
Moscow Institute of Physics and Technology
(national research university)
Phystech School of Applied Mathematics and Informatics
Chair of Discrete Mathematics

Major of study: Applied Mathematics and Informatics (bachelor program)

Specialization: Computer Science/Информатика

(Bachelor)

Student:

Abo Alnaser Mohamad



(student's signature)


Scientific supervisor:

Daynyak Aleksandr Borisovich,
канд. физ.-мат. наук, доц.



(supervisor's signature)

Adviser (if available):



(adviser's signature)

Moscow 2022

Contents

1	Introduction	4
1.1	Overview	4
1.2	Graph theory	5
1.2.1	Definitions in Graph Theory	6
1.3	Crossroads	7
1.4	Traffic flow characteristics	8
2	Main content	9
2.1	Formulation of the problem	10
2.2	Compatibility graph	11
2.3	Software creation	12
2.3.1	Input and output	13
2.3.2	Describing the algorithm	17
2.3.3	Technologies used	24
2.4	Traffic light phasing	25
3	Conclusion	30
3.1	Software	30
3.2	Summary	30

Abstract

One of the most familiar daily problems that people face is traffic jams at traffic lights at crossroads which is a situation in which several vehicles have stopped moving on a road or are moving very slowly.

One of the issues that graph theory can solve is the traffic problem at traffic lights at crossroads. Graph theory is often applied for modeling the traffic flows at crossroads into the compatibility graph where two traffic flows are connected by an edge in the compatibility graph if and only if they can move simultaneously at the crossroad without causing collisions. Modeling the compatibility graph can remarkably minimize the total waiting time.

In this paper, we study the crossroads and how to use them in order to model and split the compatibility graph into cliques where each clique is a set of flows that may simultaneously move at an intersection without causing crashes. Moreover, flows can form a time cycle where each flow is given a duration to flow and may move once within the cycle. We also present our software that accepts the traffic streams as an input, generates all possible cliques and then takes them into the compatibility graph.

Chapter 1

Introduction

1.1 Overview

The continuous increase of population and number of vehicles in towns and cities can lead to various kinds of problems, e.g. traffic accidents, noise pollution and increase in waiting time at traffic lights which can lead to traffic congestion. Therefore, traffic theory was introduced for the sake of helping engineers understand more about the traffic flow and aiming at improving road traffic, and the main problem related to it is traffic congestion.

Traffic congestion is one of the most exhausting issues nowadays due to numerous reasons that cannot be addressed and affects millions of traffic participants everyday. In order to solve such an issue, a few solutions are suggested, e.g. building new roads that may reduce this problem, but this way may need huge budgets. Therefore, instead of building new roads, the second way is to model and install an efficient traffic management system on the existing roads. Optimal traffic management systems can solve many traffic problems without the need of modifying the roads structure.

Traffic management plays a main role in reducing congestion and collisions, therefore we constantly need a traffic system in order to control the traffic flow as the town or city expands. Traffic lights are one of the most important traffic systems that can impact on the traffic flow and improve it.

Traffic lights efficiently helped with solving conflicts between traffic streams at intersections by controlling the traffic stream and deciding when the driver should move or stop driving so that cars from different directions won't cause collisions with each other.

In order to determine whether the traffic flow at crossroads is optimal

and efficient, if there are two or more flows from different directions that can move simultaneously at the same time with maximum total time flow and without causing crashes or congestion, then we can say the traffic management system is effective and efficient and this can lead us to an important branch in mathematics which is graph theory.

Our traffic control problem aims to minimize the total waiting time for traffic participants and graph theory can be applied here in order to make a graph that decreases the total waiting time at traffic lights at crossroads. In this graph, every vertex is a flow and two or more flows are called compatibility and connected by edges if and only if they can flow smoothly simultaneously without any obstacles. By modeling the compatibility graph, we can determine the optimal total waiting time at crossroads.

This thesis paper applies graph theory concepts in traffic control issues at intersections so that we can set the optimal total waiting time for traffic lights at crossroads by modeling the compatibility graph in order to reduce traffic congestion.

1.2 Graph theory

Graph Theory came out in 1735 by Swiss mathematician, L.Euler in the city of Königsberg. The city of Königsberg is located in Europe and consists of two islands, connected to each other and to the mainland by seven bridges. Königsberg bridge problem was whether it were possible to take a walk and cross each bridge exactly once and come back to the original position. In a first demonstration of graph theory, Euler showed that it was not possible.

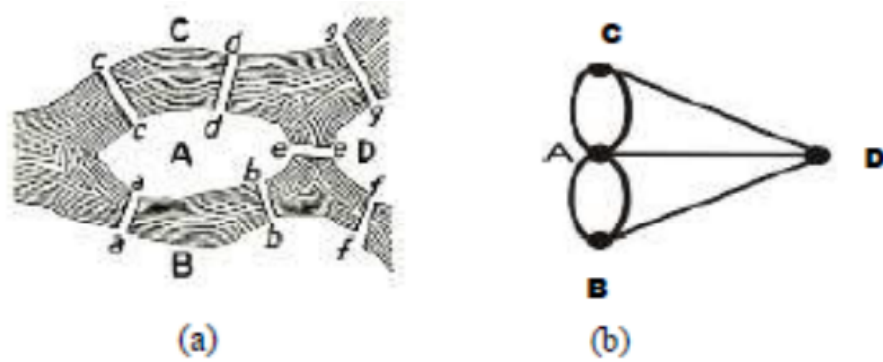


Figure 1.1. (a) Königsberg Bridges, (b) Graph that represents Königsberg Bridges. [1]

1.2.1 Definitions in Graph Theory

Today, graph theory plays an important role in different kinds of real world applications and in various branches of technology and science such as Computer Science, Engineering Science, even in Social Sciences.

Moreover, we can't turn eyes away from graph theory applications in other fields like Biology, Chemistry (for the study of molecules, construction of bonds and for the study of atoms), Electrical engineering (communication networks and coding theory), Computer Science (algorithms and computation) and operations research (scheduling).

- A **graph** $G(V, E)$ is a set of vertices $G(V)$, or nodes, and edges $G(E)$ between some or all of the vertices. When there exists a path that traverses each edge exactly once such that the path begins and ends at the same vertex, the path is known as an Eulerian circuit and the graph is known as an Eulerian graph.
- A **vertex** or node is the fundamental unit of which graphs are formed.
- An **edge** of a graph is the connection between 2 nodes or vertices of the graph.
- **Adjacent vertices** are vertices that are connected by an edge.

