BTW: Light The Traffic

Liran Farhi, Yair Feldman, Dor Granat, Tomer Kay, Eylon Shoshan, Ran Yeheskel.

ABSTRACT

Reducing waiting time in urban roads is a major goal for many applications and organizations. One approach to achieve this goal is from a municipal point of view. Specifically, the city controls its traffic light and schedules them in a clever way. Such cities are called smart cities.

Our project implements a smart city with simulated traffic, city hall interface, testing environment and a scheduling algorithm for the traffic lights. The entire project is written in python and uses open source projects such as SUMO and TraCI. Our scheduling algorithm is based on an academic paper that addresses the same problem.

Evaluating our scheduling algorithm showed a significant improvement in traffic throughput and waiting time of smart cities compared to non-smart cities. In addition, we showed our scheduling algorithm is better than a simple scheduling algorithm that utilizes the data from the smart city's sensors in a simple way.

I. INTRODUCTION

Driving is an essential part of every man's life. Almost every day we use are driving a car or using public transportation. The influence of efficient traffic control and flow on our lives is huge. The immediate impact is of course saving time in driving and prevent wasting time on the roads. But less obvious implications are also environmental and economical.

Pollution caused by congested conditions is much higher, compared to uncongested conditions [1]. Pollution is known to be a major public-health issue. Moreover it is known cause cancer and lunge diseases [2]. In addition, high pollution has also an environmental impact. This is including damage to other living creatures such as livestock, plants and even bacteria [3].

In addition, there is also an economical influence to increasing traffic flow and reduce congestion. One such example is employment growth as shown in [4].

Applications to minimize one's driving time are already exist. A few examples are Waze and Google Maps. These applications are designed to save the driving time for a single person. These applications are used by millions of users and do their job very good.

One problem with their approach is the "local" approach to minimize driving time, i.e. minimize a single driver's driving time. There are many factors which cannot be controlled and overcome by this approach. Among these factors are traffic accidents, bad traffic flow on the road and many more.

Therefore the "local" approach is inherently limited and cannot increase the traffic flow by a massive scale. A different approach is called for to improve the traffic flow not of a single driver but of much bigger scale.

The eager supporters of the local approach could claim that a central computer taking into account millions of "local" users, could increase traffic flow by massive scale. As a consequence, and no new approach is needed.

The problem with the previous claim is that the influence of the drivers is inherently limited to change the traffic flow. Of course, the impact of millions of drivers on the traffic flow is incredible, but will always be limited and relies on non-reliable individuals. Therefore it's inevitable that a new approach to increase traffic flow is needed.

One such approach is the "global" approach. This approach, taking into account the traffic flow of many drivers but not as many individuals but as one mass. An implementation of this approach is a "smart city" [5]. A smart city is a city that controls its traffic flow with the use of modern technology such as clever hardware (e.g. sensors) and software (e.g. clever traffic control algorithms).

The main example of a smart city's functionality is a sophisticated control of the city's traffic lights. Traffic lights are a major bottle neck in most cities. However, the vast majority of cities aren't trying to optimize the traffic flow in traffic-lighted junctions.

Since a clever traffic light control influence all the drivers as one mass, it has the ability to significantly increase the traffic flow in the city. We all have encountered terribly scheduled traffic lights that cause major traffic jams that could have been easily avoided.

This is exactly what a smart city is trying to change. Such cities understood the importance of efficient traffic light scheduling in order to reduce traffic jams and overall driving time.

More specifically, a smart city is consist of a smart junctions, which are monitored by sensors and are

controlled by a computer brain. The sensors' purpose is to supply the input to the computer brain of the junction, e.g. how many vehicles are in each lane, how many crossing the junction per each second and so on. The computer brain's purpose is to decide the best traffic light scheduling at any moment.

However, single junction optimizations aren't enough. A smart city also optimizes the traffic flow of many junctions. This is achieved by a communications between the different junctions (directly between them or indirectly via a central computer).

Our project is a (partial) implementation of a smart city. Our city, traffic, and sensors monitoring the junctions are all simulated. In addition, each traffic-lighted junction is scheduled by our algorithm.

In the following parts we will describe related work regarding smart cities and smart scheduling of traffic light junctions. In addition, we will describe our design and implementation of our project. Finally, we will show an evaluation of our project's performance and conclusions about our accomplishments.

