Small or medium-scale focused research project (STREP)

**ICT Call 8**

FP7-ICT-2011-8

# Cooperative Self-Organizing System for low Carbon Mobility at low Penetration Rates


## COLOMBO: Deliverable 2.3

## Performance of the Traffic Light Control System for different Penetration Rates

| Document Information | |
|---|---|
| Title | Deliverable 2.3 - Performance of the Traffic Light Control System for different Penetration Rates |
| Dissemination Level | PU (Public) |
| Version | 1.2 |
| Date | 10.11.2014 |
| Status | Final version |

| | |
|---|---|
| Authors | Michela Milano (UNIBO), Alessio Bonfietti (UNIBO), Riccardo Belletti (UNIBO), Daniel Krajzewicz (DLR), Thomas Stützle (ULB), Jérémie Dubois-Lacoste (ULB) |

**Table of contents**

# 1 Introduction

The COLOMBO project will deliver a set of modern cooperative traffic surveillance and control applications that target at different transport related objectives such as increasing mobility, resource efficiency, and environmental friendliness.

This deliverable contains the results of Task 2.4 on "Evaluation of Traffic Light Control System for different Penetration Rates" of the project's Work Package (WP) 2. The aim of the task is to test the developed traffic light algorithm on the SUMO simulator on a set of abstract scenarios and evaluate the impact of different penetration rates of equipped vehicles on it. The performed tests gave feedback on the policy model and on the selection algorithm, providing UNIBO with new modelling guidelines. Evaluation was done after automatic offline configuration of the selection algorithm, performed by ULB.

Further evaluations done after publishing [COLOMBO D2.2, 2014] (D2.2) provided us new data that helped us to improve the policy selection algorithm presented in that deliverable. We found out that non-homogeneous traffic conditions should be handled with specific policies and our system was unable to properly detect this kind of situations. This led to the definition of a second version of the policy selection algorithm, improving its flexibility. We decided to include this extension into the deliverable.

The system presented in D2.2 was tested with full knowledge of the simulated vehicles in mind. This simplification helped in modelling but it is not what the COLOMBO project assumes. COLOMBO wants to prove that valuable information about the traffic state can be obtained even when assuming low penetration rates of equipped vehicles. What we mean with *low penetration rates* is that our system will be able to detect only a small percentage of the vehicles approaching the controlled intersection (e.g. 1 % to 5 %). Some algorithms of the system presented in D2.2 rely on counting vehicles, what can be problematic if not all vehicles are sensed.one does This deliverable presents the different approaches we designed to adapt the previous system for low penetration rates of equipped vehicles.

This deliverable is divided into three parts. The first one presents the extensions to the traffic light logic with respect to the version described in the previous deliverable (Deliverable 2.2) and the adaptations developed to cope with the low penetration rates challenge. Then, the second part presents the scenarios used for the experiments proposed in this deliverable. The last part presents the automatic offline configuration and the evaluation of the system along with the obtained results. The results are quite interesting since they show that our proposal is capable to maintain the same performance regardless the penetration rate of equipped vehicles. Moreover, it is comparable to a fully actuated traffic light control system, which always detects 100 % of the vehicles.

## 1.1 Document Objectives

The aim of this document is to show how the traffic control system proposed by COLOMBO in the previous Deliverable 2.2 has been extended and how it reacts without full knowledge of the vehicles approaching the controlled intersection. We want to present all the ideas we had and how the system evolved from its previous state. The first part of the document is focused on describing these extensions.

The document proceeds by describing the scenarios used for testing the implemented extensions and the overall system. The information provided is about the structure of the road networks and about the simulated traffic flows.

Then the description of the automatic parameter tuning tools is given. This type of offline configuration is used to determine the values of the several parameters used by this system.

The document ends with the presentation of the obtained results and parameters, along with some considerations.

## 1.2 Document Structure

Chapters 2 and 3 make the first part of this deliverable and they describe all the extensions and adaptations developed for Task 2.4.

Chapter 2 presents the extensions to the policy selection algorithm needed to cope with non-homogeneous traffic and to improve the flexibility of the system.

Chapter 3 focuses on the problem of the low penetration rates of equipped vehicles and presents the adaptations proposed to cope with it.

Chapter 4 constitutes the second part of the deliverable and presents the synthetic scenarios we used to evaluate the proposed system. Most of the tests are based of scenarios inspired by the German “Richtlinie für Lichtsignalanlagen” (guidance for traffic light systems), or RiLSA.

Chapters 5 and 6 form the last part of the deliverable.

Chapter 5 describes the offline automatic parameter tuning process. This procedure is necessary to achieve good performance given the large number of parameters the system relies on and the complexity of the controlled system.

Chapter 6 presents the results of the tests and the evaluation of the proposed system and its extensions for different penetration rates of equipped vehicles.

The deliverable ends with Chapters 7 and 8, illustrating the future work and improvements that should be done.

# 2 Algorithm Extensions

This section presents the extensions to the system described in [COLOMBO D2.2, 2014]. We begin with a summary of what we described in [COLOMBO D2.2, 2014] along with the currently implemented low-level policies. We then present the second and third version of the policy selection algorithm to improve its performance.

## 2.1 Summary of the initial Policy Selection Algorithm

The main effort of the COLOMBO project is dedicated towards making self-organizing traffic lights an effective means for practical traffic light systems. The common underlying principle of the policies developed for the COLOMBO project is that each traffic light controller, controlling one or more interconnected junctions, operates independently of all other controllers and gets information only on the traffic flow on its incoming and outgoing lanes.

Both a macroscopic and different microscopic level control methods are developed within COLOMBO. Each microscopic level policy performs best under different traffic conditions and the goal of the macroscopic level policy is to select which microscopic level policy should be executed in the controlled intersection for the current traffic conditions.

Microscopic level policies focus primarily on the duration of the current green stage. Each low-level policy differs from the others mainly for the condition used to adjust the green time. It is worth mentioning that the COLOMBO project discards the traditional traffic light cycle. Instead, it proposes the concept of *chains*, sequences of stages starting with green for a specific set of lanes and ending with an *All Red* stage[1]. Each stage can be categorized on the basis of its duration:

- *Decisional*: the microscopic level policy checks whether the current stage should be persisted or not, allowing the rest of the chain to be executed. This decision is taken at every simulation step (one second in SUMO);
- *Transient*: the controller lets the current phase execute until the predefined duration has elapsed.

We also use the terms *target stage* and *commit stage* to identify respectively the green stage that starts a chain and the All Red stage that ends it.

Chapter 3 explains in detail the chain selection algorithm since it is highly affected by the fact that we face in COLOMBO the issue of low penetration rates of equipped vehicles.

The COLOMBO project proposes four microscopic level policies, discussed individually in the following sub-sections.

### 2.1.1 Phase Policy

This policy terminates the current chain as soon as another one has reached the traffic threshold after the minimum duration constraint of the current decisional stage is satisfied. This policy was designed to handle medium-low traffic situations, where this early termination would not make the traffic lights switch too often. It is worth mentioning that *Phase* never ends the current decisional stage if there are no cars opposing the currently allowed traffic flow.

### 2.1.2 Platoon Policy

This policy tries to let all the vehicles in the currently green lanes pass the intersection before releasing the green light. The policy ends the current decisional stage only if:

[1] Red light to all the incoming lanes of the controlled intersection., see D2.1 section 2.2 “Glossary”.

- its minimum duration has elapsed AND
- another chain is above the traffic threshold AND
- there are no other vehicles in the current chain lanes OR the maximum duration has elapsed

It is worth noting that even the *Platoon* policy will not switch chain unless another one requests the green light.

The maximum phase duration is taken into account in order to pre-empt the current chain execution even if there are approaching vehicles. In intense traffic situations, each decisional phase will be executed for the maximum allowed time. The definition of the maximum allowed time for a phase greatly impacts the performance of the system.

### 2.1.3 Marching Policy

This policy is adequate when the traffic looks too intense from all directions to take any online decision regarding the input lanes. In this case, there are two possible approaches:

- Falling back to a static duration for each decisional stages;
- taking into account the saturation of the outgoing lanes so to prevent traffic towards already heavily loaded lanes.

The approach we chose is the first one because we want the policies to implement simple rules.

### 2.1.4 Congestion Policy

This policy is used when the output lanes are congested and there may be vehicles waiting in front of the intersection. To avoid gridlocks, all input lanes are inhibited, i.e. the system terminates the current chain executing each decisional stage for their minimum duration time. When the commit stage is reached, no other chain is activated until the congestion has been solved.

### 2.1.5 Pheromone Levels and Microscopic Policy Selection

Given the uncertainty in the definition and measurement of “high” vs “low” traffic, we use the natural metaphor of *pheromone* to abstract the traffic conditions. Pheromone on both incoming and outgoing lanes is calculated as follows:

$$\varphi(t) = \beta\varphi(t-1) + \gamma(v_{max} - v_{mean}) \tag{2-1}$$

where $\beta$ and $\gamma$ are the evaporation and accumulation coefficients, $v_{max}$ is the maximum allowed speed on the selected lane and $v_{mean}$ is the average speed of the vehicles on that lane.