- A **directed graph** is a graph where edges have directions. Therefore, they are ordered pairs of vertices.
- An **undirected graph** is a graph where none of the edges have direction. Therefore, they are unordered pairs of vertices.
- A **spanning subgraph** is a graph that contains all the vertices of the supergraph.
- A **clique** is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent.
- A **maximal clique** is a clique that cannot be extended by including one more adjacent vertex.

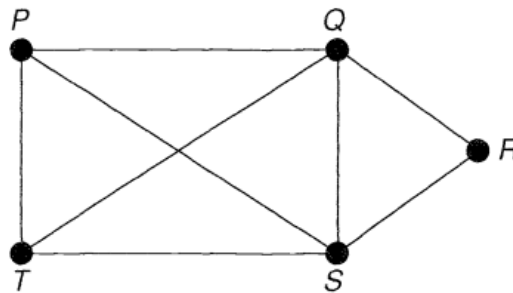


Figure 1.2. Example of a graph [7]

1.3 Crossroads

Road intersections or crossroads represent one of the main and complex traffic systems. They are places where two or more roads intersect or meet. An enormous amount of traffic collisions take place at traffic crossroads. There are several kinds of crossroads as they differ in size, configuration, and design.

The main goal of making effective crossroads is to bring down the number of traffic conflicts between traffic participants and to provide traffic flow with smoothness and comfort.

1.4 Traffic flow characteristics

There are a few characteristics for traffic flow which are necessary to be aware of so that we can study the traffic flow faster and easier. The main characteristics are as follows:

- Traffic flow volume: It's referred to with the number of automobiles that cross a certain point in a certain road within a certain time unit. It's represented by the number of automobiles per hour.
- Traffic flow density: It's referred to with the number of automobiles that are within a certain unit of road length. It's represented by the number of automobiles per km.
- Traffic flows speed: It's referred to with the distance that can be got to within a certain time unit. It's represented by distance per certain time unit.

Chapter 2

Main content

This chapter describes the proposed approach with graph theory to solving the traffic collision problem at crossroad which has given a significant results in optimizing the traffic lights.

In the optimization of traffic lights, the main factor is the total flow time which is the time period given for each traffic stream to flow. Therefore, the solution of this issue is concerned to maximize the total flow time of traffic participants.

In block 2.1, a brief summary of how graph theory can be applied to the traffic light problem at crossroads. A crossroad example is presented and we describe what are the compatible traffic flows and how a graph can be compatible.

Block 2.2 describes how to model the compatibility graph from existing traffic flows of a crossroad by generating all possible maximal cliques where each clique is a set of traffic flow that can move simultaneously and safely.

In block 2.3, a complete description of the web application software that has been made during this work. We explain the input and output of the software and the steps that are followed by the algorithm in order to get the main result which is modeling the compatibility graph of a crossroad intersection. We also list the technologies that were used for implementing the software.

Block 2.4, describes what traffic light phasing is and how it can applied to the compatibility graph. We assign durations for all traffic streams and we propose a mathematical model of the traffic cycle in order to minimize the total waiting time.

2.1 Formulation of the problem

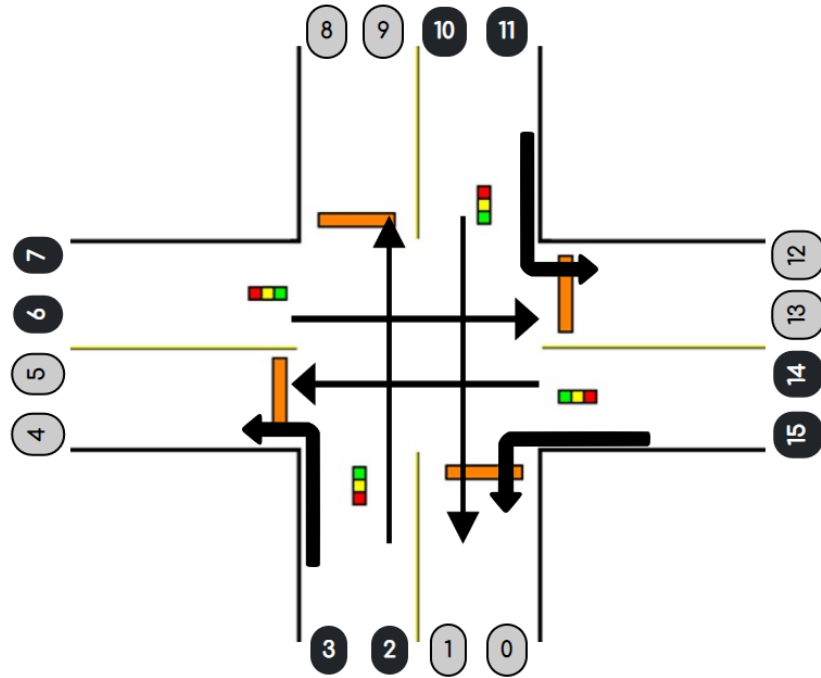


Figure 2.1. Traffic Flow Intersection

The Figure 2.1 above shows a traffic flow intersection with traffic streams $(3,4)$, $(15,0)$, $(14,5)$, $(11,12)$, $(10,1)$, $(6,13)$ and $(2,9)$ whose vertices are a, b, c, d, e, f and g respectively.

Some streams can flow in the intersection simultaneously without threatening each other and for that reason these streams can be referred to by the term **compatible**.

The compatibility model and how a traffic intersection must work is made earlier by traffic engineers and the process can depend on several characteristics of the traffic flow such as traffic pattern and volume of traffic flow.

In order for the traffic engineers to be able to determine how these traffic streams must flow, they have to collect all the information about the traffic intersection compatibility into a graph called the **compatibility graph**.

2.2 Compatibility graph

The compatibility graph can be divided into cliques where each clique is a set of traffic streams that can move at a traffic intersection at the same time without causing collisions.

In order to model the compatibility graph of a traffic crossroad, we must follow these steps:

1. Notice the direction of every traffic stream and where it ends.
2. Label each traffic flow and assign it with a vertex.
3. Determine whether a stream can flow concurrently aside with other streams or it has to stop and follow the traffic light with these streams (As shown in figure 2.1 streams (c) and (e) are not allowed to flow simultaneously together, but on the other hand other streams such as (a) , (b) , (c) , (d) and (f) are allowed to flow concurrently).
4. Assign every traffic stream with a vertex on the graph.
5. Link two traffic streams (vertices) with an edge on the graph if and only if these streams are compatibility which means that they will not cause dangerous conflicts while flowing concurrently.