II. RELATED WORK

Some papers also tries to find a good scheduling algorithms for smart cities. Some solutions differ in hardware of the sensors and the algorithm.

For example [6] introduces a new architecture for a smart city that should improve costs and performance for large cities. Since our focus is more software oriented we won't cover such papers.

The paper [7] describes their project, which implements a smart city with IoT sensors monitored with a user friendly Android app. Their algorithm is based on asking multiple queries and choosing the best schedule by the top k results. They tested their algorithms on real data from cities and also on synthetic data.

The scheduling algorithm presented in [8] surpasses its predecessors. In this paper the algorithm was tested in SUMO environment and obtained great results. They also tested their algorithm on a real city (Singapore) and on synthetic data created by SUMO.

III. DESIGN AND IMPLEMENTATION

In general overview, we can divide our project into three main parts: the simulator, city hall's interface, and the scheduling algorithm. The simulator generates the simulated traffic of our city, and displays it visually. The city hall's interface can get real time statistics on the current traffic flow and compare different algorithms to optimize the traffic flow. The scheduling algorithm manages the traffic lights in the entire city by the input it receives from the simulation's traffic sensors. We will now describe in detail how exactly we implemented each part. We wrote our entire project in python.

The simulation is based on SUMO- Simulation of Urban Mobility. This is a big open source project that designed to handle big road networks (such as a city), simulate traffic on them, and display the simulation visually on a map with a GUI.

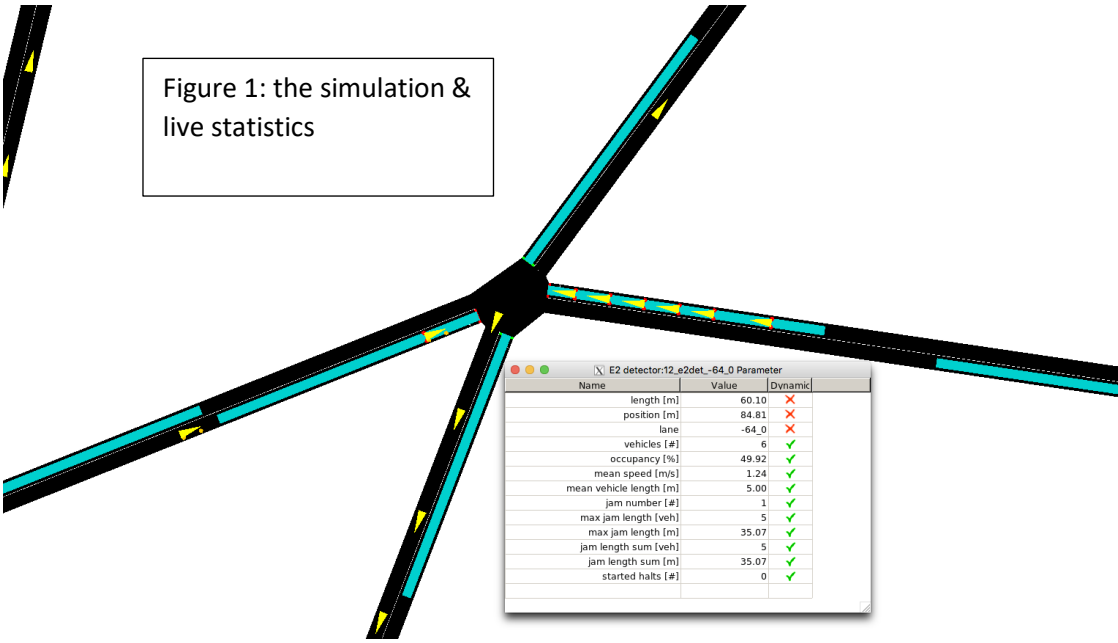


Figure 1: the simulation & live statistics

In addition it can export the trip info of all the vehicles in the simulations. This information consists of distance traveled, amount of gas consumed, time spent in waiting at a red light and so on.

SUMO can be accessed via TraCI which is a TCP based client/server architecture that provide access to SUMO. All the communication with sumo is done by the TraCI interface.

In order to create a SUMO simulation, these is a need to feed it with a configuration file, which configures some parameters of the simulation such as input of the road network and desired output. The road network is

given as XML file with a specific structure. Some open source projects such as OSM (Open Street Map) can take a real city and output such XML file which could be feed into SUMO. Since big cities are hard to monitor and test we used in our simulations just one junction with incoming traffic from two opposite sides.