Pheromone on the incoming and outgoing lanes is averaged and serves as input for the stimulus function. This is a function that maps the low-level policies in the $\varphi_{in} \times \varphi_{out}$ space. The chosen type of function is a Gaussian centered where the policy performs best. The generic stimulus function is:

$$s_{i,j}(\varphi_{in}, \varphi_{out}) = \frac{1}{V} e^{-\frac{(\varphi_{in}-\mu_{in})^2}{2\sigma_{in}^2} - \frac{(\varphi_{out}-\mu_{out})^2}{2\sigma_{out}^2}} \tag{2-2}$$

with

$$V = \iint_{[0,\varphi_{max}]\times[0,\varphi_{max}]} e^{-\frac{(\varphi_{in}-\mu_{in})^2}{2\sigma_{in}^2} - \frac{(\varphi_{out}-\mu_{out})^2}{2\sigma_{out}^2}} d\varphi_{in} d\varphi_{out} \tag{2-3}$$

### 2.1.6 Swarm Policy and Stimulus Function Parameters

We now present again the list of the parameters used by the *Swarm* algorithm and by the stimulus functions.

**Table 2.1: Parameters for the *Swarm* macroscopic level policy**

| Parameter | Symbol | Meaning |
|---|---|---|
| `pheroMaxVal` | $\varphi_{max}$ | Maximum value of the pheromone level, both for input and output lanes |
| `betaNo` | $\beta_{in}$ | Pheromone evaporation coefficient for input lanes |
| `gammaNo` | $\gamma_{in}$ | Pheromone accumulation coefficient for input lanes, to be multiplied by the vehicle count. |
| `betaSp` | $\beta_{out}$ | Pheromone evaporation coefficient for output lanes. |
| `gammaSp` | $\gamma_{out}$ | Pheromone accumulation coefficient for output lanes, to be multiplied by the vehicle count. |
| `pChange` | $p_{change}$ | Probability to select a new policy each time the algorithm is invoked. |
| `thetaMax` | $\theta_{max}$ | Maximum threshold value. |
| `thetaMin` | $\theta_{min}$ | Minimum threshold value. |
| `thetaInit` | | Initial threshold value. |
| `learningCox` | | Threshold update learning coefficient. |
| `forgettingCox` | | Threshold update forgetting coefficient. |
| `threshold` | | Value of the traffic threshold. |
| `maxCongestionDuration` | | Maximum duration in seconds of an all red stage decided by the *Congestion* policy before a reset of the pheromone levels is performed. |

**Table 2.2: Parameters for the stimulus function associated to a microscopic level policy**

| Parameter | Symbol | Meaning |
|---|---|---|
| `StimCox` | $\frac{1}{V}$ | Global scaling factor. |
| `StimCoxExpIn` | | 0 if the $\varphi_{in}$ should not be considered in the stimulus function, 1 otherwise. |
| `StimOffsetIn` | $\mu_{in}$ | Offset relative to the pheromone level of input lanes, moves the Gaussian along the $\varphi_{in}$ axis. |
| `StimDivisorIn` | $2\sigma_{in}^2$ | Scales the width of the Gaussian along the $\varphi_{in}$ axis. |
| `StimCoxExpOut` | | 0 if the $\varphi_{out}$ should not be considered in the stimulus function, 1 otherwise |
| `StimOffsetOut` | $\mu_{out}$ | Offset relative to the pheromone level of output lanes, moves the Gaussian along the $\varphi_{out}$ axis. |

## 2.2 Non-homogeneous Traffic

It is important to highlight that the stimulus function treats averaged pheromone values. Hence, the system does not take into account the dispersion of the pheromone among the approaching lanes. For example, the *Phase* policy should perform best when a "dominant" flow and a minor one on the opposite direction characterize the traffic conditions. *Marching* policy, instead, should be more

adequate when traffic loads are homogeneous. This hypothesis was verified by comparing each low-level policy in a set of tests based on non-homogeneous traffic flow. The results (see subsection 6.1) proved the necessity to detect the dispersion of the pheromone among the incoming lanes. Therefore, the stimulus function should now take into account a third input value related to the dispersion of the pheromone on the incoming lanes, which is calculated as follows:

$$\varphi_{dsp\ in} = \varphi_{in\ max} - \frac{\sum_{i=1, i\neq max}^{n} \varphi_{in\ i}}{n-1} \qquad (2\text{-}4)$$

where $\varphi_{in\ max}$ is the highest pheromone value on the input lanes, $\varphi_{in\ i}$ is the pheromone value on the $i$-th incoming lane and $n$ is the number of incoming lanes.

The new stimulus function is:

$$s_{i,j}\left(\varphi_{in}, \varphi_{out}, \varphi_{dsp\ in}\right) = \frac{1}{V} e^{-\frac{(\varphi_{in}-\mu_{in})^2}{2\sigma_{in}^2} - \frac{(\varphi_{out}-\mu_{out})^2}{2\sigma_{out}^2} - \frac{\left(\varphi_{dsp\ in}-\mu_{dsp\ in}\right)^2}{2\sigma_{dsp\ in}^2}} \qquad (2\text{-}5)$$

The introduction of the new pheromone level increases the list of parameters associated to every low-level policy. Table 2.3 presents the new parameters.

**Table 2.3: Stimulus function new parameters**

| Parameter | Symbol | Meaning |
|---|---|---|
| `StimCoxExpDispersionIn` | | 0 if the $\varphi_{dsp\ in}$ should not be considered in the stimulus function, 1 otherwise. |
| `StimOffsetDispersionIn` | $\mu_{dsp\ in}$ | Offset relative to the pheromone dispersion on the input lanes, moves the Gaussian along the $\varphi_{dsp\ in}$ axis. |
| `StimDivisorDispersionIn` | $2\sigma_{dsp\ in}^2$ | Scales the width of the Gaussian along the $\varphi_{dsp\ in}$ axis. |

The addition of the third input defines what is called as *Swarm version 2*, which is the second version of the policy selection algorithm.

# 3 Algorithm Adaptation for Low Penetration Rates

Every chain measures the traffic associated to its *target lanes*[2]. The chain selection algorithm makes use of this indicator to decide which chain should be executed. *Platoon* and *Phase* low-level policies also use this measure to determine the duration of decisional stages by applying a threshold (the so-called *traffic threshold*) over it. The current implementation of this indicator is called CTS (*Cars-Times-Seconds*) and it is described in subsection 3.1. Since it relies on sensed cars, we wanted to develop and evaluate two new different approaches that could cope with the incomplete knowledge problem the system will face with Low Penetration Rates. The first approach, called eCTS (*estimated Cars-Times-Seconds*) is a variation of the current approach in which the cars are not sensed but estimated based on other measures that can be perceived. This approach is described in subsection 3.2. The last proposed approach tries to estimate the length of the queues of vehicles waiting for the green light. The chosen name is *Queue length* and it will be described in subsection 3.3.

Still, the system requires to sense the vehicles approaching the controlled intersection. The COLOMBO proposal must be capable to adapt to low penetration rates scenarios, which means few to almost none vehicles are detectable. When the system fails to sense cars, the traffic measures associated to every chain are not updated and the traffic threshold applied by *Platoon* and *Phase* may never be reached. This event will result in a never-ending green for one chain, preventing any kind of chain switch. We devised an additional dynamic traffic threshold based upon exponential decay, which will be presented in subsection 3.4.

## 3.1 Sensed Cars-Times-Seconds

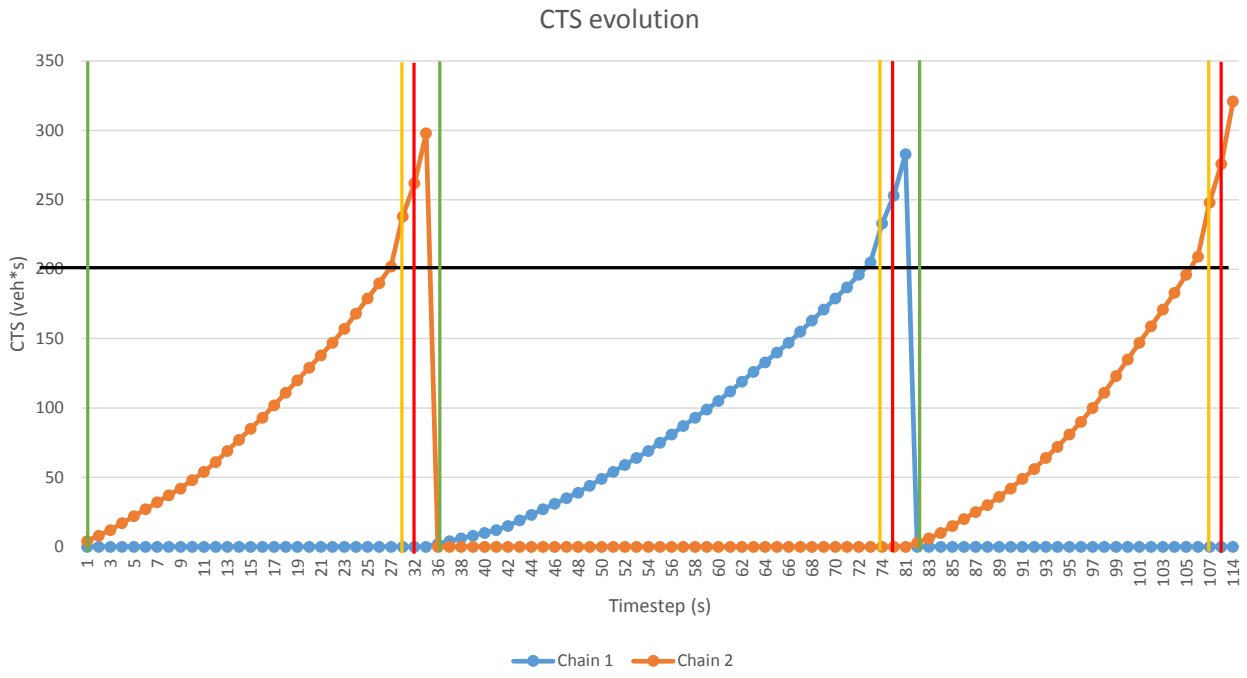


**Figure 3.1: Example of the evolution of CTS values of two chains with Phase Policy activated. Vertical green, yellow and red lines indicate the change of the lights controlling the current chain target lanes and the black bold line is the traffic threshold. Chain 1 is the first one being executed.**

[2] The *target lanes* are the lanes approaching the intersection that chain *c* will grant green to.