The compatibility graph of traffic intersection in figure 2.1 can be seen below:

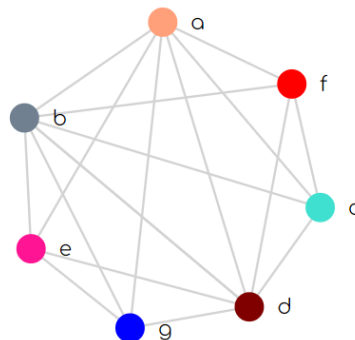


Figure 2.2. compatibility graph of traffic intersection in figure 2.1

As a result, the traffic flows are considered as vertices in the compatibility graph and two flows or vertices are connected by an edge if and only if they are compatibility which means they have the permission to move simultaneously without unpleasant consequences.

We can see the crossroad traffic flow in figure 2.1 above as follows:

- The traffic streams (a) , (b) and (d) can turn left without the need to stop at the traffic light as they don't intersect with other streams.
- The traffic stream (c) has to stop and follow the traffic light as it intersects with the streams (g) and (e) .
- The straight traffic stream (f) intersects with streams (g) and (e) , therefore it must follow the traffic light.
- Traffic stream at index (e) can't flow without following the light as it can cause collisions with the streams (c) and (f) .
- The straight traffic stream (g) intersects with (c) and (f) . As a result, it has to follow the traffic light.

2.3 Software creation

As a part of this work, a web application software was created and developed in order to simulate the process of modeling the compatibility graph from existing streams of a traffic intersection.

The applied algorithm expects an input which is the flow streams that was inserted to the software interface by the user after that, the output, which is explained in details later on, is be displayed on the screen.

We found out that such a software hasn't been implemented before and a decision was made to create and develop it in order to model optimal and

effective traffic intersections, avoid mistakes that may happen while modeling the graph and save the time spent on this process.

2.3.1 Input and output

- What is the input to the algorithm? The input to the algorithm can be defined in an object of key-value pairs where key is a letter used to represent the traffic flow in the compatibility graph as a vertex and value is an array having the start and end points of the traffic flow. The user enters these traffic streams into the traffic intersection in the user interface and determines where a stream must start and end. Moreover, the user has the ability to choose whether to apply a right-hand or left-hand traffic intersection as shown below:

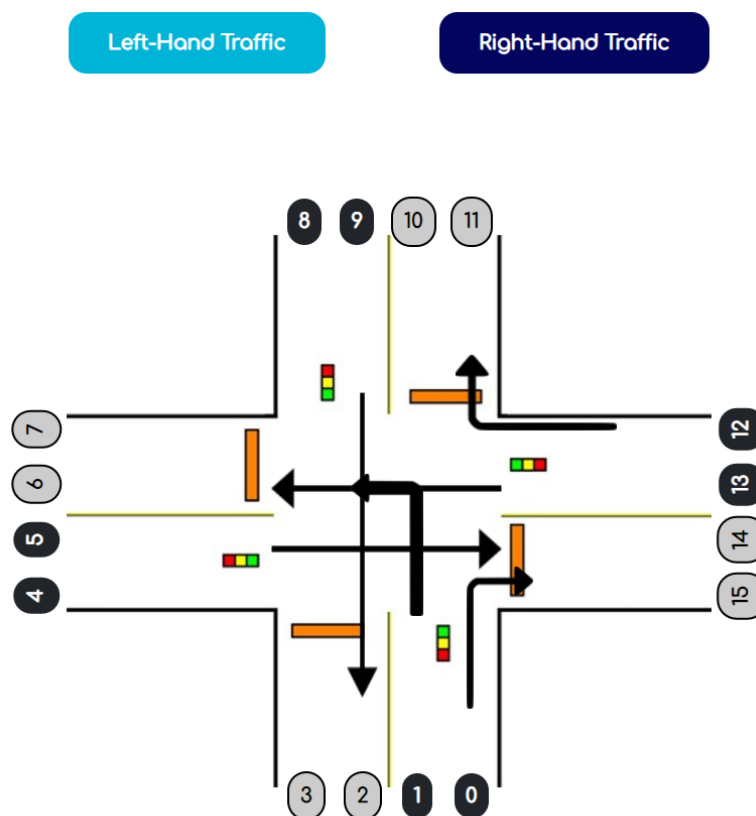


Figure 2.3. Right-hand traffic intersection.

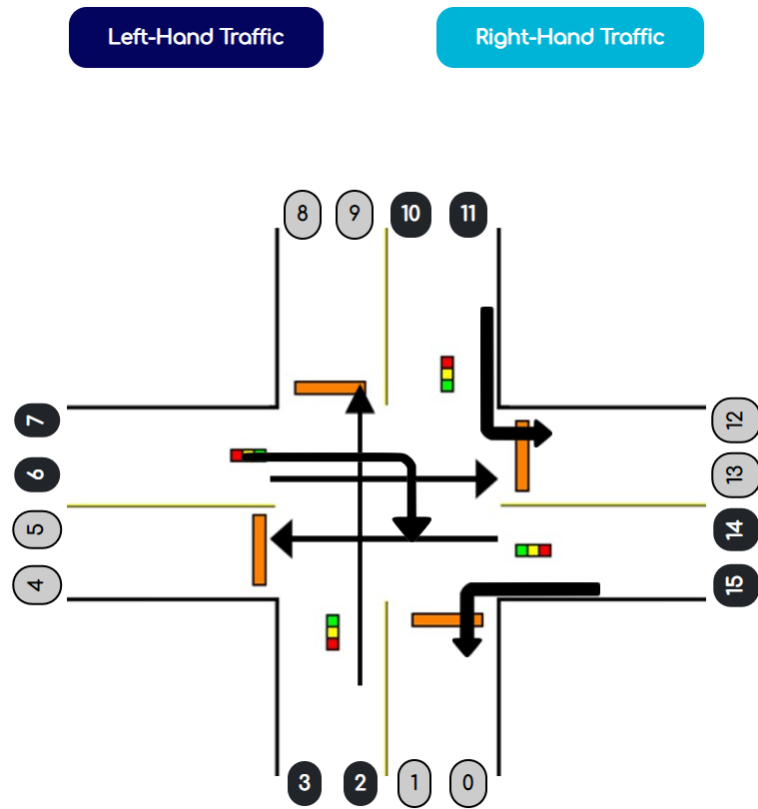


Figure 2.4. Left-hand traffic intersection.

In this section, we considered the right-hand traffic intersection in figure 2.3.

- What is the output to the algorithm? After entering all the traffic streams to the intersection, the algorithm generates the output which consists of three parts:
1. The list of traffic streams that were inserted by the user where every stream is labeled with a unique color and a letter that is used later to represent the traffic stream as a vertex in the compatibility graph.