SUMO schedules its traffic lights with a "static schedule", i.e. each lane get an equal amount of time to cross the junction (green light) regardless of traffic congestion and overload on each lane. In order to change that, a better understanding of the simulation is needed. The simulation is divided into steps and each step could be simulated via the TraCI interface. Between these steps a specially designed scheduling algorithm could be invoked. This is how we invoked our own smart scheduling of the traffic lights like a real smart city.

At the end of the simulation, an XML file is exported with the trip info, this file is very helpful in the evaluation of our scheduling algorithms and in the statistics to the city hall's interface.

In our project we don't want the simulation to be the only process running. The city hall's requirement is that we could get real time statistics on the current traffic flow in the city. The major difference between simulated traffic to evaluate the scheduling algorithm, and the simulated traffic of a real city is that in a real city traffic never stops and the road network is much more complex. Therefore, the city hall's interface has to run along with the SUMO simulator. In order to achieve that we created two threads: one with runs the simulator and the second the city hall's interface. As a consequence, synchronization between the two threads is necessary.

The second logical part of our project is the city hall's interface. Currently this interface has two main functionalities: real time statistics of the traffic flow, and statistics of the entire simulation- in order to evaluate and test the suggested scheduling algorithm.

In order to achieve the first functionality- the real time statistics, changing the sumo simulator was needed. The real time statistics are accessed by a right mouse click on a desired junction (See Figure 1).

This functionality was achieved by changing the SUMO code so that an extra tab will open on a mouse click and the desired action will occur on clicking it. Since SUMO is an open source project this was possible without using reverse engineering tools. Still, since SUMO is millions lines of code this wasn't an easy task. We had to find the part of the code that displays these tabs on right mouse click and add another tab. In addition we had to find the part of the code that operates when clicking on such a tab. This search was done by a smart search of strings in the code, and debugging to see on certain actions, what part of the code is running.

Moreover, since access to the SUMO simulation is granted with TraCI, we had make a connection with the SUMO app and our own city hall interface. But instead of using TraCI, we used a "simple" TCP connection with python sockets. We chose to implement the connection and not changing TraCI code as well, in order to integrate our new functionality.

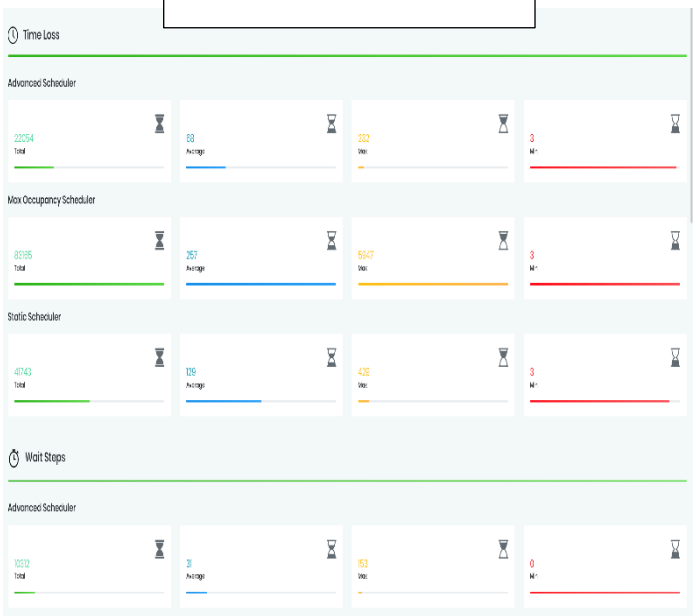
Changing the SUMO code isn't enough to get real time statistics. The city hall's interface has to receive information from the SUMO simulation via the server client connection established (as explained in the previous paragraph). Afterwards, parsing the information and making various calculations is needed.

The real time information is given from TraCI, and this kind of information is used in the real time statistics of the city hall's interface and in the scheduling algorithm as well. In order to modulate our code, we divided this real time information into three logical parts: City, Junction, and Detector.

We will now describe each of the three classes. City is the collection of junctions of the entire city, Junction is one junction with one or more traffic lights. It contains the traffic lights and detectors in that junction. Detector is a single sensor that detects traffic in a specific lane.

The second functionality of the city hall's interface is the statistics of the entire simulation when it ends. As mentioned before this is to evaluate and test the smart scheduling algorithms. Currently, the user's interface is a simple web application that only displays the statistics and not the simulation itself. The simulation is still presented in the SUMO app.

