Funded by the European Union
NextGenerationEU
CZECH RECOVERY PLAN
MINISTRY OF EDUCATION, YOUTH AND SPORTS

Demonstration of codes for SNF-A

Ivan Eryganov

FACULTY OF MECHANICAL ENGINEERING
institute of mathematics

PROJECT "ACCELERATION OF GREEN SKILLS AND SUSTAINABILITY AT BUT".
REGISTRATION NUMBER NPO_VUT_MSMT-2143/2024-5.

Description

- The code in Jupyter Notebook plans workinng day of bikesharing rebalancing operator for Brno.
- It works with a dynamic network flow model and data about rentals.
- The optimization is solved using pyomo language for modelling mathematical programming problems.
- Different solvers can be used.

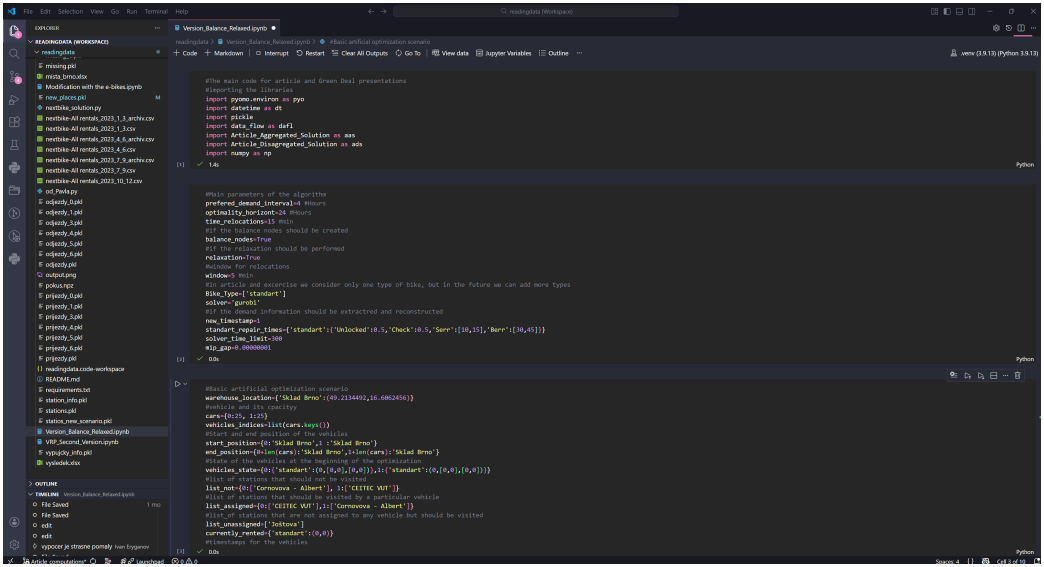


Figure: Reading the libraries, setting the parameters of calculation

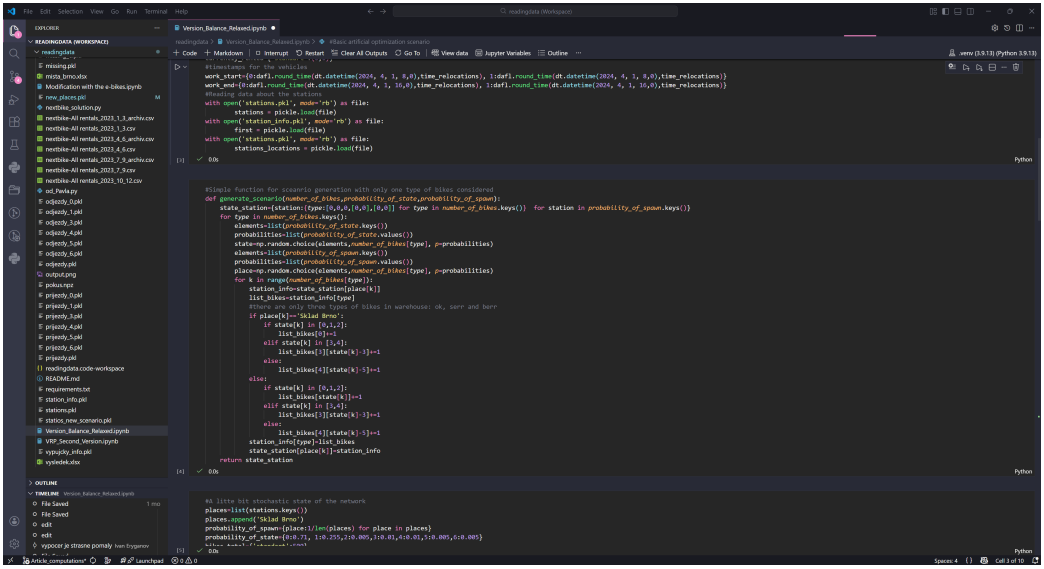
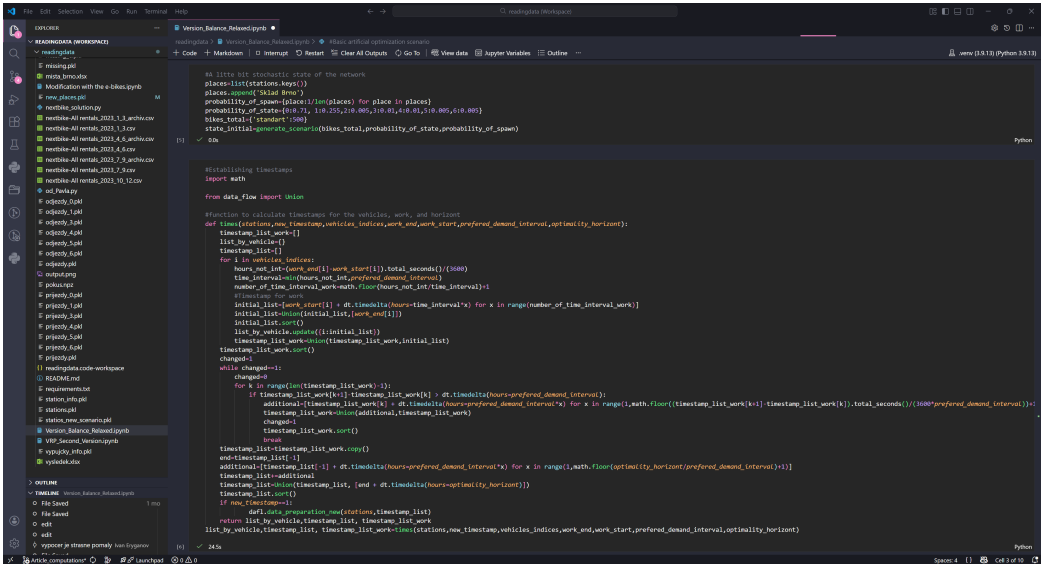


Figure: Generation of scenarios to test robustness



```
#A little bit stochastic state of the network
places=list(stations.keys())
places.append('Sklad Brno')
probability_of_spawn={place:1/len(places) for place in places}
probability_of_state={0:0.71, 1:0.255,2:0.005,3:0.01,4:0.01,5:0.005,6:0.005}
bikes_total={'standart':500}
state_initial=generate_scenario(bikes_total,probability_of_state,probability_of_spawn)
```

```
2.0s
Python
```

