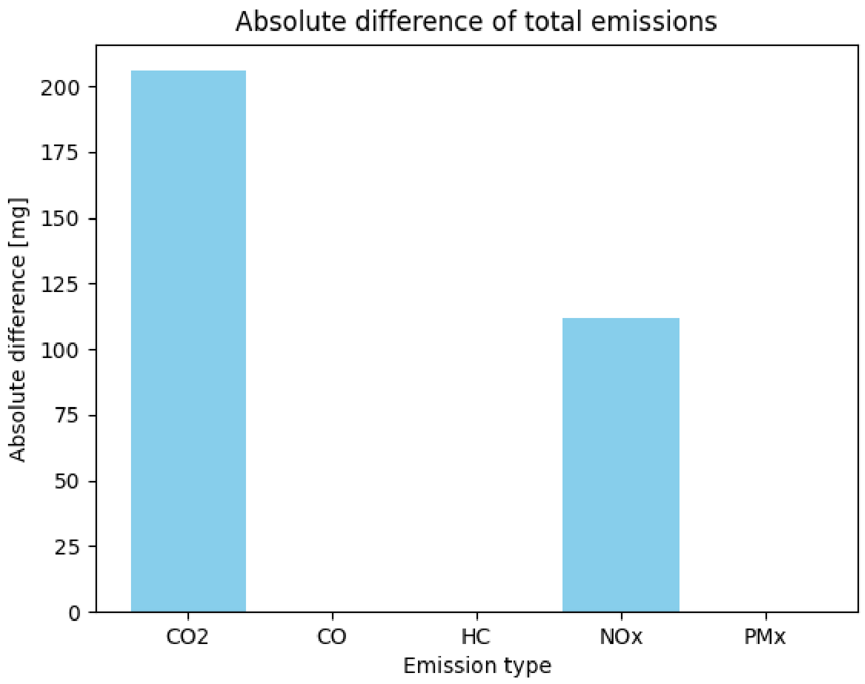


Figure 4: Vehicle-type output example

5 Limitations

5.1 Driver Behaviour Variability

The tool might not fully account for the wide range of human driving behaviors. While the tool can simulate general trends and provide valuable insights into the impact of start-stop technology, the complexity of human driving behaviors remains a challenge. These behaviors can fluctuate due to a multitude of factors, such as mood, traffic conditions, individual driving habits, and even the time of day. Obviously, such variability can result in some difference between simulated results and real-world outcomes.

5.2 Vehicle Type Representation

The tool's current framework primarily models conventional petrol vehicles, which poses a limitation in adequately representing Hybrid Electric Vehicles (HEVs). HEVs have complex powertrains that can switch between combustion and electric power, significantly altering their start-stop behavior and emissions output compared to traditional vehicles. This means the tool may not accurately simulate the reduced emissions, HEVs can achieve during electric-only operation or capture the nuances of their regenerative braking systems.

5.3 Regulatory and Environmental Standards

The use of the HBEFA4/PC petrol Euro-4 model for emissions estimation anchors it to a specific regulatory framework that may not align with the varied emissions standards across different regions. The tool's reliance on a particular emissions model means it may require adaptation to reflect the regulatory environment of a given region.

References

M. Gressai, B. Varga, T. Tettamanti, and I. Varga. Investigating the impacts of urban speed limit reduction through microscopic traffic simulation. *Communications in Transportation Research*, 1:100018, 2021. ISSN 2772-4247. doi: 10.1016/j.commtr.2021.100018. URL `https://www.sciencedirect.com/science/article/pii/S2772424721000184`.

M. Keller, S. Hausberger, C. Matzer, P. Wüthrich, and B. Notter. HBEFA Version 3.3. *Background Documentation, Berne*, 12, 2017.

A. Kovács, A. Leelőssy, T. Tettamanti, D. Esztergár-Kiss, R. Mészáros, and I.L. Lagzi. Coupling traffic originated urban air pollution estimation with an atmospheric chemistry model. *URBAN CLIMATE*, 37, 2021. ISSN 2212-0955. doi: 10.1016/j.uclim.2021.100868. URL `https://m2.mtmt.hu/api/publication/32002135`.

P.A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wiessner. Microscopic traffic simulation using SUMO. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2575–2582, 2018. doi: 10.1109/ITSC.2018.8569938.