Figure 2: The Web interface



This feature was implemented as follows: the input is the XML file contains the simulation log. The XML file is then parsed to several collections containing data such as waiting time of the cars, number of cars participated, gas consumption and more. On these collections various statistics are calculated (e.g., average, median, maximum). There is a possibility to compare several XML files representing different simulations. This feature is useful to compare the performances of different traffic light scheduling algorithms.

The third part of our project is the scheduling algorithm. The algorithm decides at each step of the simulation which traffic lights will be green and which will be red. This part of the "back-end" of the project, i.e. the user is oblivious to the scheduling algorithm.

From the implementation's point of view, the scheduler is an abstract class that initialized with a City (a class described above). In addition, a scheduler class has to implement a schedule method that upon calling, schedules the traffic lights.

All operations on traffic lights, e.g. setting a traffic light green, check whether a traffic light is green, receive input from a detector, is done via the class methods of the object. The class method operations are implemented with TraCI.

We consider several scheduling algorithms, which will be presented in the following paragraphs. This will conclude the implementation description.

The first algorithm is the Static Scheduler. This is the default scheduling algorithm of SUMO. This algorithm gives each lane in each junction a constant amount of green-light time, in a round robin way. There are two main observations to this scheduler: it schedules each junction independently of the other junctions. In addition, it does not use the input from the detectors on the lanes. Therefore, this scheduling algorithm corresponds with the scheduling algorithm of a non-smart city (regular city).

The second algorithm, is the Max Occupancy Scheduler. This is a relatively simple algorithm. At each step it chooses the most occupied lane to be scheduled next, i.e. grants a green light to the lane with the most cars. There are three important observations to this algorithm. First, this algorithm schedules each junction independently of the other junctions. Second, this algorithm does use the information from the detectors in order to decide which lane is the most occupied. Thus this is the simplest algorithm that suits the definition of a smart city. Third, this algorithm may cause starvation of a specific lane. If a certain lane is never the most occupied lane in a junction it will never be given a green light.

The third algorithm is called Mixed Scheduler. This algorithm is a composition of several scheduling algorithms, i.e. each junction is scheduled independently of the other junctions and by a different algorithm. Since in a smart city some traffic-lighted junctions might still be regular (i.e. scheduled statically) this is a very important approach in every project that simulates a smart city.

Finally, the forth algorithm is the best one we have, and the most sophisticated one. The solution is based on [6]. The idea is to maximize the throughput of the traffic under the junction and in addition, to prevent starvation to lanes with less traffic.
In order to achieve that the traffic lights are scheduled in a round-robin way. Each junction is scheduled independently of the other junctions.

Before the algorithm itself we need several definitions and notations:
1) A cycle for traffic light i as the time between two consecutive times i became green.
2) An epoch is one iteration of the round-robin scheduling.
3) f_i is the rate of cars arriving to the ith lane in the junction.
4) fe_i is the rate of cars exiting the ith lane (when the lane is green).
5) T_i is the cycle length of the ith traffic light.
6) M is a mapping between a traffic light i and the other traffic lights that can be green with i at the same time.

The scheduling algorithm:

1. epoch ← {all traffic lights}
2. find the traffic light with maximal f_i in epoch. If $f_i = 0$, return to (1).
3. search in M[i] another traffic light with the maximal f_j.
4. schedule i, j for $T_i \cdot \frac{f_i}{fe_i}$ time units. (turn the other lights to red, and then i, j to green).
5. epoch ← epoch \ {i,j}. If epoch is empty then, epoch ← {all traffic lights}
6. return to (2).

First, note that in (4) we don't necessarily turn i, j to red after $T_i \cdot \frac{f_i}{fe_i}$ seconds but only in the case that there is another traffic light in the current epoch with f > 0. (This is because scheduling is also turning the previous traffic lights to red).

In addition, $T_i \cdot \frac{f_i}{fe_i}$ is the time will take for all cars that arrived to the ith lane to cross the junction.

The time units are depends on the units of f_i, fe_i, and T_i.

Moreover, note that $T_i \cdot \frac{f_i}{fe_i}$ time units, could be a relatively long time. This is because the goal is to maximize the throughput and not to minimize the maximal waiting time (which is a different problem). So, although no lane is starved, a car in a lane can wait for quite a lot of time in theory. In this case, it is possible to limit the scheduled time to a certain constant T_{max} . This constant can be set empirically by the simulation results.