The current approach for measuring traffic associated to a chain is called CTS (*Cars-Times-Seconds*). Every chain has a counter associated to it that is reset to 0 after the related chain is executed. This counter is updated according to the following equation:

$$CTS_c(t_i) = CTS_c(t_{i-1}) + \#cars_{c\_target\_lanes}(t) \quad (3\text{-}1)$$

where $CTS_c(t)$ is the new value of the *CTS* counter for chain *c*, $CTS_c(t_{i-1})$ is the value of the counter at the previous time-step and $\#cars_{c\_target\_lanes}(t)$ is the number of cars on the *target lanes* of chain *c*.

When a commit stage is completed, the chain whose counter has the highest value is selected deterministically.

## 3.2 Estimated Cars-Times-Seconds

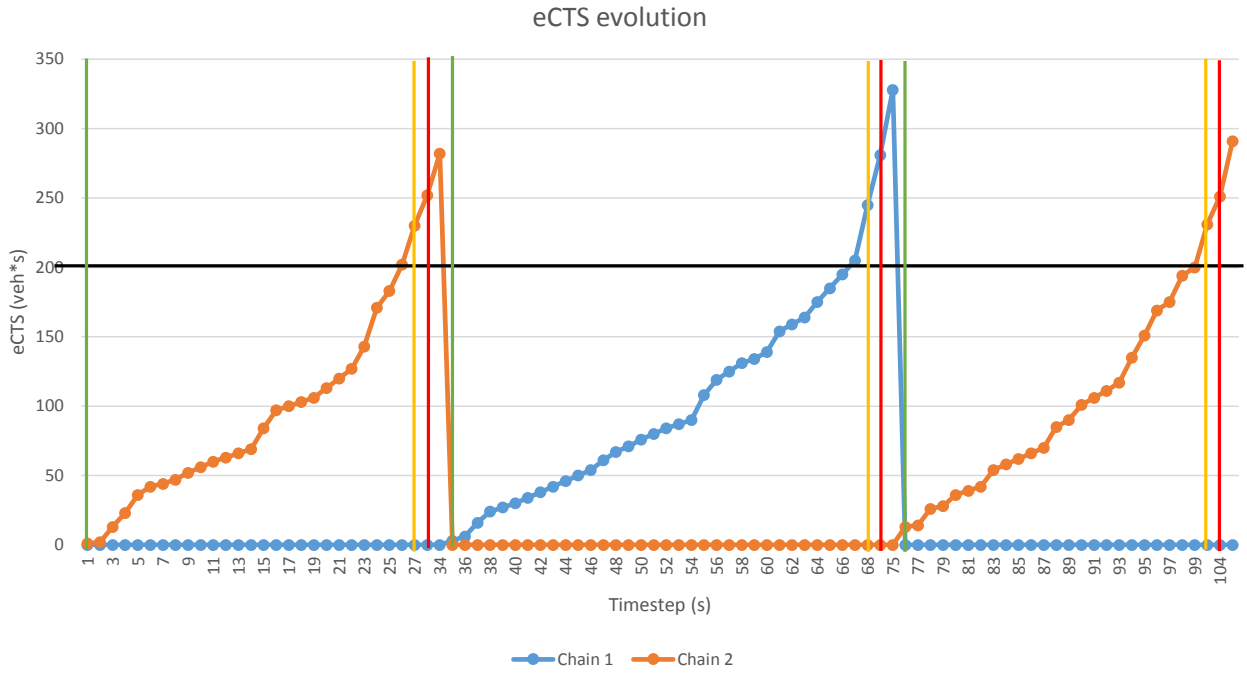


**Figure 3.2: Example of the evolution of eCTS values of two chains with Phase Policy activated. Vertical green, yellow and red lines indicate the change of the lights controlling the current chain target lanes and the black bold line is the traffic threshold. Chain 1 is the first one being executed.**

This approach is identical to the one described in subsection 3.1 with the exception that the cars are not counted. Instead, the number of cars is inferred from the speed, the acceleration and the position of the detectable cars.

When a car is detected, its speed and acceleration are collected. If the vehicle's speed is positive and no acceleration is measured, we infer it is flowing freely toward the intersection and it is discarded. As soon as a variation in the acceleration is detected, the vehicle may be used to infer the number of cars waiting in front of the intersection. If the vehicle's speed drops below a specific *speed threshold* we might infer this is caused by some undetectable vehicles in front of it. We then use its position and the average length of a vehicle to estimate the number of stopped cars. Let $d$ be the distance of the detected vehicle from the intersection and $l$ the average length of a vehicle, we have:

$$estimated\_\#cars = \frac{d}{l} \quad (3\text{-}2)$$

The estimated number of cars obtained with Equation (3-2) replaces $\#cars_{c\_target\_lanes}(t)$ from Equation (3-1).

The chain selection is done in the same way as for the CTS approach: the system selects the chain whose counter has the highest value.

As it was stated before, this approach defines a new parameter (see Table 3.1).

**Table 3.1: Additional parameters related to *eCTS***

| Parameter | Symbol | Meaning |
|---|---|---|
| `thresholdSpeed` | | Defines the fraction of the maximum allowed speed below which a vehicle is inferred to be stopping. |

## 3.3 Queue Length


**Figure 3.3: Example of the evolution of Queue Length values of two chains with Phase Policy activated. Vertical green, yellow and red lines indicate the change of the lights controlling the current chain target lanes and the black bold line is the traffic threshold. Chain 1 is the first one being executed.**

*The Queue length* approach differs substantially from *CTS* and *eCTS*. The traffic measure associated to the chain is the estimated length of the longest queue formed on its target lanes. This approach relies on the position of the detected vehicles.

Since the system updates only the traffic measures of the inactive chains, it seems implicit that all the vehicles detected on the target lanes are stopped in front of a red light. Based on this, if the system sees a solitary vehicle (Figure 3.4) it is because there are undetectable vehicles that prevent it to move further towards the intersection.

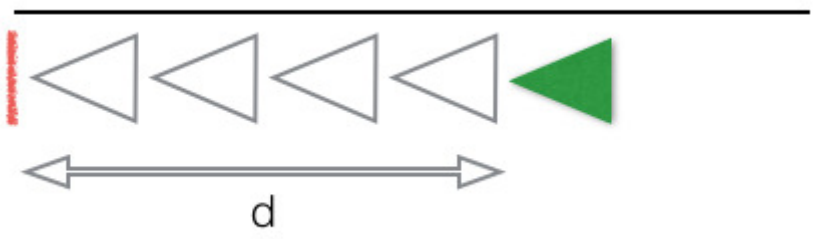


**Figure 3.4: A detectable vehicle (green) preceded by undetectable vehicles. The estimated queue length is equal to *d*.**

This approach looks for the farthest vehicle from the intersection for every target lane and keeps the maximum value as the traffic indicator associated to that chain.

As for all the previous cases, the selected chain is deterministic and the system chooses the one with the longest queue.

## 3.4 Dynamic Traffic Threshold

A quantity is subject to exponential decay if it decreases at a rate proportional to its current value following the equation:

$$N(t) = N_0 e^{\frac{-t}{\tau}} \tag{3-3}$$

where $N(t)$ is the quantity at time $t$, $N_0$ is the initial quantity (i.e. the quantity at time $t = 0$ and it is equal to 1 in our case) and $\tau$ is the *mean lifetime* (or the *exponential time constant*).

The dynamic traffic threshold, named *decayThreshold*, is introduced to prevent the possibility of a policy (*Platoon* or *Phase*) to execute the same decisional stage (which means the same chain) indefinitely, what is more likely to happen as the percentage of detectable vehicles decreases. The constant $\tau$ is a tunable parameter and its value should guarantee the execution of a decisional stage for a reasonable amount of time before forcing the current policy to end it. Whenever the traffic threshold is checked, the system verifies if the "traditional" traffic threshold or the dynamic one is reached. We use a random number between 0 and 1 to check if the *decayThreshold* is reached. We chose this probabilistic approach instead of waiting for it to drop to 0 so to not transform it into a simple counter.

Table 3.2 shows the two additional parameters we defined for the *Swarm* macroscopic level policy.

**Table 3.2: Additional parameters related to *decayThreshold***

| Parameter | Symbol | Meaning |
|---|---|---|
| `decay_threshold` | | 0 if the dynamic traffic threshold should not be used, 1 otherwise. |
| `decay_constant` | $\tau$ | Exponential time constant. |

# 4 Simulation Scenarios

COLOMBO's WP2 uses for two different tasks. Evaluation scenarios[3] are used to determine how the developed system performs. Additionally, scenarios used for configuring the traffic light algorithm are needed. In the following, the prepared scenarios are presented. The first one presented (subsection 4.1) is very simple and has no underlying study. On the other hand, the second scenario (subsection 4.2) is based on the German "Richtlinie für Lichtsignalanlagen" (guidance for traffic light systems), or RiLSA for short [FGSV, 2010a].