A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J.-P. Hubaux. TraCI: An Interface for Coupling Road Traffic and Network Simulators. In *Proceedings of the 11th Communications and Networking Simulation Symposium*, CNS '08, pages 155–163, New York, NY, USA, 2008. ACM. ISBN 1-56555-318-7. doi: 10.1145/1400713.1400740. URL `http://doi.acm.org/10.1145/1400713.1400740`.

Fangxi Xie, Wei Hong, Yan Su, Qingnian Wang, Hang Zhu, and Chunshan Gu. Direct start of a gasoline direct-injection engine without a starter. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 230(4):491–502, 2016. doi: 10.1177/0954407015587403. URL `https://doi.org/10.1177/0954407015587403`.

6 Appendix: the full code of the Start-stop Program

The code is available at https://github.com/bmetrafficlab/SUMO_start_stop_system.

6.1 main.py

```python
import os
import startstop as Stp

"""
 Simulation settings:
 For your own simulation, provide your own SUMO configuration:
 - sumoCfg: Path to the SUMO simulation's configuration file
 - start_stop_ratio: Ratio of the start-stop vehicles in the simulation given in %
 - ratio_based_simulation: If True, the vehicles are assigned to the start-stop
   vehicles group by the given ratio, otherwise, the vehicle class type "start-stop-
   vehicle" is used
 - idle_values: Dictionary of idle emission values (CO2, CO, HC, NOx, PMx) based on
   the HBEFA4/PC_petrol_Euro-4 model. Selected default parameters can be overwritten
"""
# <-------------------- USER SETTINGS -------------------->
sumocfg = "C:/One_lane_signalized_full_green/cfg_70_free.sumocfg"
ratio_based_simulation = True
start_stop_ratio = 80
idle_time_in_sec = 7
idle_values = None
# Example parameter selection:
# idle_values = {'CO2': 1.8, 'CO': 3.0126e-12}
# <-------------------- END OF USER SETTINGS -------------------->

def main():
    """ Start-stop simulation functions. The user does not have to edit anything in
   the main function. """
    if start_stop_ratio > 100 or start_stop_ratio == 0:
        print("The start_stop_ratio must be larger than 0% and lower or equal to
   100%!")
        return
    if sumocfg:
        if os.path.exists(sumocfg):
            print("Extracting SUMO configuration settings...")

            # Extract the step length size from the given SUMO configuration
            stepsize = float(Stp.extract_step_length(sumocfg))
            duration = float(Stp.extract_duration(sumocfg))

            # Start the simulation
            print("Starting SUMO simulation...")
            Stp.run_simulation(sumocfg, duration, stepsize, start_stop_ratio,
   ratio_based_simulation, idle_time_in_sec, idle_values)
            return
        else:
            print("The given SUMO configuration file does not exist!")
            return
    else:
        print("Please provide a SUMO configuration file for the simulation.")
        return

if __name__ == "__main__":
    main()
```