We can see the traffic streams from the intersection in figure 2.3 below:

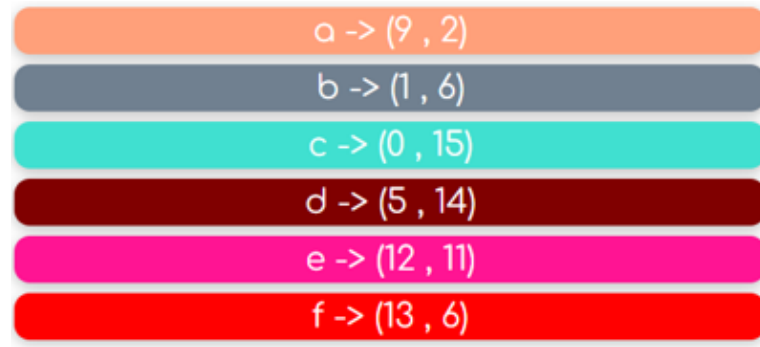


Figure 2.5. The set of traffic flow entered by user from figure 2.3.

The compatibility flows can be seen as following:

- Flow (a) is compatibility with the flows c, e.
 - Flow (b) is compatibility with the flows c, e.
 - Flow (c) is compatibility with the flows d, e, f, a, b.
 - Flow (d) is compatibility with the flows e, f, c.
 - Flow (e) is compatibility with the flows c, d, f, a, b.
 - Flow (f) is compatibility with the flows c, d, e.
2. The list of maximal cliques where each clique is a set of traffic streams that can flow concurrently at the same time without causing dangerous conflicts.

In order to optimize the traffic lights with the compatibility graph, first, we need to split the compatibility graph into maximal cliques where each clique contains of as many compatibility vertices as possible that may simultaneously move at the traffic intersection.

The list of all maximal cliques of the intersection from figure 2.3 is as following:

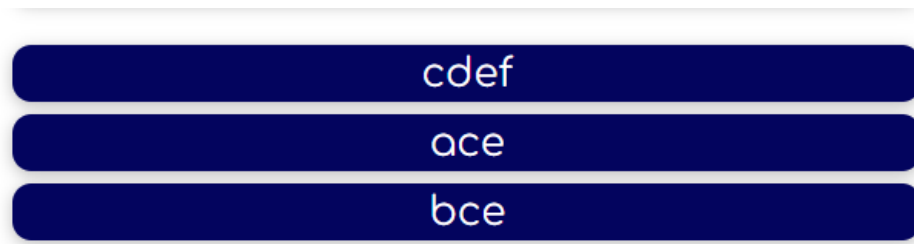


Figure 2.6. The list of maximal cliques from traffic intersection in figure 2.3.

3. The generated compatibility graph of the traffic intersection using the entered traffic streams and the maximal cliques.

The description from the first output from figure 2.5 and the second output from figure 2.6 can help us with generating the compatibility graph by collecting all information we need about the flow streams in order to model the graph which can be seen as follows:

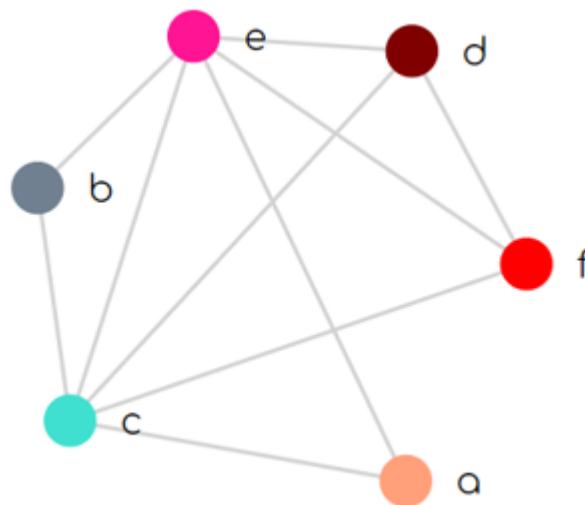


Figure 2.7. The compatibility graph of figure 2.3.

The given compatibility graph which represents the intersection compatibility of the traffic flow crossroad in figure 2.3 can be described as:

- Every two vertices on the graph are connected by an edge if they achieve the compatibility rule which states that adjacent vertices can move concurrently at the same time if and only if they don't intersect with each other at some point and cause crashes when flowing simultaneously, such as c, d, e and f which can be seen in figure 2.8 below.

- The cliques of a compatibility graph are the maximum complete sub-graphs of the compatibility graph.

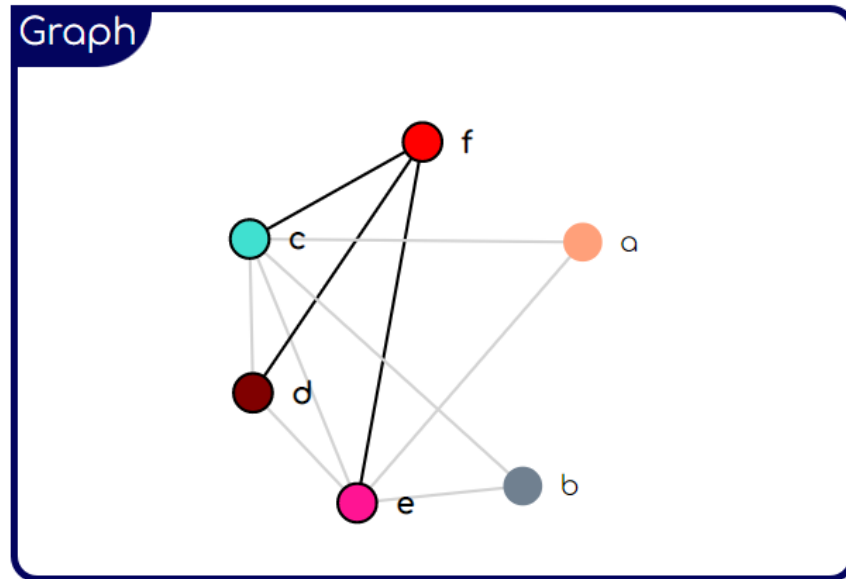


Figure 2.8. Example of a maximum complete subgraph of the compatibility graph of figure 2.3

The user has the ability to hover over any of the graph vertices such as vertex (f) and edges of the adjacent vertices of vertex (f) is displayed in bold. This shows that the vertices f, c, d, and e can flow simultaneously because they are connected with each other by edges.

2.3.2 Describing the algorithm

The software uses our implemented algorithm whose main purpose is to optimize the traffic lights by generating the compatibility graph which can minimize the total waiting time for traffic participants.