## 4.1 Controlled Cross Scenario

**Figure 4.1: Structure of the intersection.**

The simple scenario indicated in Figure 4.1 has been used for comparing the single policies one against the other to test their behaviour in presence of non-homogeneous traffic flows. A central intersection and four peripheral traffic lights on the outgoing lanes characterize this scenario. (In Figure 4.1 only one of these peripheral traffic lights is indicated explicitly.) While the four peripheral traffic lights use fixed timing, the central one is controlled by our algorithms. The distance between the central intersection and every peripheral traffic light highly influences the results and we tried the values 200 m and 500 m. We also run the same tests without those traffic lights, as already given in D2.2.

The timings for the static traffic lights are the following:

- Green stage of 31 s;
- Yellow stage of 4 s;
- Red stage of 4 s;
- An additional red stage of 39 s, whose duration is determined by the sum of the previously mentioned stages.

The additional red stage is introduced to force a synchronization between the four traffic lights. North and south traffic lights start with the additional red stage, while east and west traffic lights begin their cycle with the green stage. The peripheral traffic lights are needed to simulate any kind

[3] Evaluation scenarios are discussed within [COLOMBO D5.3, 2014].

of possible obstruction for the outgoing traffic flow. Their existence allows also to accumulate pheromone on the outgoing lanes.

The traffic light system controlling the central intersection is able to detect vehicles up to 100 m away for the input lanes and up to 80 m for the outgoing ones.

## 4.2 RiLSA

For configuring the traffic light algorithm, a scenario was used that is based on the first example from the German “Richtlinie für Lichtsignalanlagen” (guidance for traffic light systems), or RiLSA for short [FGSV, 2010a]. This guidance’ examples book [FGSV, 2010b] shows how traffic light timings are determined from given intersection layouts, traffic demand, and the stages of the traffic light.

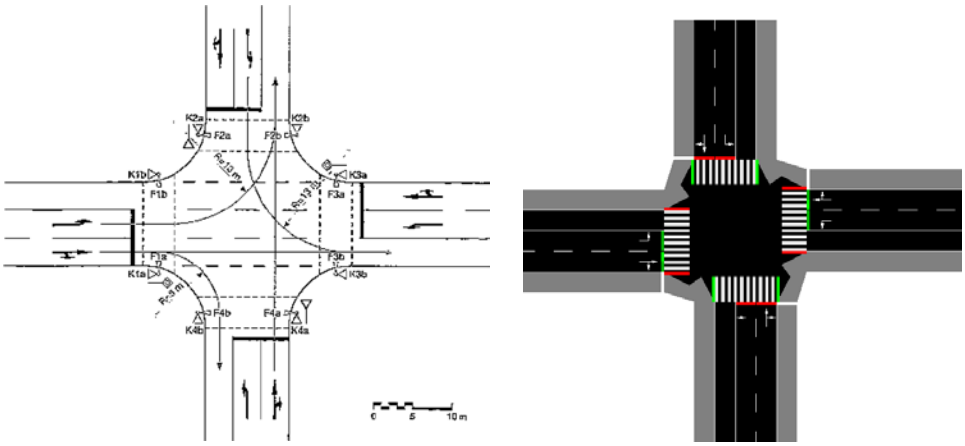

**Figure 4.2: First of the RiLSA examples; a) as shown within the RiLSA appendix, b) as simulated.**

The major intersection of the scenario, located in its middle, resembles the layout of the first RiLSA example, as shown in Figure 4.2. However, as the developed traffic light algorithm uses information about both, incoming and outgoing lanes, the downstream flow had to be limited to form jams that are visible to the algorithm. Therefore, the scenario was extended by additional traffic lights that affect the outgoing lanes only, depicted in Figure 4.3.

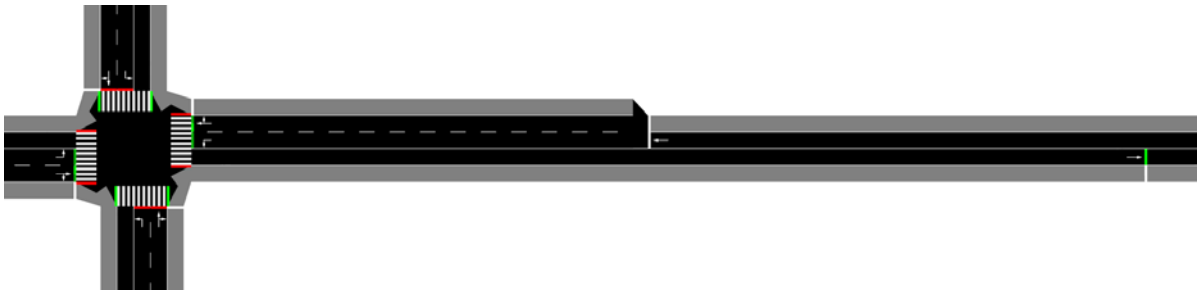

**Figure 4.3: Limiting the outflow by additional traffic lights (at the right side of the image).**

The scenario was aimed to be used with flows that are variable over time. Because the flows given in RiLSA are static, the traffic light timings from RiLSA could not be used. Instead, the green times were split proportionally to the maximum sums of the traffic flows coming from opposite directions, preserving the red and yellow times as given in RiLSA. The center traffic light gives green to the north-south direction for 38 s, and 34 s to the west-east direction. The overall cycle time is 82 s. The outer traffic lights have the same cycle time but restraint the outgoing flows by giving only 31 s green. All green lights switch to yellow for 4 seconds before becoming red. Because the simulated period covers only one hour, no attempts to build a daily program plan that changes over time was attempted.

### 4.2.1 Traffic Generator

The task was to develop a scenario that on the one hand resembles traffic patterns as occurring in the real-world, while on the other hand, is short in execution time to perform several simulations in reasonable time, which is something needed for the parameter tuning tasks (see Chapter 5). To obtain a realistic demand curve over time, one scenario from the scenario set "RiLSA1LoadCurves" as given in [COLOMBO D5.3, 2014] was used. This scenario uses daily load curves to determine a given TLS algorithm's performance in case of a demand that changes over a day. Based on initial evaluations of the scenario behaviour that included all load curves types[4], the combination of the three load curve types was chosen as following:

- North-south: afternoon peak;
- South-north: afternoon peak;
- East-west: two peaks;
- West-east: morning peak.

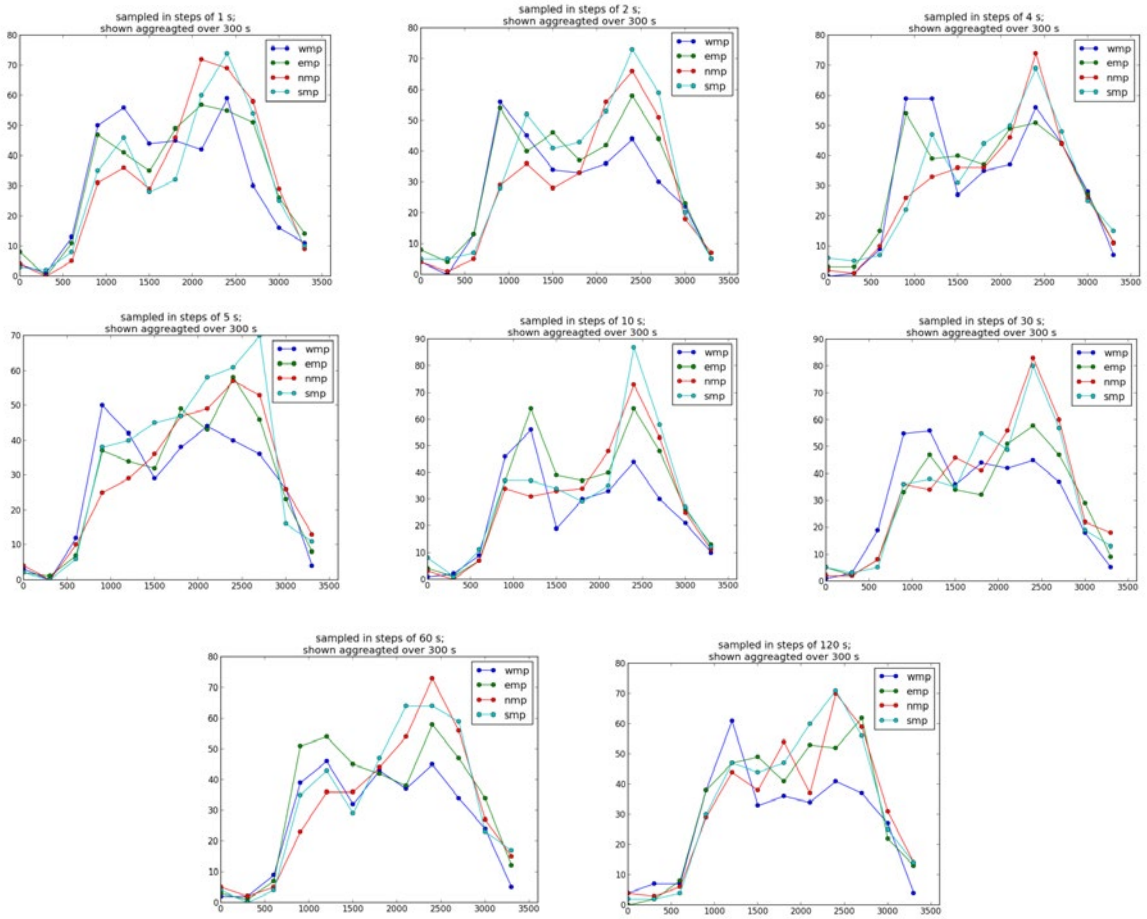


**Figure 4.4: Sampling examples as obtained from the demand generator (wmp: west-east; emp: east-west; nmp: north-south; smp: south-north).**

To obtain the configuration scenario that shall be fast in execution, the traffic demand had to be downsized to one hour. The first action was therefore to determine how the traffic demand can be sampled. The following sampling procedure was chosen:

- Build the load curve first (randomly) using the daily load curves

[4] Load curves and their application for simulation scenarios are discussed in D5.3 „ Traffic Light Algorithm Evaluation System " section 3.1

- Build a derived (“sampled”) demand by taking *x* seconds every *dt* seconds from the initial demand

Where

*dt=86400/3600*x*

To determine a sampling that matches the development of the original curve best, different x were tested: 1, 2, 4, 5, 10, 30, 60, and 120 s.