6.2 startstop.py

```
import os
import xml.etree.ElementTree as ET
import traci
import matplotlib.pyplot as plt


def run_simulation(sumocfg, duration, steptime, start_stop_ratio,
    ratio_based_simulation, idle_time_in_sec, idle_values=None):
    """
     A function to start the SUMO simulation with the given SUMO configuration using
    the extracted SUMO settings.
     SUMO is set to dump the emission data at the end of the simulation. The emission
     dump is then copied and
     overwritten by the start-stop emission data. This way, both the non start-stop
    data and start-stop data are saved.
    """
    # Create results folder if it does not exist
    if not os.path.exists("results"):
        os.makedirs("results")
        print("Results folder was created.")

    default_idle_values = {'CO2': 1.4,
                           'CO': 6.556e-11,
                           'HC': 4.569000000000001e-13,
                           'NOx': 0.611700000001176,
                           'PMx': 7.192094822220001e-05}

    if idle_values is None:
        idle_values = {'CO2': 1.4,
                       'CO': 6.556e-11,
                       'HC': 4.569000000000001e-13,
                       'NOx': 0.611700000001176,
                       'PMx': 7.192094822220001e-05}
    else:
        emission_types = ['CO2', 'CO', 'HC', 'NOx', 'PMx']

        for type in emission_types:
            if type not in idle_values:
                idle_values[type] = default_idle_values[type]

        print(idle_values)

    # Set the SUMO command to start SUMO with GUI and dump emissions to the results
    folder
    sumocmd = ["sumo-gui", "-c", sumocfg, "--start", "--emission-output", "results/
    emissions_default.xml"]
    # Start SUMO with the command
    traci.start(sumocmd)

    # Actual simulation time
    simulation_time = 0.00

    # Idle values
    idle_co2 = idle_values['CO2']
    idle_co = idle_values['CO']
    idle_hc = idle_values['HC']
    idle_nox = idle_values['NOx']
    idle_pmx = idle_values['PMx']
    # Idle emissions
    idle_co2_s = idle_co2 * idle_time_in_sec
    idle_co_s = idle_co * idle_time_in_sec
    idle_hc_s = idle_hc * idle_time_in_sec
    idle_nox_s = idle_nox * idle_time_in_sec
    idle_pmx_s = idle_pmx * idle_time_in_sec

    # Data storages
    start_stop_data = {}
    regular_data = {}
    start_stop_vehicles = set()
    assigned_vehicles = set()
    total_vehicles_processed = 0
    vehicle_emission_data_per_step = {}

    stop_times = {}  # Dictionary to store stop times

    start_stop_threshold = start_stop_ratio / 100.0

    # Simulation steps
    for i in range(int(duration / steptime)):
        traci.simulationStep()
        current_vehicle_ids = set(traci.vehicle.getIDList())
        for vehicle_id in current_vehicle_ids:
            if vehicle_id not in assigned_vehicles:
                vehicle_type = traci.vehicle.getTypeID(vehicle_id)
                total_vehicles_processed += 1
                if ((len(start_stop_vehicles) / total_vehicles_processed) <
start_stop_threshold and
                        ratio_based_simulation):
                    start_stop_vehicles.add(vehicle_id)
                elif vehicle_type == 'start-stop-vehicle':
                    start_stop_vehicles.add(vehicle_id)
                    traci.vehicle.setColor(vehicle_id, (255, 0, 0, 255))
                assigned_vehicles.add(vehicle_id)

            # Read current values
            current_speed = traci.vehicle.getSpeed(vehicle_id)
            current_co2 = traci.vehicle.getCO2Emission(vehicle_id)
            current_co = traci.vehicle.getCOEmission(vehicle_id)
            current_hc = traci.vehicle.getHCEmission(vehicle_id)
            current_nox = traci.vehicle.getNOxEmission(vehicle_id)
            current_pmx = traci.vehicle.getPMxEmission(vehicle_id)

            # Track stop times
            if current_speed == 0:
                stop_times[vehicle_id] = stop_times.get(vehicle_id, 0) + steptime

            # Start-stop functionality
            if vehicle_id in start_stop_vehicles:
                if current_speed == 0:
                    if vehicle_id not in start_stop_data:
                        start_stop_data[vehicle_id] = {'time': simulation_time, '
speed': current_speed,
                                                       'co2': current_co2}
                    else:
                        start_stop_data[vehicle_id]['time'] += steptime

                    if vehicle_id not in vehicle_emission_data_per_step:
                        vehicle_emission_data_per_step[vehicle_id] = {}
                        vehicle_emission_data_per_step[vehicle_id][simulation_time] =
 {'CO2': current_co2,

  'CO': current_co,

  'HC': current_hc,

  'NOx': current_nox,

  'PMx': current_pmx}
                else:
                    vehicle_emission_data_per_step[vehicle_id][simulation_time] =
 {'CO2': current_co2,

  'CO': current_co,

  'HC': current_hc,

  'NOx': current_nox,

  'PMx': current_pmx}
            else:
                # Vehicle is moving
                if vehicle_id in start_stop_data:
                    # Check if it was stationary for more than 2 seconds