We will now give some intuition and motivation to this algorithm. Think of an imaginary square area around the junction to be scheduled. We want that when a green light is given to a lane all the cars that inhibit this square will pass in that time period where the light is green. This area is called the Ready Area as described in [6]. Therefore in $T_i \cdot f_i/fe_i$ time the number of cars passing the junction is $T_i \cdot f_i$. The number of cars waiting in the lane is exactly the same number. This explains step (4) of the algorithm.

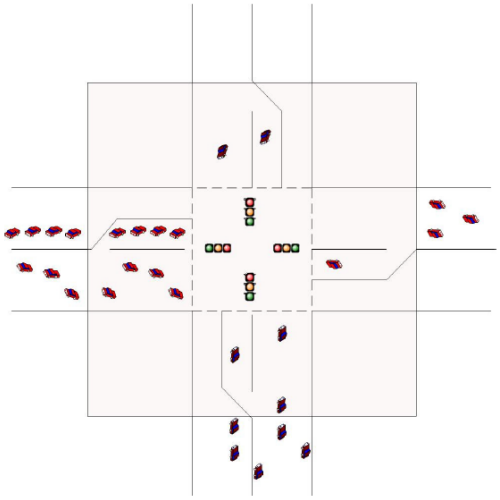


Fig. 2. Ready Area Around the Traffic Light

In addition, if we could set two traffic light to green at the same time we would like to do so in order to increase the traffic throughput. This explains step (3). We find a traffic light j that could be scheduled with traffic light I that was chosen in step (2).

The choice of the traffic light in step (2) is the most occupied lane that wasn't scheduled in the current epoch. When the most occupied lane has occupancy zero, this means the current epoch is over and a new one starts.

The idea of scheduling the traffic lights in epochs is in order to prevent starvation. Unlike the Max Occupancy Scheduler, this algorithm doesn't starve any lane of cars in a junction. This obviously could harm the throughput of traffic in a junction, but a real city cannot let a traffic lane be starved.

We will now describe the implementation of the algorithm in our project. Since the simulation is consist of discrete steps and not fluid time, a time unit is defined as a simulation step. The mapping M is calculated before the simulation- during the initialization of the scheduling class. The epoch variable is a local variable in the scheduler class.

In addition, the calculation of f_i, fe_i is required. SUMO has many detectors (sensors). We used E2 type which can sense how many vehicles are on each lane. This detector use gravity detection technology and can determine the number of cars on it. Therefore, this detector has to be long enough to capture all the cars in the lane. From the input of these detectors one can calculate f_i, fe_i.

The detectors are placed in the map configuration file to SUMO as described in the beginning of this section. The output of the detectors is accessed by the algorithm through the Detector class (which receives its information from TraCI).

This concludes the description of the algorithm and its im/plementation in our project. In addition, this concludes the description of out project's implementation.

IV. EVALUATION

In this section we evaluate the different scheduling algorithms and compare them. We remind that there are three algorithms: the static scheduler, max occupancy scheduler, and the sophisticated algorithm.

We are going to run several simulations on each algorithm and check several parameters, such as, average waiting time, average speed, etc. Then, we will compare the results between the different algorithms.

As described before this is done with the city hall's interface. There is a script that runs the simulation with a given city's map and scheduling algorithm. The script outputs a table comparing the different performances of the algorithm. The statistics is obtained from the XML file SUMO output as a logging file of the simulation.

We tested our algorithms on a simple one junction map, i.e. the entire city is one junction with traffic coming from two opposite sides. This is a simple map that is useful when one want to have a controlled and monitored environment from the simulation.

We will show the results of the evaluation simulation. And analyze the results.

Statistic	tripinfo_realtime_sophisticated.xml	tripinfo_realtime_static.xml	tripinfo_realtime_max_occupancy.xml
Total Time-Loss	14262.119999999995	28935.869999999984	14509.919999999996
Average Time-Loss	18.570468749999993	37.67691406249998	18.893124999999994
Max Time-Loss	81.73	74.76	82.07
Minimum Time-Loss	12.81	13.81	13.45
Total waitSteps	851	9877	989
Average waitSteps	1.1080729166666667	12.860677083333334	1.2877604166666667
Max waitSteps	53	40	53
Minimum waitSteps	0	0	0
Total Duration	72356.0	86970.0	72587.0
Average Duration	94.21354166666667	113.2421875	94.51432291666667
Max Duration	153.0	151.0	154.0
Minimum Duration	88.0	88.0	88.0
Total routeLength	779378.7999999892	779378.7999999892	779378.7999999892
Average routeLength	1014.8161458333193	1014.8161458333193	1014.8161458333193
Max routeLength	1015.1	1015.1	1015.1
Minimum routeLength	1013.1	1013.1	1013.1

The left column contains the different statistics calculated on each simulation. The next three columns represent each one of the three algorithms.