A first view shows that the curves start to deteriorate at higher sampling rates as expected. Therefore, a sampling of *x=1* was used. This was used as well as an additional “scenario set” within the Traffic Lights Evaluation System presented in [COLOMBO D5.3, 2014].

# 5 Parameter Tuning

The performance optimization of the traffic light control algorithm proposed in the COLOMBO project and described in Chapters 2 and 3 of this deliverable, has a large number of parameters that need to be appropriately set to achieve best possible performance. Automatic algorithm configuration tools have been shown to provide crucial advantages for the identification of performance optimizing parameter settings of parameterized algorithms, of which traffic light controllers are specific case, when compared to manual approaches to this task [Hoos, 2012]. Here, we followed this approach that was successfully applied to the tuning of heuristic and exact algorithms for tackling optimization problems and apply here for the first time automatic algorithm configuration techniques for the parameter tuning of traffic light controllers. In the following, we first give in Section 5.1 a high-level view of the used software, before we describe in Section 5.2 the setup used for the tuning and in Section 5.3 the tuning output.

## 5.1 Tuning Software

The irace package [López-Ibáñez et al., 2011] implements a generic iterated racing procedure for the configuration of algorithms.

Racing was first proposed in machine learning for model selection, and later adapted for the configuration of optimization algorithms. A racing procedure consists of evaluating candidate configurations over different instances, and discarding the worse configurations as soon as sufficient statistical evidence is found against them. This allows to restrict evaluations on further instances to the best configurations only, and typically produces much better results than the naive approach of testing all candidate configurations on all instances.

The iterated racing procedure than consists of these three main steps:

- Sample new configurations according to a particular distribution;
- Select the best configurations from the newly sampled ones by means of racing;
- Update the sampling distribution in order to bias the sampling towards the best configurations.

These three steps are repeated until a termination criterion is met. In iterated racing, each configurable parameter has an independent sampling distribution, which is a normal distribution for numerical parameters, or a discrete distribution for categorical parameters (those with values that do not support an ordering). The update of the distributions consists of modifying the sampling distributions, the mean and standard deviation in the case of the normal distribution, or the discrete probability values of the discrete distributions. The update biases the distributions to increase the probability of sampling, in future iterations, the parameter values in the best configurations found, or closely around them. After new configurations are sampled, the best configurations are selected by means of racing. The whole iterated racing procedure can be seen as an efficient way to implement positive reinforcement towards good configurations.

The purpose of the irace package is additionally to provide an easy way for the algorithm designer to make use of the underlying procedure. Tuning a software using irace is mainly done by writing a parameter file. This parameter file contains the name of the parameters, their types, the ranges of their possible values, and possibly some constraints to define when a parameter is relevant or not based on the value of other parameters.

A simple script, called "hook-run", must be written to provide an interface between irace and the software to be tuned. irace will always call the hook-run in a standard way, and this script is responsible to call the target software appropriately, and to return a single numerical value that represents the solution quality obtained by the candidate configuration in the evaluation.

The irace package itself is freely available at http://iridia.ulb.ac.be/irace. irace is implemented in R, a freely available, powerful statistical language, though no knowledge of R is needed to use irace. More details about the irace methods and software as well as recent improvements of the software are given in [COLOMBO D3.2, 2014].

## 5.2 Tuning Setup

For the tuning, cluster nodes available at the IRIDIA laboratory of the ULB partner have been used. The infrastructure is set up is done in two steps. Firstly, installation of SUMO with the traffic light control software must be done. This is a time consuming task as specific software needs to be installed and it has to be made sure that everything works in the cluster environment made of dozens of independent compute-nodes. Then, the preparation of the setup for tuning the traffic light algorithms essentially requires the definition of parameter files and the definition of "instances" of the problem that is to be tuned. In the following, we focus on these two aspects by first giving an example of the parameter files.

In Figure 5.1 we give the extract of the parameter file for the tuning that concerns the basic parameters that are common to all configurations of the traffic light control software. Given are for each parameter the name, the command line switch that is used to specify the parameter value for the code, the type of parameter ('r', 'I', 'c' indicate real-valued, integer, and categorical parameters, respectively), the range of possible values the parameter can take (indicated by the lower and upper bounds for real-valued and integer parameters) a possible condition that needs to be satisfied so that the specific parameter is enabled. There are 17 parameters of which two are integer while the other 15 are real-valued parameters. For each of the *Phase*, *Platoon*, and *Marching*, policies, additional parameters are required that have to be added to the parameter file.

```
# name                  switch                    type    values          [|conds (R syntax)]

threshold               "--threshold "            i       (10, 3000)
max_cong_d              "--max_cong_d "           i       (30, 120)
decay_constant          "--decay_constant "       r       (-0.001, -0.00001)
thrspeed                "--thrspeed "             r       (1, 8)
change_plan_p           "--change_plan_p "        r       (0.01000, 0.99000)
gamma_sp                "--gamma_sp "             r       (0.01000, 1.00000)
beta_sp                 "--beta_sp "              r       (0.01000, 0.99999)
gamma_no                "--gamma_no "             r       (0.01000, 1.00000)
beta_no                 "--beta_no "              r       (0.01000, 0.99999)
theta_max               "--theta_max "            r       (0.60000, 1.00000)
theta_min               "--theta_min "            r       (0.03000, 0.50000)
theta_init              "--theta_init "           r       (0.50000, 0.60000)
learning_cox            "--learning_cox "         r       (0.00010, 0.01000)
forgetting_cox          "--forgetting_cox "       r       (0.00010, 0.10000)
phase_stim_cox          "--phase_stim_cox "       r       (0.02000, 1.00000)
platoon_stim_cox        "--platoon_stim_cox "     r       (0.02000, 1.00000)
marching_stim_cox       "--marching_stim_cox  "   r       (0.02000, 1.00000)
```

**Figure 5.1: Example of a parameter file for the tuning of traffic lights; generic part common to all strategies. The parameter file is for tuning the variants where a single Gaussian function is used.**

In Figure 5.1, we give the excerpt of the parameter file corresponding to the Phase policy; the sections for the Platoon and Marching policy are analogous to the one for the Phase policy and have been omitted here. For the Phase, Platoon and Marching policies there are decisions taken whether the parameters $\varphi_{in}$, $\varphi_{out}$, $\varphi_{dsp\,in}$ (see Equations (2-2), (2-3) and (2-4)) should be used or not (for details see also Table 2.2 and Table 2.3). This decision whether the parameter is used or not is modeled as a categorical parameter in the tuning. If any of these three parameters is used, then each of them requires the setting of two additional, real-valued parameters. The need for these additional parameters is modeled by a condition as given in Figure 5.2.

In total, this makes 35 parameters that have to be tuned in the case where the traffic light control algorithm uses a single Gaussian function.

```
# name                          switch                              type  values
       [|conds (R syntax)]
#### PHASE
# in
phase_stim_cox_exp_in          "--phase_stim_cox_exp_in "          c     (0, 1)
phase_stim_divisor_in          "--phase_stim_divisor_in "          r     (1, 10.0)
       | phase_stim_cox_exp_in == "1"
phase_stim_offset_in           "--phase_stim_offset_in "           r     (0.00000, 10.0)
       | phase_stim_cox_exp_in == "1"

# out
phase_stim_cox_exp_out         "--phase_stim_cox_exp_out "         c     (0, 1)
phase_stim_divisor_out         "--phase_stim_divisor_out "         r     (1, 10.0)
       | phase_stim_cox_exp_out == "1"
phase_stim_offset_out          "--phase_stim_offset_out "          r     (0.0, 10.0)
       | phase_stim_cox_exp_out == "1"

# dispersion
phase_stim_cox_exp_disp_in     "--phase_stim_cox_exp_disp_in"      c     (0, 1)
phase_stim_divisor_disp_in     "--phase_stim_divisor_disp_in"      r     (1, 10.0)
       | phase_stim_cox_exp_disp_in == "1"
phase_stim_offset_disp_in      "--phase_stim_offset_disp_in"       r     (0.0, 10.0)
       | phase_stim_cox_exp_disp_in == "1"
```

**Figure 5.2: Example of a parameter file for the tuning of traffic lights; given is the part corresponding to the Phase policy. Note that many parameters here are conditional parameters, which is because the parameters relevant to the Phase policy are only needed in case the phase policy is actually chosen.**

As a second main step for allowing the tuning, we need to define “instances” of the problem. For the tuning, the scenario that was chosen for defining the instances is the first example of the German “Richtlinie für Lichtsignalanlagen” (RiLSA), which is described in more detail in Section 4.2.1. In the traffic light control scenarios, one single instance corresponds to a particular entry flow and distribution of vehicles. Note that the flow of vehicles is simulated based on a random distribution of the times vehicles enter the scenario, while once the vehicles entered, the simulation done by SUMO and the traffic light control is deterministic. The entry times of the vehicles hence are random and therefore subject to a variation that depends on the random number seed used for the simulation and specific parameter settings (vehicle maximum speed, simulated penetration rate etc.). For the usage of the irace tool for tuning, we generated a total of 1100 possible instances for each possible value of penetration rates in {100, 50, 25, 10, 5, 2.5, 1}% and approaches (CTS, eCTS and Queue length, as defined in 3.1, 3.2 and 3.3). Of these 1100 instances, 1000 were used for the tuning while 100 were reserved for a possible evaluation of the obtained configurations.