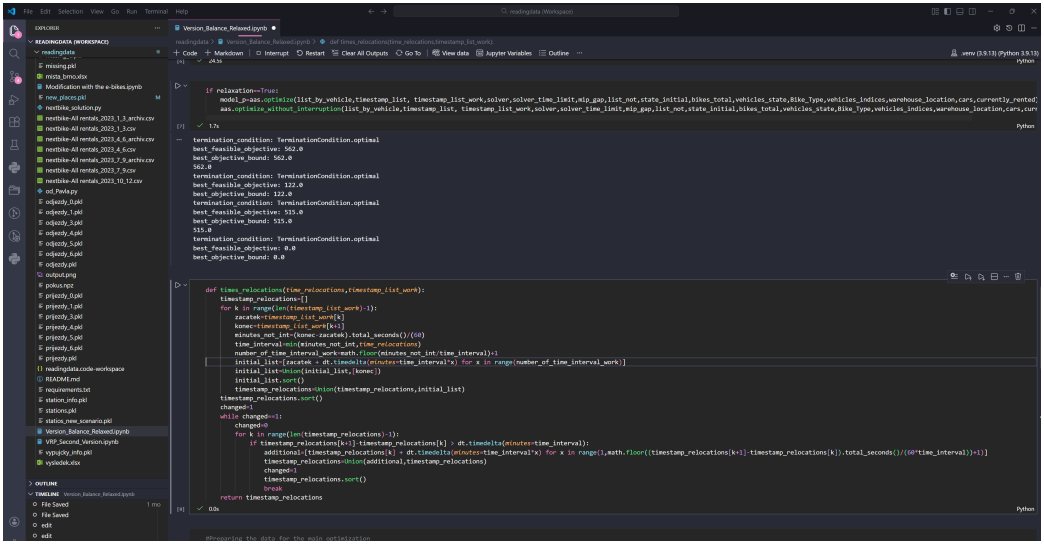
```
#Establishing timestamps
import math

from data_flow import Union

#Function to calculate timestamps for the vehicles, work, and horizont
def times(stations,new_timestamp,vehicles_indices,work_end,work_start,prefered_demand_interval,optimality_horizont):
    timestamp_list_work=[]
    list_by_vehicle={}
    timestamp_list=[]
    for i in vehicles_indices:
        hours_not_int=(work_end[i]-work_start[i]).total_seconds()/(3600)
        time_interval=min(hours_not_int,prefered_demand_interval)
        number_of_time_interval_work=math.floor(hours_not_int/time_interval)+1
        #timestamps for work
        initial_list=[work_start[i] + dt.timedelta(hours=time_interval*x) for x in range(number_of_time_interval_work)]
        initial_list=Union(initial_list,[work_end[i]])
        initial_list.sort()
        list_by_vehicle.update({i:initial_list})
        timestamp_list_work=Union(timestamp_list_work,initial_list)
    timestamp_list_work.sort()
    changed=1
    while changed==1:
        changed=0
        for k in range(len(timestamp_list_work)-1):
            if timestamp_list_work[k+1]-timestamp_list_work[k] > dt.timedelta(hours=prefered_demand_interval):
                additional=[timestamp_list_work[k] + dt.timedelta(hours=prefered_demand_interval*x) for x in range(1,math.floor((timestamp_list_work[k+1]-timestamp_list_work[k]).total_seconds()/(3600*prefered_demand_interval))+1
                timestamp_list_work=Union(additional,timestamp_list_work)
                changed=1
                timestamp_list_work.sort()
                break
    timestamp_list=timestamp_list_work.copy()
    end=timestamp_list[-1]
    additional=[timestamp_list[-1] + dt.timedelta(hours=prefered_demand_interval*x) for x in range(1,math.floor(optimality_horizont/prefered_demand_interval)+1)]
    timestamp_list=additional
    timestamp_list=Union(timestamp_list, [end + dt.timedelta(hours=optimality_horizont)])
    timestamp_list.sort()
    if new_timestamp==1:
        daf1.data_preparation_new(stations,timestamp_list)
    return list_by_vehicle,timestamp_list, timestamp_list_work
list_by_vehicle,timestamp_list, timestamp_list_work=times(stations,new_timestamp,vehicles_indices,work_end,work_start,prefered_demand_interval,optimality_horizont)
```

```
24.5s
Python
```