                    if start_stop_data[vehicle_id]['time'] >= 2:
                        adjusted_co2 = current_co2 + idle_co2_s
                        start_stop_data[vehicle_id]['co2'] = adjusted_co2

                    if vehicle_id not in vehicle_emission_data_per_step:
                        if start_stop_data[vehicle_id]['time'] >= 2:
                            vehicle_emission_data_per_step[vehicle_id] = {}
                            vehicle_emission_data_per_step[vehicle_id][
simulation_time] = {
                                'CO2': current_co2 + idle_co2_s,
                                'CO': current_co + idle_co_s,
                                'HC': current_hc + idle_hc_s,
                                'NOx': current_nox + idle_nox_s,
                                'PMx': current_pmx + idle_pmx_s}
                    else:
                        if start_stop_data[vehicle_id]['time'] >= 2:
                            vehicle_emission_data_per_step[vehicle_id][
simulation_time] = {
                                'CO2': current_co2 + idle_co2_s,
                                'CO': current_co + idle_co_s,
                                'HC': current_hc + idle_hc_s,
                                'NOx': current_nox + idle_nox_s,
                                'PMx': current_pmx + idle_pmx_s}
                    start_stop_data[vehicle_id]['time'] = 0
                    start_stop_data[vehicle_id]['speed'] = current_speed
        else:
            # Regular vehicle data collection
            regular_data[vehicle_id] = {'time': simulation_time, 'speed':
current_speed, 'co2': current_co2}

        simulation_time = i * steptime

# Close TraCI
traci.close()

print("Simulation ended.")
print("Starting emission data processing...")
```

```
    # Handle emission results
    if create_start_stop_emissions(vehicle_emission_data_per_step, stop_times):
        print("Start-stop emission data successfully written in results/
    emissions_start_stop.xml")
    else:
        print("Error in emission results handling!")
        return

    print("Calculating cumulative emissions...")
    calculate_cumulative_emissions()

    print("Program ended successfully.")
    return


def extract_step_length(sumocfg):
    """
     A function to extract the step-length setting from the SUMO configuration file
    given by the user.
     If the configuration does not contain the step-length, it defaults to 1.
    """
    tree = ET.parse(sumocfg)
    root = tree.getroot()
    step_length_value = 1

    for time_element in root.iter('time'):
        step_length = time_element.find('step-length')
        if step_length is not None:
            step_length_value = step_length.attrib.get('value')
            print("Step Length Value:", step_length_value)
        else:
            print("Step Length Value set to default:", step_length_value)
    return step_length_value


def extract_duration(sumocfg):
    """
     A function to extract the end time setting from the SUMO configuration file
    given by the user. If the configuration
     does not contain the end time, it defaults to 1000.
    """
    tree = ET.parse(sumocfg)
    root = tree.getroot()
    duration_value = 1000

    for time_element in root.iter('time'):
        duration = time_element.find('end')
        if duration is not None:
            duration_value = duration.attrib.get('value')
            print("End time:", duration_value)
        else:
            print("End time defaults to:", duration_value)
    return duration_value


def create_start_stop_emissions(start_stop_data, stop_times):
    """
    This function copies the original SUMO emissions XML file and replaces CO2
    emission data for start-stop vehicles.
    :param start_stop_data: Contains data of the start-stop vehicles (dictionary)
    :param stop_times: Contains the stop time information for vehicles (dictionary)
    :return: Returns True when successfully done, or False on errors.
    """
    if start_stop_data and stop_times:
        original_emissions_file = "results/emissions_default.xml"
        start_stop_emissions_file = "results/emissions_start_stop.xml"

        # Copy the original emissions file
        if os.path.exists(original_emissions_file):
            if copy_file(original_emissions_file, start_stop_emissions_file):
                tree = ET.parse(start_stop_emissions_file)
                root = tree.getroot()

                for timestep in root.findall('timestep'):
                    # Extract time attribute
                    time = timestep.attrib['time']

                    # Iterate through vehicle elements
                    for vehicle in timestep.findall('vehicle'):
                        # Extract id attribute
                        vehicle_id = vehicle.attrib['id']
                        vehicle_id = str(vehicle_id)

                        # Find the vehicle id with the current timestep in the start-
    stop data
                        if start_stop_data is not None:
                            vehicle_start_stop = start_stop_data.get(vehicle_id, None
    )
                            if vehicle_start_stop is not None:
                                selected_vehicle_id = vehicle_start_stop.get(float(
    time), None)
                                if selected_vehicle_id is not None:
                                    vehicle.attrib['CO2'] = str(selected_vehicle_id['
    CO2'])
                                    vehicle.attrib['CO'] = str(selected_vehicle_id['
    CO'])
                                    vehicle.attrib['HC'] = str(selected_vehicle_id['
    HC'])
                                    vehicle.attrib['NOx'] = str(selected_vehicle_id['
    NOx'])
                                    vehicle.attrib['PMx'] = str(selected_vehicle_id['
    PMx'])

                                else:
                                    break
                # Write the modified XML back to file
                tree.write('results/emissions_start_stop.xml')
                return True
            else:
                return False
        else:
            print("Missing original emissions file! (results/emissions_default.xml)")
            return False
    else:
        print("Missing data in emission handling!")
        return False


def copy_file(original_file, new_file):
    """
    This function is to copy the original emission file before modifying it with the
    stop-start data.
    :param original_file: The original file to copy.
    :param new_file: The new file's path and extension.
    :return: True on success and False on failure.
    """
    try:
        with open(original_file, 'rb') as f_original:
            with open(new_file, 'wb') as f_new:
                # Read from the original file and write to the new file
                f_new.write(f_original.read())
        print(f"File '{original_file}' copied to '{new_file}' successfully.")
        return True
    except FileNotFoundError:
        print("One of the files doesn't exist.")
        return False


def calculate_cumulative_emissions():
    # Parse the XML file
    tree = ET.parse('results/emissions_default.xml')
    root = tree.getroot()

    # Initialize sums for each emission type
    sum_CO2 = 0
    sum_CO = 0
    sum_HC = 0
    sum_NOx = 0
    sum_PMx = 0

    # Iterate through each timestep
    for timestep in root.findall('timestep'):
        # Iterate through each vehicle within the timestep
        for vehicle in timestep.findall('vehicle'):
            # Extract and sum the values for each emission type
            sum_CO2 += float(vehicle.get('CO2'))
            sum_CO += float(vehicle.get('CO'))
            sum_HC += float(vehicle.get('HC'))
            sum_NOx += float(vehicle.get('NOx'))
            sum_PMx += float(vehicle.get('PMx'))

    # Print the sum of each emission type
    print("------------------------------")
    print("Cumulated emissions for non start-stop case:")
    print("Sum of CO2:", sum_CO2, "mg")
    print("Sum of CO:", sum_CO, "mg")
    print("Sum of HC:", sum_HC, "mg")
    print("Sum of NOx:", sum_NOx, "mg")
    print("Sum of PMx:", sum_PMx, "mg")

    # Parse the XML file
    tree = ET.parse('results/emissions_start_stop.xml')
    root = tree.getroot()

    # Initialize sums for each emission type
    sum_CO2_start_stop = 0
    sum_CO_start_stop = 0
    sum_HC_start_stop = 0
    sum_NOx_start_stop = 0
    sum_PMx_start_stop = 0

    # Iterate through each timestep
    for timestep in root.findall('timestep'):
        # Iterate through each vehicle within the timestep
        for vehicle in timestep.findall('vehicle'):
            # Extract and sum the values for each emission type
            sum_CO2_start_stop += float(vehicle.get('CO2'))
            sum_CO_start_stop += float(vehicle.get('CO'))
            sum_HC_start_stop += float(vehicle.get('HC'))
            sum_NOx_start_stop += float(vehicle.get('NOx'))
            sum_PMx_start_stop += float(vehicle.get('PMx'))

    # Print the sum of each emission type
    print("-------------------------------")
    print("Cumulated emissions for start-stop case:")
    print("Sum of CO2:", sum_CO2_start_stop, "mg")
    print("Sum of CO:", sum_CO_start_stop, "mg")
    print("Sum of HC:", sum_HC_start_stop, "mg")
    print("Sum of NOx:", sum_NOx_start_stop, "mg")
    print("Sum of PMx:", sum_PMx_start_stop, "mg")
    print("-------------------------------")
    print("Absolute differences:")
    print("Difference of CO2:", abs(sum_CO2_start_stop - sum_CO2), "mg")
    print("Difference of CO:", abs(sum_CO_start_stop - sum_CO), "mg")
    print("Difference of HC:", abs(sum_HC_start_stop - sum_HC), "mg")
    print("Difference of NOx:", abs(sum_NOx_start_stop - sum_NOx), "mg")
    print("Difference of PMx:", abs(sum_PMx_start_stop - sum_PMx), "mg")

    categories = ['CO2', 'CO', 'HC', 'NOx', 'PMx']
    values = [abs(sum_CO2_start_stop - sum_CO2), abs(sum_CO_start_stop - sum_CO), abs
    (sum_HC_start_stop - sum_HC),
              abs(sum_NOx_start_stop - sum_NOx), abs(sum_PMx_start_stop - sum_PMx)]
    # Create bar graph
    plt.bar(categories, values, color='skyblue')

    # Adding title and labels
    plt.title('Absolute difference of total emissions')
    plt.xlabel('Emission type')
    plt.ylabel('Absolute difference [mg]')

    # Show the plot
    plt.show()
    return
```