Each of the SUMO simulations corresponds to one hour and ten minutes of real time (that is, 4200 seconds). Vehicles that enter the net in the first 5 minutes are discarded as this time is twice the free-flow duration of a vehicle and corresponds to a “heat-up” phase for the simulation and they do not represent a realistic situation as initially the net is empty [Antoniou et al., 2014]. The solution quality for evaluating the performance of a specific setting of the parameters of the traffic light algorithms is the average "waiting steps" of vehicles (excluding the vehicles that entered the network in the first 5 minutes). SUMO simulations are divided into atomic time steps (1 s), and a waiting step occurs when a vehicle waits because of a signal or because of other vehicles. If vehicles are blocked in the network out of any reason, a soft penalization is applied, where each vehicle not leaving the net was given a number of waiting steps of 4200. This "equals" to a vehicle waiting during the whole simulation (1 h 10 min). If no vehicles at all leave the net, which corresponds to a complete grid-lock, the simulation is given a solution quality value of 4200. This results in a much stronger penalization than the soft penalization as in this case the value 4200 is not averaged but it is the final result.

The tuning runs were executed on dedicated compute nodes, each equipped with two AMD Opteron 6272 processors and 64 GB of main memory. Each of the processors has 16 cores, runs at 2.1GHz and has two times 16 MB of L2/L3 cache. Each of the tuning runs consisted of a budget of 20000 evaluations, that is, during each tuning 20000 simulations have been run. We used the version 1.05

of the irace software available at http://cran.r-project.org/web/packages/irace/. The irace software allows for a parallelization of the tuning runs, where several parameter configurations can be tested on an instance in parallel. Usually, tuning runs were using 10 to 50 processes in parallel, resulting in computation times for a single tuning of ca. 8 to 24 hours wall-clock time.

## 5.3 Tuning Output and Validation

From the tuning, as final output a set of surviving parameter configurations are returned. Typically, of these only the best performing one is used for further evaluation. The results obtained by these surviving candidates are detailed in the following Chapter.

# 6 Results

This section collects all the experimental data related to what we have described in this deliverable.

Subsection 6.1 collects the results of the comparisons between the policies in presence of non-homogeneous traffic flows. The data provided by those experiments led us to extend the policy selection algorithm, as described in Section 2.

Subsection 6.2 presents the evaluation for different penetration rates of equipped vehicles of what we have dubbed as *Swarm version 2* (see subsection 2.2). We tested all the adaptations presented in Chapter 3 against themselves, a static and a fully-actuated approach.

## 6.1 Single Policies Comparison with Non-homogeneous Traffic Flows

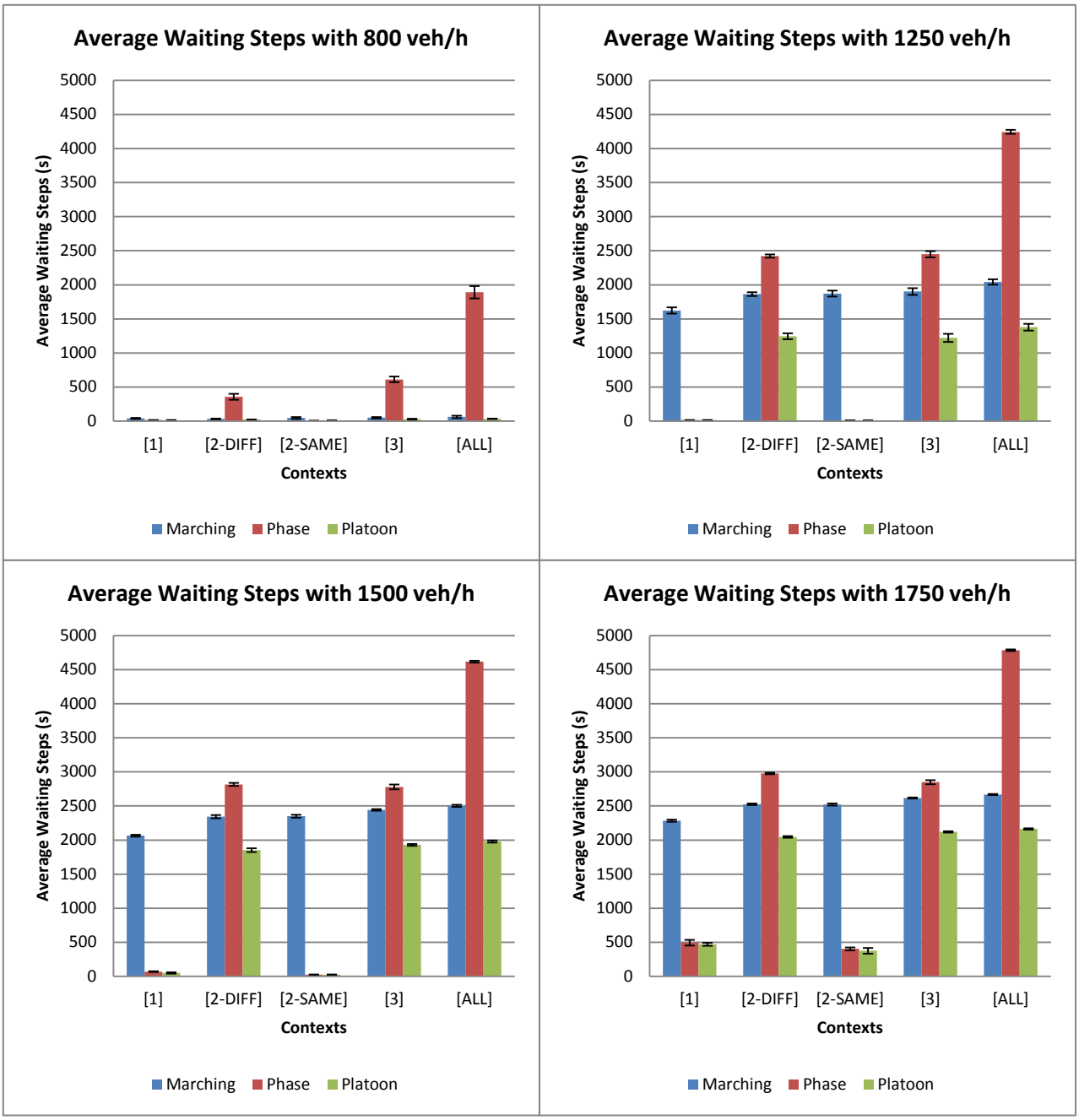


**Figure 6.1: Average waiting steps without peripheral traffic lights**

The scenario taken into account for this comparison is a simple standard crossroad, as described in subsection 4.1. Vehicles are modelled as standard cars, whose maximum speed allowed is 50 km/h. Drivers are supposed to behave well (no U-turns, no overtaking) and their route is straight from the point of departure to the arrival (no one turns when it reaches the central intersection).

We wanted to test five traffic conditions:

1. High traffic load on a single lane, low traffic load on the other three ([1]);
2. High traffic load on two lanes laying on different directions ([2_DIFF]);
3. High traffic load on two lanes laying on the same direction ([2_SAME]);
4. High traffic load on three lanes ([3]);
5. High traffic load on all the four lanes (this is the only homogeneous traffic load) ([ALL]).