Figure: Creation of timestamps and pulling the required data about demand

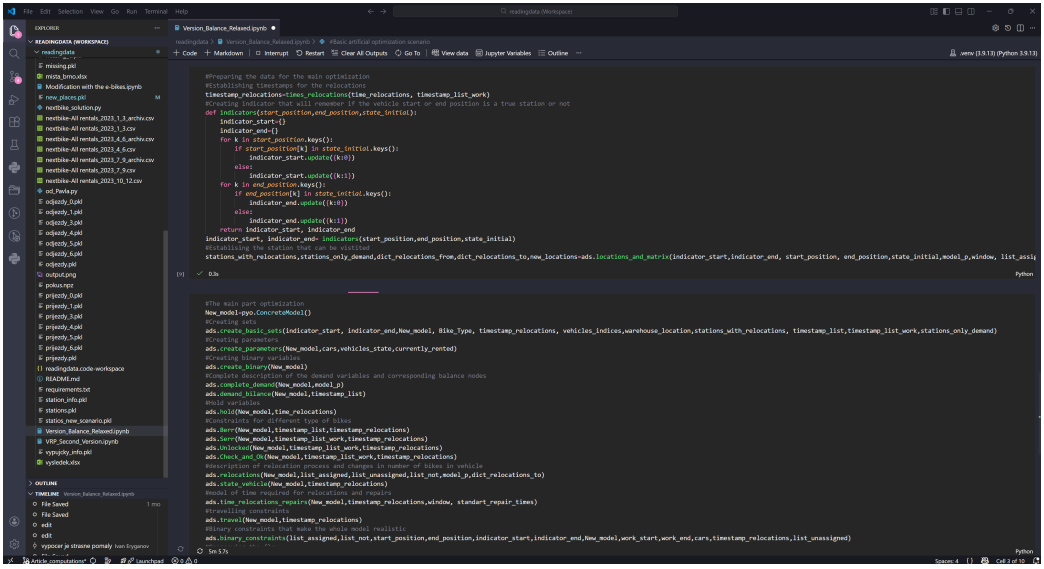


```
if relaxation==True:
    model_p=aas.optimize(list_by_vehicle,timestamp_list, timestamp_list_work,solver,solver_time_limit,mip_gap,list_not,state_initial,bikes_total,vehicles_state,Bike_Type,vehicles_indices,warehouse_location,cars,currently_rented)
    aas.optimize_without_interruption(list_by_vehicle,timestamp_list, timestamp_list_work,solver,solver_time_limit,mip_gap,list_not,state_initial,bikes_total,vehicles_state,Bike_Type,vehicles_indices,warehouse_location,cars,curr
```

```
termination_condition: TerminationCondition.optimal
best_feasible_objective: 562.0
best_objective_bound: 562.0
562.0
termination_condition: TerminationCondition.optimal
best_feasible_objective: 122.0
best_objective_bound: 122.0
termination_condition: TerminationCondition.optimal
best_feasible_objective: 515.0
best_objective_bound: 515.0
515.0
termination_condition: TerminationCondition.optimal
best_feasible_objective: 0.0
best_objective_bound: 0.0
```

```
def times_relocations(time_relocations,timestamp_list_work):
    timestamp_relocations=[]
    for k in range(len(timestamp_list_work)-1):
        zacatek=timestamp_list_work[k]
        konec=timestamp_list_work[k+1]
        minutes_not_int=(konec-zacatek).total_seconds()/(60)
        time_interval=min(minutes_not_int,time_relocations)
        number_of_time_interval_work=math.floor(minutes_not_int/time_interval)+1
        initial_list=[zacatek + dt.timedelta(minutes=time_interval*x) for x in range(number_of_time_interval_work)]
        initial_list=Union(initial_list, [konec])
        initial_list.sort()
        timestamp_relocations=Union(timestamp_relocations,initial_list)
    timestamp_relocations.sort()
    changed=1
    while changed==1:
        changed=0
        for k in range(len(timestamp_relocations)-1):
            if timestamp_relocations[k+1]-timestamp_relocations[k] > dt.timedelta(minutes=time_interval):
                additional=[timestamp_relocations[k] + dt.timedelta(minutes=time_interval*x) for x in range(1,math.floor((timestamp_relocations[k+1]-timestamp_relocations[k]).total_seconds()/(60*time_interval))+1)]
                timestamp_relocations=Union(additional,timestamp_relocations)
                changed=1
                timestamp_relocations.sort()
                break
    return timestamp_relocations
```

Figure: Pre-optimization to establish the promising stations to visit, creation of timestamps for relocations



```
#Preparing the data for the main optimization
#Establishing timestamps for the relocations
timestamp_relocations=times_relocations(time_relocations, timestamp_list_work)
#Creating indicator that will remember if the vehicle start or end position is a true station or not
def indicators(start_position,end_position,state_initial):
    indicator_start={}
    indicator_end={}
    for k in start_position.keys():
        if start_position[k] in state_initial.keys():
            indicator_start.update({k:0})
        else:
            indicator_start.update({k:1})
    for k in end_position.keys():
        if end_position[k] in state_initial.keys():
            indicator_end.update({k:0})
        else:
            indicator_end.update({k:1})
    return indicator_start, indicator_end
indicator_start, indicator_end= indicators(start_position,end_position,state_initial)
#Establising the station that can be visitied
stations_with_relocations,stations_only_demand,dict_relocations_from,dict_relocations_to,new_locations=ads.locations_and_matrix(indicator_start,indicator_end, start_position, end_position,state_initial,model_p,window, list_assig
```

```
#The main part optimization
New_model=pyo.ConcreteModel()
#Creating sets
ads.create_basic_sets(indicator_start, indicator_end,New_model, Bike_Type, timestamp_relocations, vehicles_indices,warehouse_location,stations_with_relocations, timestamp_list,timestamp_list_work,stations_only_demand)
#Creating parameters
ads.create_parameters(New_model,cars,vehicles_state,currently_rented)
#Creating binary variables
ads.create_binary(New_model)
#Complete description of the demand variables and corresponding balance nodes
ads.complete_demand(New_model,model_p)
ads.demand_bilance(New_model,timestamp_list)
#Hold variables
ads.hold(New_model,time_relocations)
#Constraints for different type of bikes
ads.Berr(New_model,timestamp_list,timestamp_relocations)
ads.Serr(New_model,timestamp_list_work,timestamp_relocations)
ads.Unlocked(New_model,timestamp_list_work,timestamp_relocations)
ads.Check_and_Ok(New_model,timestamp_list_work,timestamp_relocations)
#Description of relocation process and changes in number of bikes in vehicle
ads.relocations(New_model,list_assigned,list_unassigned,list_not_model_p,dict_relocations_to)
ads.state_vehicle(New_model,timestamp_relocations)
#model of time required for relocation and repairs
ads.time_relocations_repairs(New_model,timestamp_relocations,window, standart_repair_times)
#travelling constraints
ads.travel(New_model,timestamp_relocations)
#binary constraints that make the whole model realistic
ads.binary_constraints(list_assigned,list_not,start_position,end_position,indicator_start,indicator_end,New_model,work_start,work_end,cars,timestamp_relocations,list_unassigned)
```