The user first inserts the traffic streams to the traffic crossroad interface. After that, a labeled list of traffic streams is displayed in addition to a list of all possible maximal cliques, where each clique is a set of traffic streams that may flow simultaneously at the same time without dangerous consequences, and the generated compatibility graph.

The steps of how the algorithm works can be explained as follows:

1. Pairing traffic flows with different endpoints:

The used crossroad in the software consists of 8 traffic tracks and up to 16 possible traffic streams can flow whether straight streams or turn streams as shown in figure 2.3 above.

At this point, the algorithm takes the user input which is an object of key-value pairs where **key** is a letter used to represent the traffic flow in the compatibility graph as a vertex and **value** is an array having the start and end points of the traffic flow. The algorithm iterates over these values and pairs them into a single array if and only if they have different endpoints. Eventually, all of the generated arrays that contain paired traffic flows with different endpoints are added to a new single array u .

Consider the traffic flow below in figure 2.9 that starts at index (9) and ends at (14), it can't be paired with the traffic flow (5, 14) as they have the same endpoint and it can lead to a collision. On the other hand the flows (5, 14) and (0, 15) have different endpoints and can be paired together as following $[[5, 14], [0, 15]]$.

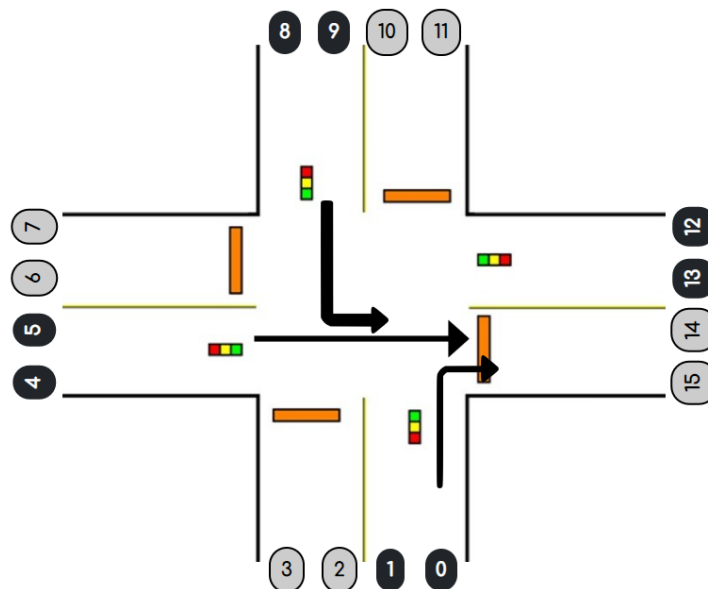


Figure 2.9. Example of traffic flows with the same endpoint.

2. Generating all possibilities of traffic intersections:

As mentioned before, the user has to input the traffic streams into the interface and if two streams intersect, they must not be connected by an edge in the compatibility graph. Therefore, it's important to generate all intersection possibilities in order to prohibit traffic streams or vertices that might intersect from being connected by an edge in the graph.

We predefined nested arrays of straight streams and turn streams such as (9, 2) and (1, 6) respectively. The algorithm uses these nested arrays to generate all possible intersections that might take place on the traffic crossroad by taking the **Cartesian product**, which is the collection of all ordered pairs obtained by the product of the two non-empty sets, between all the traffic flows array.

For instance, consider the two sets $A = \{[1, 6]\}$ and $B = \{[8, 3], [9, 2], [5, 14], [4, 15], [5, 10], [13, 2]\}$ where A is a set containing a traffic stream that starts at index (1) and ends at (6), and B is a set containing all traffic streams that might intersect with the stream in set A at the flow time (see figure 2.10).

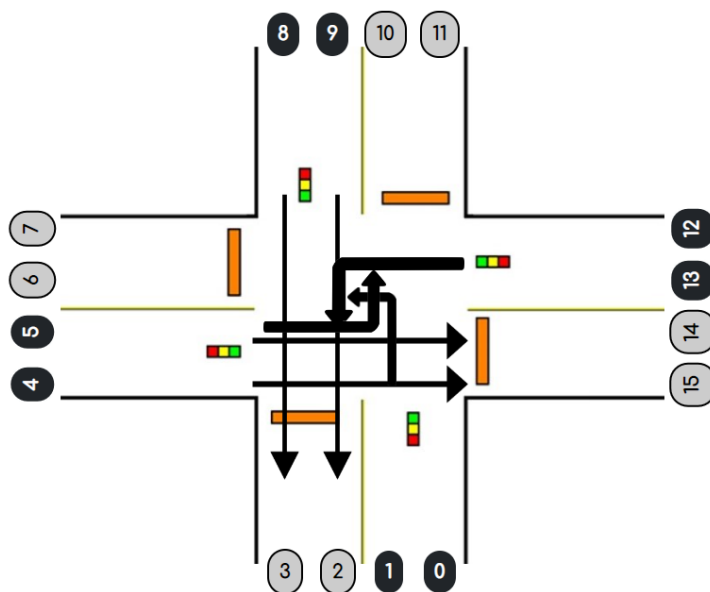


Figure 2.10. Traffic flows that intersect with the flow (1,6) in set A .

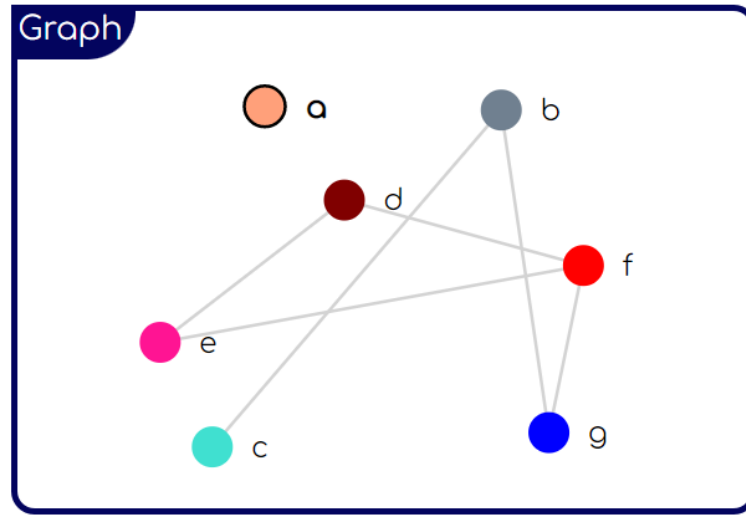


Figure 2.11. Compatibility graph of traffic intersection in figure 2.10.