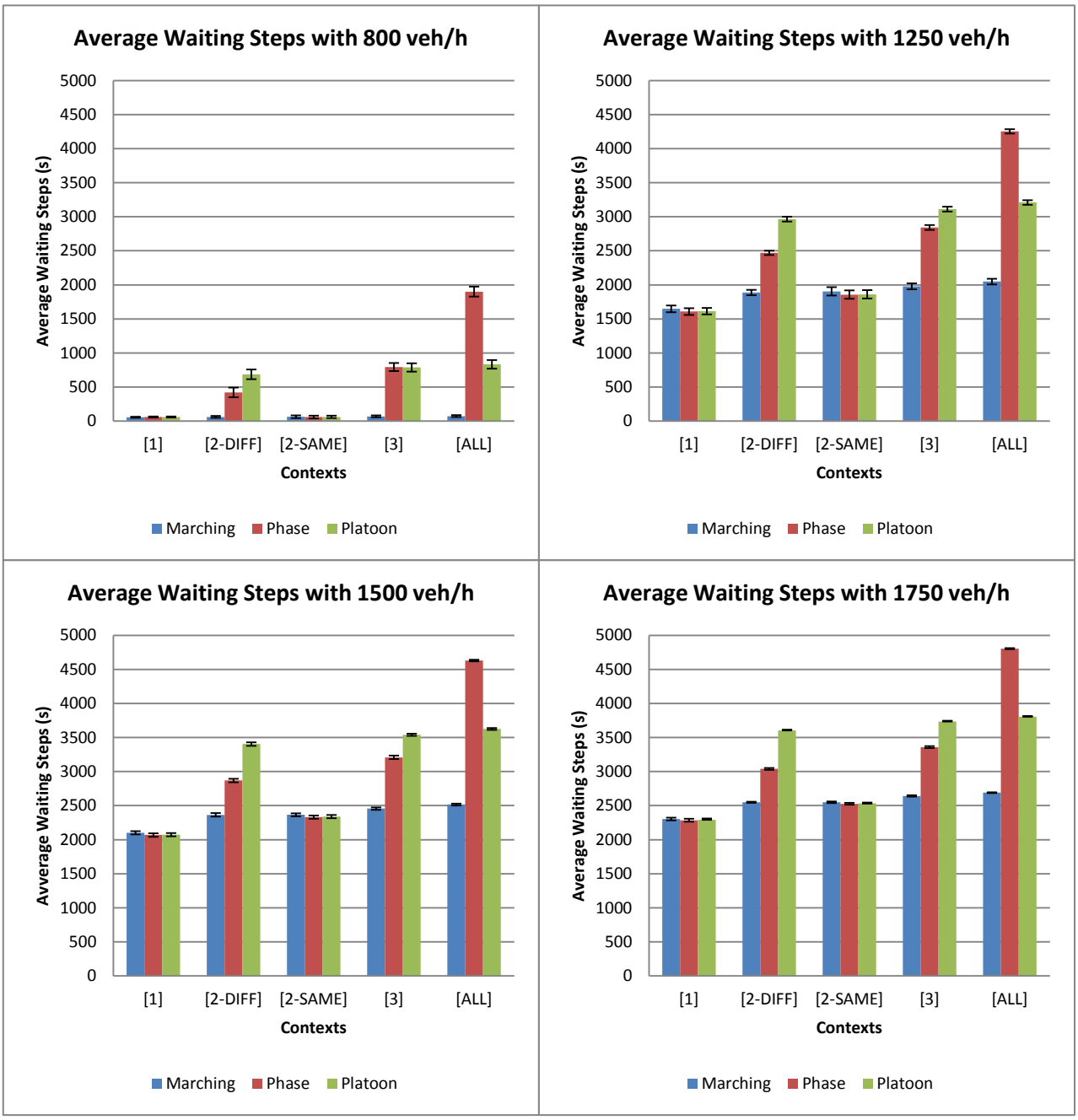


**Figure 6.2: Average waiting steps with peripheral traffic lights 200m away from central intersection.**

In order to obtain a significant amount of data, we analysed five different kind of vehicle rates: 800 veh/h (vehicles/hour), 1250 veh/h, 1500 veh/h and 1750 veh/h. The vehicle rate used for low traffic load is 100 vehicles/hour.

We first run some tests without the traffic lights on the outgoing lanes. *Platoon* policy outperforms every other policy in most cases, as Figure 6.1 shows. Both *Platoon* and *Phase* outperform *Marching* in contexts [1] and [2_SAME]. *Platoon* beats the other policies because it creates platoons of vehicles that are free to leave the network since nothing blocks them after they pass the central intersection.

We then added the four peripheral traffic lights, firstly 200 m away from the central intersection and secondly 500 m away from it. The results of the first case (see Figure 6.2) can be summarized this way:

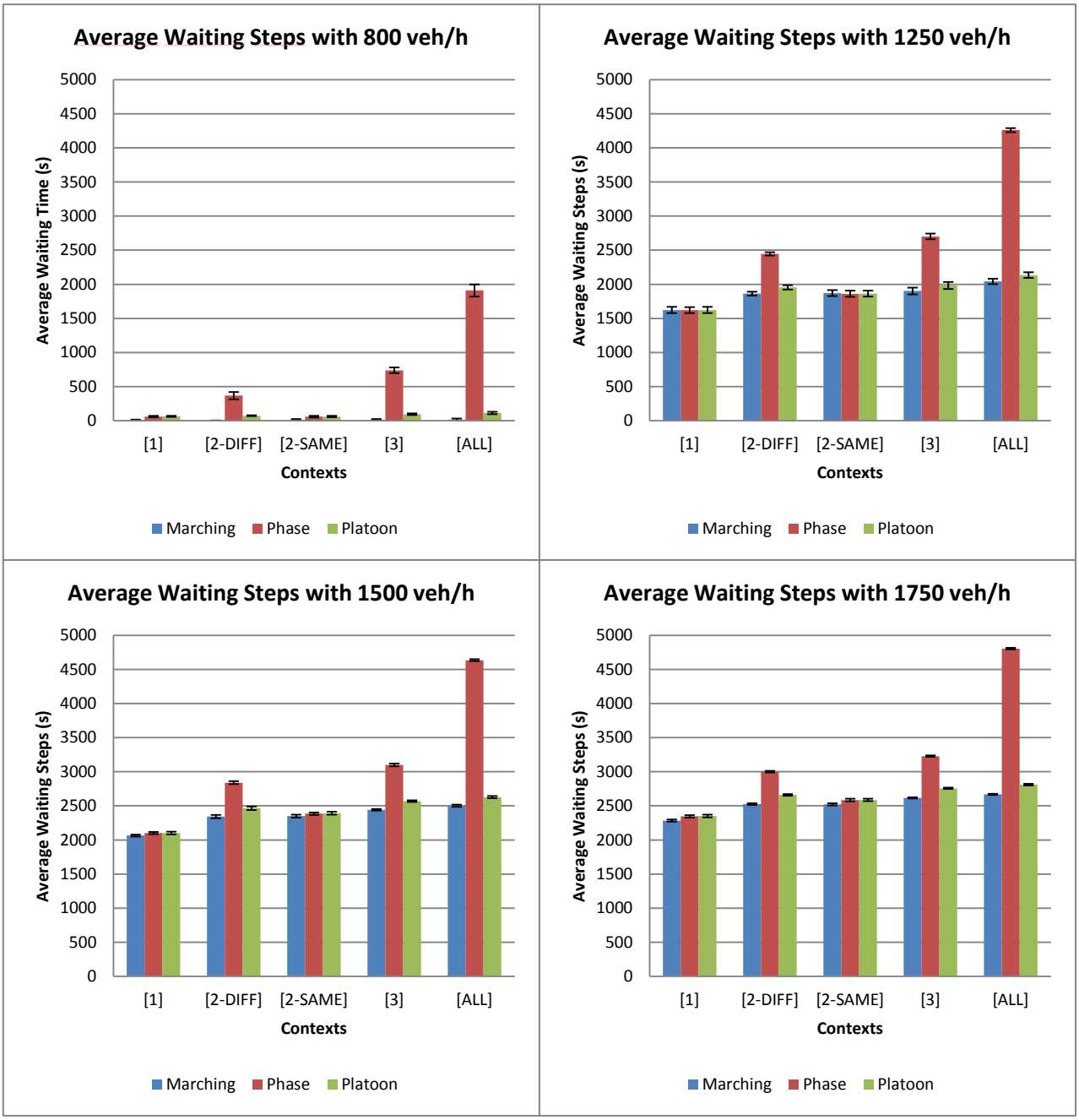


**Figure 6.3: Average waiting steps with peripheral traffic lights 500m away from central intersection**

- *Marching* works well in contexts [3], [2-DIFF] and [ALL] along with all heavy traffic loads. This policy adopts a static approach so this is why it works well when the traffic is balanced from all the incoming directions;
- *Phase* fits well in context [2-SAME] for every vehicle rates and in context [1] for high vehicle rates. This policy maintains the green light as long as there are no cars on the opposing directions. The results prove that this behaviour is desirable when there is a dominant traffic flow opposed by a sporadic one, which is exactly the context called [2-SAME].
- *Platoon* gets only few significant results in context [2-SAME] and [1] like *Phase* does.

The results of the last case with the peripheral traffic lights at a distance of 500m of the central traffic light differ heavily from the previous cases. The *Marching* policy shows the overall best results (Figure 6.3); in fact, in each case it appears to the best performing policy. This is quite tricky and it may suggest that the static behaviour of *Marching* paired with the distance of 500 m from the central intersection to the peripheral traffic lights might enable some sort of "green wave" effect. *Phase* and *Platoon* get good results that are comparable to *Marching* in contexts [1] and [2-SAME].