Figure: The main optimization routine

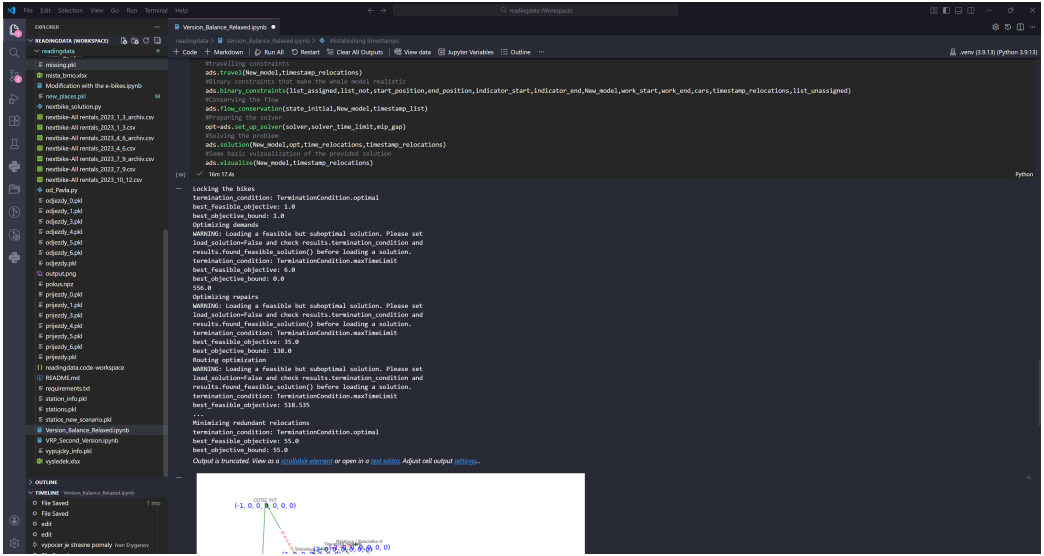


Figure: Log of the optimization

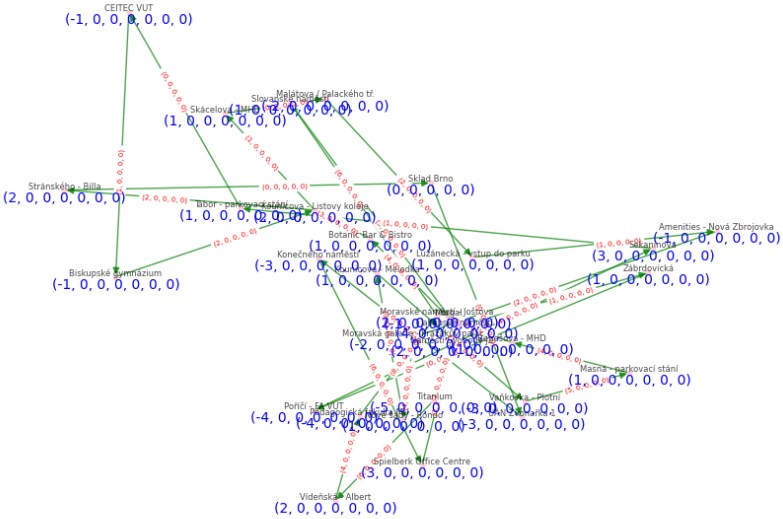


Figure: Basic vizualization of operations and route