Then, the Cartesian product of these two sets is the entire collection of all ordered pairs:

$A \times B = \{([1,6],[8,3]), ([1,6],[9,2]), ([1,6],[5,14]), ([1,6],[4,15]), ([1,6],[5,10]), ([1,6],[13,2])\}$. As a result, the traffic stream (1, 6) can't flow concurrently with any of the other traffic streams as it can lead to traffic accidents, therefore the stream (1, 6) can't be connected with the other streams by edges in the compatibility graph and this can be seen in the figure 2.11 above where:

- Vertex $a \rightarrow (1, 6)$
- Vertex $b \rightarrow (8, 3)$
- Vertex $c \rightarrow (9, 2)$
- Vertex $d \rightarrow (5, 14)$
- Vertex $e \rightarrow (4, 15)$
- Vertex $f \rightarrow (5, 10)$
- Vertex $g \rightarrow (13, 2)$

We keep generating all possible intersections and eventually we add all of them into a new single array v .

3. Generating all compatible pairs of traffic flows:

At this point, the algorithm applies the **combinations without repetition** of our traffic flows which gives us all the different groups of traffic flows that can be made by these flows, so that two groups are different if and only if they have different flows.

Consider the traffic flows from figure 2.5, we take all of the these flows entered by the user, make combinations without repetition between them and we append all of the combinations into a new single array z which can be seen as following:

$$z = [$$

$$[[9, 2], [1, 6]], [[9, 2], [0, 15]], [[9, 2], [5, 14]], [[9, 2], [12, 11]],$$

$$[[9, 2], [13, 6]], [[1, 6], [0, 15]], [[1, 6], [5, 14]], [[1, 6], [12, 11]],$$

$$[[1, 6], [13, 6]], [[0, 15], [5, 14]], [[0, 15], [12, 11]], [[0, 15], [13, 6]],$$

$$[[5, 14], [12, 11]], [[5, 14], [13, 6]], [[12, 11], [13, 6]]$$

$$]$$

In order for every combination of flows in array z to be in the compatibility graph, the combination must exist in the array u from the first step which contains all combinations of traffic flows with different endpoints because 2 traffic streams can't be compatible if they have the same endpoint. Moreover, each combination from array z must not exist in the array v from the second step which contains all combinations of flows that intersect with each other, otherwise it can lead to traffic collisions.

Eventually, the algorithm traverses over all the combinations in array z and if a combination i doesn't exists in u or exists in v , then we pop i out from the array z and we keep traversing till we have in array z only combinations of flows which don't have the same endpoint and don't

intersect with each other during the flow time. We can see as follows that the length of array z decreased from 15 to 10 as we popped out 5 combinations:

$$z = [$$

$$[[9, 2], [0, 15]], [[9, 2], [12, 11]], [[1, 6], [0, 15]], [[1, 6], [12, 11]],$$

$$[[0, 15], [5, 14]], [[0, 15], [12, 11]], [[0, 15], [13, 6]],$$

$$[[5, 14], [12, 11]], [[5, 14], [13, 6]], [[12, 11], [13, 6]]$$

$$]$$

4. Generating all maximal cliques:

At this point, we have our array z from the third step which contains all possible combinations of compatible flows and our goal here is to generate all maximal cliques from array z where each clique is a set of compatible traffic streams (see figure 2.6).

The algorithm traverses over the array z and starts from the first combination of flows i and compares it with each of the other combinations of flows j . If a flow in j doesn't exist in i and can be compatible with all flows that exist in i , then we append it to i . We keep traversing and appending till the end of the array z is reached, after that array z will be having all possible cliques of compatible streams which can be seen as follows:

$$z = [$$

$$[[9, 2], [0, 15], [12, 11]], [[9, 2], [12, 11], [0, 15]],$$

$$[[1, 6], [0, 15], [12, 11]], [[1, 6], [12, 11]],$$

$$[[0, 15], [5, 14], [12, 11], [13, 6]], [[0, 15], [12, 11], [13, 6]],$$

$$[[0, 15], [13, 6], [12, 11]], [[5, 14], [12, 11], [13, 6]],$$

$$\begin{aligned} & [[5, 14], [13, 6]], [[12, 11], [13, 6]] \\ &] \end{aligned}$$

Eventually, the array z consists of all possible cliques and in order to be able to model the compatibility graph, we must have only the set of all maximal cliques. Therefore, the algorithm traverses over the array z and if all elements of some clique i exist in another clique, we remove clique i from the array z as it's not a maximal clique. The algorithm keeps traversing and checking till array z will be having only the set of all possible maximal cliques that can be seen below:

$$\begin{aligned} z = [& \\ & [[9, 2], [0, 15], [12, 11]], \\ & [[1, 6], [0, 15], [12, 11]], \\ & [[0, 15], [5, 14], [12, 11], [13, 6]], \\ &] \end{aligned}$$

which is the same as if we define array z as follows:

$$\begin{aligned} z = [& \\ & [a, c, e], \\ & [b, c, e], \\ & [c, d, e, f], \\ &] \end{aligned}$$

We can see that we got the same maximal cliques as shown in figure 2.6 and currently we can use the array z in order to build the compatibility graph.

5. Modeling the compatibility graph:

In order for the graph library which is used in the software to build and configure the compatibility graph of the traffic intersection, we need to pass to it distinct arrays of flows where every array must consists of only 2 flows (vertices).

The algorithm takes the the array z that contains all the maximal cliques where every clique consists of multiple flows (vertices), after that it traverses over each clique i , applies combinations without repetition between the vertices of that clique and every combination of vertices of clique i is pushed to a new array x where every nested array of array x consists of 2 compatible flows that can flow simultaneously without causing collisions. We can see these compatible flows of array x as follows:

$$x = [$$

$$[a, c], [a, e], [c, e], [b, c],$$

$$[b, e], [c, e], [c, d], [c, e],$$

$$[c, f], [d, e], [d, f], [e, f],$$

$$].$$

2.3.3 Technologies used

In the process of creating and developing the algorithm and software, many technologies were involved such as:

- **JavaScript** which is a programming language and it was used to write the algorithm.
- **React JS** which is a frontend JavaScript library for building user interfaces and we used it for building the software user interface.

- **react-d3-graph** which is a library for modeling interactive and configurable graphs with React and we used this library for building the compatibility graph.
- **HTML5** and **CSS3** were used alongside JavaScript and React for building the structure and layout of the web page.

2.4 Traffic light phasing

Traffic light phasing plays an important role in making the traffic movement safe and at the same time efficient. Efficient traffic lights usually refers to less waiting time which is the total red light time in a cycle, less traffic jams or increase of the flow volume.