We will now compare the static scheduler with the max occupancy scheduler. Recall that the static scheduler correspond to a regular city and the max occupancy corresponds to the simplest algorithm for a smart city. We can see that the total and average time loss in the smart city is twice as low as in the regular city. However, the maximal time loss is actually higher in the max occupancy scheduler.

One possible explanation to this could be the high overhead in context switches. A context switch is when a traffic light turns red and another turns green. Since the drivers are not perfect, there is an overhead in this action, i.e. there is a time wasted until the first car in the lane notices the change of light and there is a delay in pressing the gas pedal and car acceleration. Therefore, constant context switches could result high overhead and increase in the time loss. Since this is a relatively rare event, just the maximal time loss is greater in the max occupancy scheduler, and the total and average time loss is lower.

We can verify that the route length in each algorithm is the same since the simulation environment is the same. The traffic light scheduling does not change the routes of the vehicles.

Since the static scheduler is a reference to a regular city, we can conclude that the performance of the max occupancy scheduler is an improvement to a regular city. So even the simplest idea to a smart city scheduling could cause a massive improvement.

We now compare the sophisticated algorithm with the max occupancy algorithm. This comparison is important to see whether or not the sophisticated algorithm improves on the simplest smart scheduling.

We can see that the sophisticated algorithm improves the max occupancy scheduler on every parameter. The improvement isn't as massive as previous comparison, but still it is significant.

The sophisticated algorithm also suffers from the same problem as the max occupancy algorithm. The maximum time loss is still higher compared to the static scheduler. The explanation is identical as in the max occupancy scheduler.

V. CONCLUSIONS

Increasing traffic overflow is a major objective in most cities. We introduced the approach of a smart city that takes advantage of recent technological progress in order to accomplish this goal. A smart city manages its traffic-lighted junctions with a scheduling algorithm that receives input from sensors indicating on the traffic on that junction.

Since we don't have a real city to try an implement a smart city we implemented a simulated one. We succeed to simulate traffic in a desired city and get real time statistics during the simulation. This is a desired functionality by the city hall. Moreover we created a way to test our scheduling algorithms, which could also be used by the city hall.

Our testing showed a significant improvement by many parameters including total and average waiting time in red lights. We also showed our sophisticated algorithm is better than a simple algorithm a smart city could apply.

This is a POC to the idea that a smart city could significantly increase traffic flow and reduce waiting time.

There are many interesting future work to be done. From an algorithmic angle, we want to find a clever algorithm that won't schedule each junction independently. A green wave is a powerful technique used to increase traffic flow, even in a static scheduling in current non-smart cities. Moreover, we would like our future scheduler to consider the overhead of context switches of the traffic lights.

From the angle of the city hall's interface we would like to create a server from which the city hall could receive statistics and data in a user friendly way.

From the testing point of view, we would like to test our algorithms on a real city (perhaps a neighborhood in a real city). Currently, we only tested our algorithm on one junction only.

Finally, from the simulation perspective, we would like to create a different simulation environment that will give us more freedom and control over the simulation parameters and inputs.

REFERENCES

[1] Air pollution and health risks due to vehicle traffic. Kai Zhanga and Stuart Battermanb.

[2] Public-health impact of outdoor and traffic-related air pollution: a European assessment. N.Künzli MD et al.

[3] Air pollution in cities. Helmut Mayer

[4] Does traffic congestion reduce employment growth? Kent Hymel

[5] Smart and Digital City: A Systematic Literature Review. Annalisa Cocchia.

[6] M2M Communications for Smart City: An Event-Based Architecture. Jiafu Wan

[7] Top -- k Query Based Dynamic Scheduling for IoT-enabled Smart City Waste Collection. Theodoros Anagnostopoulos et al.

[8] An Intelligent Traffic Light Scheduling Algorithm Through VANETs. Maram Bani Younes, Azzedine Boukerche