In order to move further with traffic light phasing, we have to understand what the word "phasing" mean.

- Phase and phase set:

When talking about phasing, a little confusion may occur since traffic engineers call it differently from what traffic technicians call it. When technicians speak about phasing, they usually refer to the individual movements of traffic participants where every participant is a **phase**, whereas the traffic engineers refer to phasing as a combination of multiple phases where every combination is a **phase set**.

In this section, we considered phase as an individual traffic stream and phase set as a clique of compatibility graph which is a set of traffic streams.

An illustration of the phase and phase set can be seen in figure 2.12.

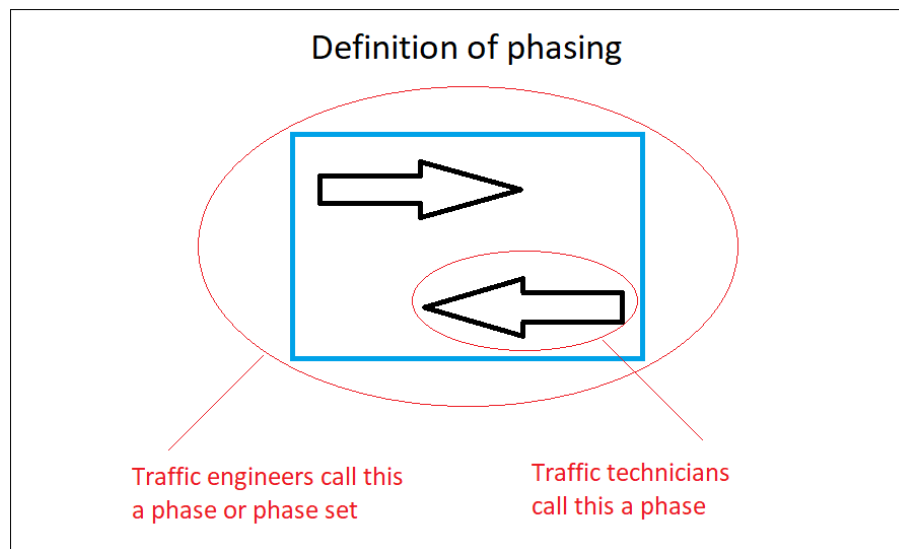


Figure 2.12. Definition of phasing.

- Traffic light cycle:

The **traffic light cycle** can be defined as the one complete set of all traffic light phase sets. In figure 2.13 below, we can see an illustration of a traffic light cycle that consists of 6 phases and 3 phase sets:

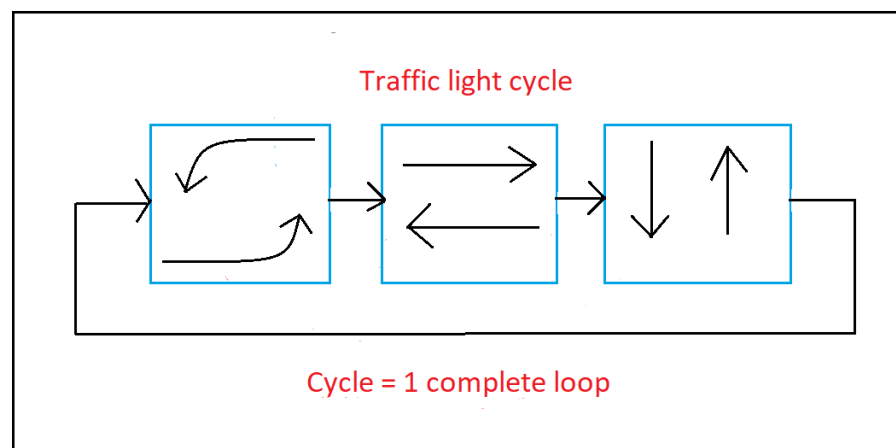


Figure 2.13. Example of a traffic light cycle.

Each arrows box in figure 2.13 is a phase set, whereas each arrow is an individual phase.

The time a traffic light takes to complete one full cycle is called the **cycle length**. For instance, if every phase set from the figure above occupies 20 seconds, then the traffic light cycle length would be equal to 60 seconds.

As we already know, the graph is compatible if two traffic streams can flow simultaneously without causing crashes and the purpose of modeling such a graph for traffic crossroads is to organize the movement of traffic participants and to avoid possible collisions between them and also to minimize the total waiting time.

Each phase (traffic flow) is given a period of time to flow and in order to do that, let's consider the following example in figure 2.14:

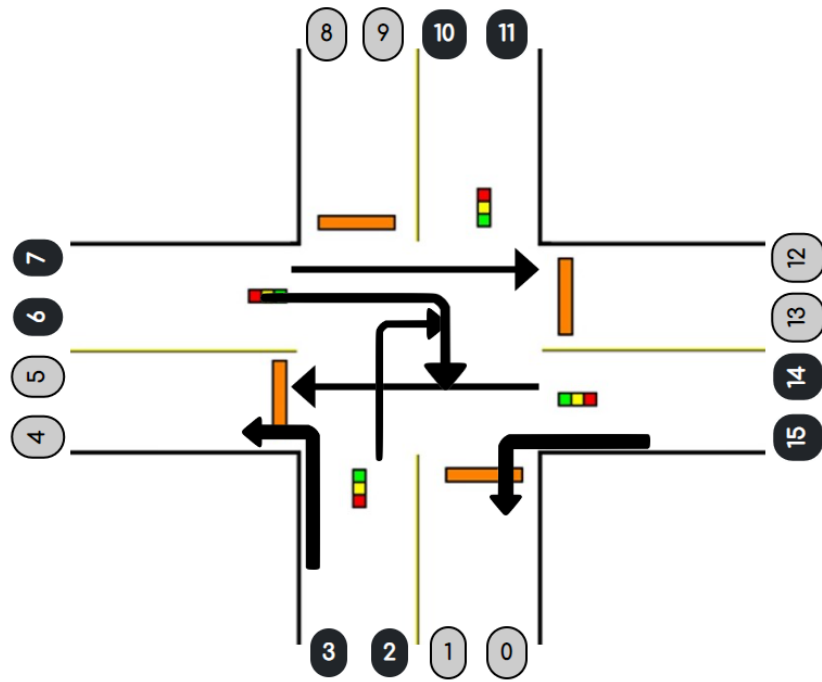


Figure 2.14. Example of a traffic light cycle.

- Flow a is compatible with b, c, d, e, f
- Flow b is compatible with a, e, d
- Flow c is compatible with a, e, d
- Flow d is compatible with a, b, c, e, f
- Flow e is compatible with a, b, c, d, f
- Flow f is compatible with a, d, e

According to the traffic intersection in figure 2.14 and its compatible flow, we can represent the compatibility graph:

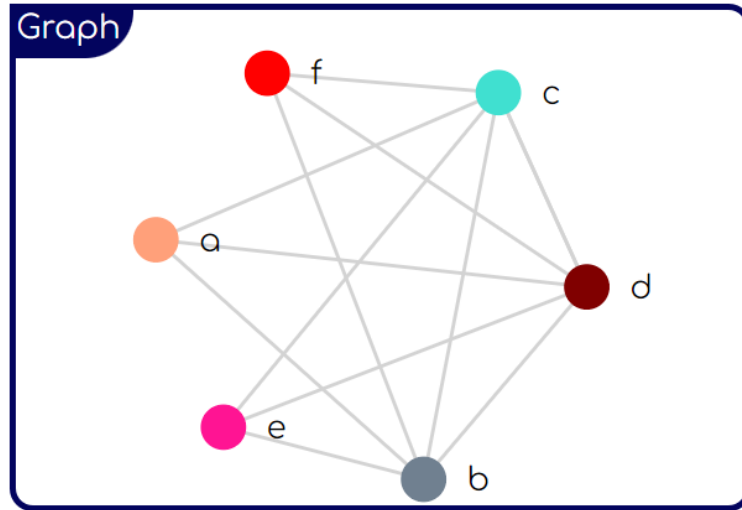


Figure 2.15. Example of a traffic light cycle.

From our compatibility graph we can say that it has 3 phase sets (cliques): $K_1 = \{a, b, d, e\}$, $K_2 = \{a, c, d, e\}$, $K_3 = \{a, d, e, f\}$. In phase set 1, the traffic flows a, b, d, e has the green light to flow, then, in phase set 2, traffic flows a, c, d, e has the green light and so on.

Suppose we assign a duration d_i to each clique K_i .

- $K_1 = \{a, b, d, e\} \rightarrow d_1$
- $K_2 = \{a, c, d, e\} \rightarrow d_2$
- $K_3 = \{a, d, e, f\} \rightarrow d_3$

Now, our goal is to find the values of these d_i 's so the total waiting time is as minimum as possible. The answer is obtained by observing the following: b has the red light during these phase sets K_2 and K_3 , so b 's total waiting time is $d_2 + d_3$. The same goes with the other flows where a 's, c 's, d 's, e 's and f 's total waiting time is 0, $d_1 + d_3$, 0, 0 and $d_1 + d_2$ respectively. Since b 's total waiting time is $d_2 + d_3$, so b 's total flow time is d_1 and so on.

So, we have to minimize Z which is the total waiting time for all traffic flows:

$$Z = 0 + (d_2 + d_3) + (d_1 + d_3) + 0 + 0 + (d_1 + d_2) = 2d_1 + 2d_2 + 2d_3$$

Let's suppose that the minimum flow time for a traffic flow is 20 seconds, then the total cycle is 120 seconds since we have 6 traffic flows.

So, the problem is to reduce Z subject to

$$d_2 + d_3 \geq 20$$

$$d_1 + d_3 \geq 20$$

$$d_1 + d_2 \geq 20$$

The example above is an illustration that ignores the traffic flow volume which is the number of automobiles that cross a certain point in a certain road within a certain time unit and the time durations that are given to phases are created equally.

Conclusion

3.1 Software

A web application software has been developed as a part of this work. As what we explained earlier, the software algorithm accepts the traffic flows of a traffic intersection as an input and generates the compatibility graph of that intersection as an output.

The source code is available at <https://github.com/LordHomie/Graduation-thesis-project-mipt-2022>.

3.2 Summary

During the study, it was found that graph theory can be applied to the traffic light problem at crossroads in order to organize the movement of traffic participants and to avoid possible traffic accidents and also to minimize the total waiting time with the help of traffic lights which are used for solving conflicts between traffic streams at intersections by controlling the traffic streams. We first considered a traffic crossroad with its traffic streams, after that we split the crossroad into maximal cliques of compatible traffic streams and we were able to get the optimal compatibility graph of that crossroad. Moreover, we assigned a duration for each traffic stream and eventually we made a mathematical model of the traffic cycle in order to minimize the total waiting time.

In addition, an algorithm was developed for this traffic light problem which can save time for modeling the compatibility graph. It takes the user input which is a set of traffic streams, after that it generates all possible maximal cliques and eventually displays the configured compatibility graph.

The algorithm showed efficiency in modeling the compatibility graph so

that it only connects traffic streams (vertices) by an edge if and only if they can flow concurrently without causing collisions. However, we're planning to develop the algorithm so that it can also assign a flow time for each traffic stream and try to minimize the total waiting time depending on the volume of each stream.

References list

1. Application of graph theory concept for traffic light control at crossroad, Ekky Kurnia Setiawan, I. Ketut Budayas (2017)
2. Traffic Light Control at Crossroad using Graph theory with Matlab, Dr. Sweta Srivastav (2020)
3. Optimal Feasible Green Light Assignment to a Traffic Intersection using Intersection Graph, A. Bharali, A. K. Baruah (2013)
4. Traffic Control Using Graph Theory, Th. Riedel, U. Brunner (1994)
5. Signal Groups of compatibility Graph in Traffic Control Problems, Arun Kumar Baruah, Niky Baruah (2012)
6. Introduction to Graph Theory Fourth edition, Robin J. Wilson (1996)
7. Graduate Texts in Mathematics, J.A. Bondy, U.S.R. Murty (2008)
8. Application Of Graph Theory in Representing and Modelling Traffic Control Problems, Shakera Tanveer (2016)
9. Writing a report on experiments with algorithms, Herman Haverkort (2013)
10. Research on Traffic Light Adjustment Based on Compatibility Graph of Traffic Flow, Qiang Wang, Lei Wang, Guangcun Wei (2011)
11. Traffic Control Problems using Graph Connectivity, Arun Kumar Baruah (2014)
12. Application of Graph Theory in Traffic Management, Darshankumar Dave, Nityangini Jhala (2014)

13. Compatibility Graphs on Traffic Lights Waiting Time Optimization, Anie Lusiani¹, Euis Sartika, Agus Binarto, Endang Habinuddin (2020)
14. Optimal Traffic Control Urban Intersection, Slobodan Guberinic, Gordana Senborn, Bratislav Lazic (2008)
15. Introduction to Traffic Signal Phasing, Jeffrey W. Buckholz, Ph.D., P.E., PTOE