## 6.2 Evaluation of Swarm Version 2 Policy Selection Algorithm for different Penetration Rates

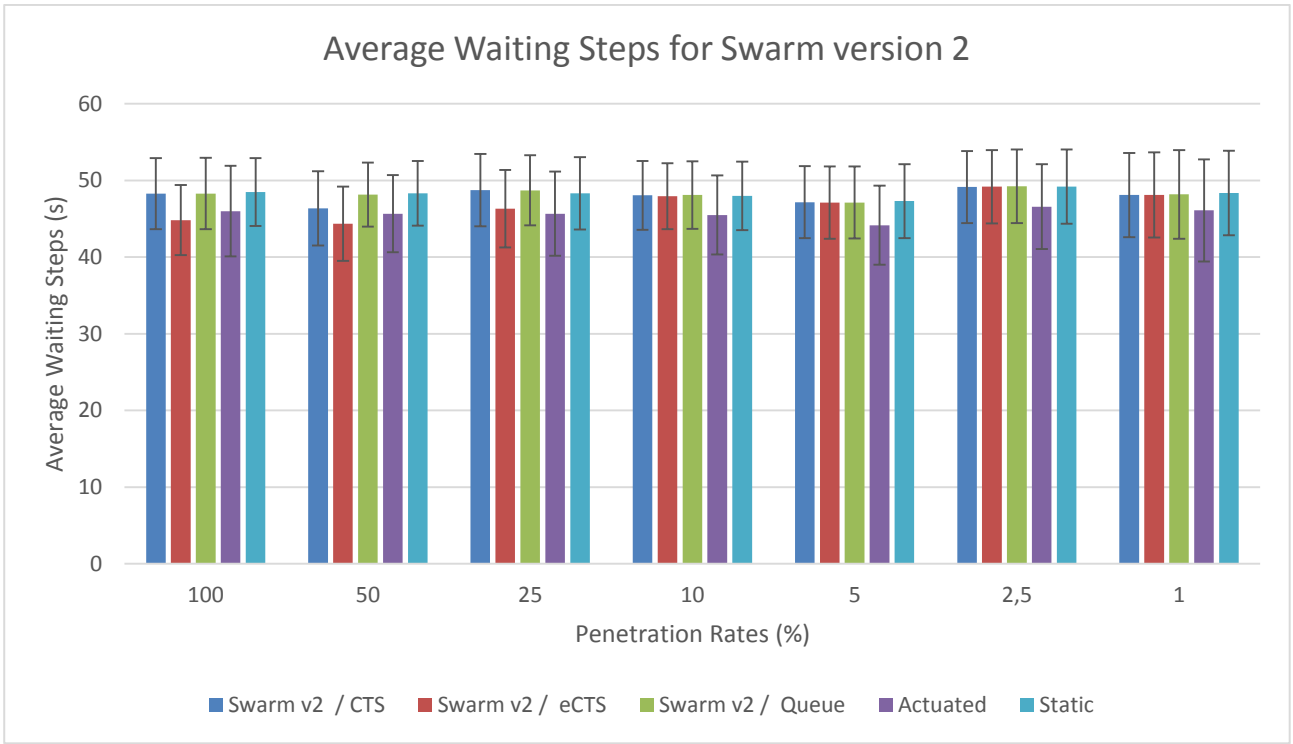


**Figure 6.4: Average Waiting Steps for Swarm version 2**

The scenario taken into account for this evaluation is the RiLSA (see subsection 4.2). The traffic instances were created with the aid of the traffic generator described in subsection 4.2.1.

We considered 100%, 50%, 25%, 10%, 5%, 2.5% and 1% penetration rates of equipped vehicles. Chapter 3 presents two chain selection algorithm adaptations for the low penetration rates challenge, named *estimated Cars-Times-Seconds* (*eCTS*) and *Queue length* (*Ql*). This evaluation employs both and the original one (*Cars-Times-Seconds* or *CTS*). The dynamic traffic threshold proposed in subsection 3.4 is disabled only during the evaluation of the 100% case. We created 100 traffic configurations for each penetration rate under examination. It was not possible to use the

same set of 100 traffic configurations for each penetration rate since the system needs the undetectable vehicles to be labelled with a specific type in the route file.

The applied parameters were the result of several automatic parameter tuning process done by ULB, using the tool described in Chapter 4 and following the process outlined there.. A different tuning process was run for every penetration rate (7) for every approach (3). The evaluations were performed using the best parameter configurations found by the automatic tuning process.

Figure 6.4 presents the averaged average waiting steps of the vehicles simulated for every set of the 100 traffic configurations for the different penetration rates. It compares the three alternatives presented in Chapter 3 against themselves, a static approach and a traffic actuated control system. It is worth mentioning that only our proposal is affected by the low penetration rate of equipped vehicles, since the actuated system relies on inductive loops and the static approach does not sense cars.

The results obtained do not point out a clear winner between the three alternatives proposed in Chapter 3. Since they all share the same performance, the best choice would be the simplest one, which is *CTS*, combined with the dynamic traffic threshold presented in subsection 3.4[5]. COLOMBO proposal proves to perform really well compared to the actuated control system. It is interesting to show that our system performance does not degrade with the decrease of the penetration rate. It is worth mentioning again *actuated* is not affected by the low penetration rates of equipped vehicles, i.e. it is always capable to sense 100% of the simulated cars. Our proposal is capable of obtaining comparable results even when detecting only 1% of the vehicles.

### 6.2.1 Applied Parameters

The parameters we applied for the previous evaluations are collected in appendix B.1. Every table gathers together the different values of the parameters for the different penetration rates.

The meaning of every parameter can be found in Table 2.1, Table 2.2, Table 2.3, Table 3.1, and Table 3.2. It is worth to point out some peculiar values related to some stimulus functions. The stimulus function of a policy is represented by a single Gaussian centred where the policy performs best (see subsections 2.1.5 and 2.2). Sometimes *irace* was not able to determine this single best spot. These cases are recognizable because a policy ignores all the three levels of pheromone (`STIM_COX_EXP_IN`, `STIM_COX_EXP_OUT` and `STIM_COX_EXP_DISPERSION_IN` all set to 0) and the stimulus function degenerate into a plane. For example, *Marching* policy stimulus function degenerates into a plane in almost all cases (see Table B.4, Table B.9 and Table B.14). This somehow weird result may further prove the need of further extensions to the policy selection algorithm.

[5] As we said before, the dynamic traffic threshold is necessary otherwise *Platoon* and *Phase* will get stuck easily on a decisional stage.

# 7 Next Steps

The promising results presented in subsection 6.2 show the COLOMBO proposal performance is comparable to more sophisticated systems, like the fully-actuated one. We think that a third version of the system may give better results. A possible improvement would be to extend the stimulus functions so to take into account more than one Gaussian. This will enable the possibility to define more than one optimal traffic condition for a given low-level policy and may address the weird parameter configurations of the *Marching* policy stimulus function. As it was done for *Swarm version 2*, an offline automatic parameter tuning process needs to be done before evaluating it.

The system should then be tuned on a more complex scenario. Even if that scenario is composed by nine of the already tuned scenarios, this does not imply that the obtained parameter may work well if applied to all the nine controllers. The interactions between neighbouring *swarm*-controlled intersections were not taken into account in the previous offline tuning procedure because there was only one of that traffic light controller. Basically, every scenario should be tuned by itself.

The system must also be adapted to include interactions with pedestrians and public transport, which is the aim of the next task (Task 2.5). The new extensions should be tested in the same way the current ones were examined. The system should be then tested using real scenarios, like Pasubio or Andrea Costa.

# 8 Summary

This document presents the evolution of the COLOMBO proposal from its preliminary state described into Deliverable D2.2. The previous system was designed with full knowledge of the traffic in mind. This simplification facilitated the design of the *Swarm* control algorithm.

The aim of COLOMBO is to develop a traffic light control system inspired from research in swarm intelligence that relies on information about the traffic state gathered using V2X communication assuming *low penetration rates* of equipped vehicles. The previous proposal has been extended on two sides: firstly, we enhanced the low-level policy selection algorithm; secondly, we adapted the whole *Swarm* algorithm to cope with the low penetration rates problem. The first enhancements aim to solve one of the simplifications of the system presented in D2.2. In that first version of the system, the stimulus function modelling the desirability of a low-level policy in the pheromone space was associated to a single Gaussian function. The pheromone space itself was a 2-dimensional space represented by the average pheromone on the input and on the output lanes. We identified that this solution was unable to distinguish accurately homogeneous and non-homogeneous traffic. We then designed the so-called *Swarm version 2* algorithm adding a new dimension to the pheromone space: the dispersion of the pheromone between the input lanes. We then further improved the stimulus functions by representing them with more than one Gaussian function, developing the so-called *Swarm version 3*. This third version of the algorithm allows to define more than one desirable position of the pheromone space for a single policy.

The second enhancements focus on the low penetration rates of equipped vehicles issue. The first thing we though was that the system performance should not degrade depending on the percentage of detectable vehicles. With this in mind, we checked our system for possible weaknesses related to traffic detection. The system senses vehicles in two occasions: the first one is related to pheromone update and the second one is related to the chain selection. While the pheromone is related to the average speed of the vehicles and should be insensible to the low-penetration rates, the chain selection algorithm relies on counting vehicles. The low penetration rates issue critically affects this approach, named *CTS*. We then designed two possible alternative implementations of the chain selection algorithm, called *eCTS* and *Queue length*.

Our system depends on more than 50 parameters whose values must be tuned properly. This task would be impossible to do by hand, so we rely on offline automatic parameter tuning procedures. We tuned *Swarm version 2* with all the three previously mentioned approaches. The provided results were quite interesting: the system is not only able to obtain comparable performance with all the selected penetration rates of equipped vehicles but also is comparable to a fully-actuated traffic light control system.

There is still more room for improvements: the *Swarm version 3* system still needs to be tuned properly and the results obtained for *Swarm version 2* let believe the performance will improve.

# Appendix A - References

[Antoniou et al., 2014] Antoniou, C., et al.: Traffic Simulation: Case for guidelines, Report of European Commission Joint Research Centre Institute for Energy and Transport, Luxembourg, 2014

[COLOMBO D2.2, 2014] COLOMBO project consortium: Policy Definition and dynamic Policy Selection Algorithms, deliverable 2.2, March 2014.

[COLOMBO D3.2, 2014] COLOMBO project consortium: Results of the offline Configuration and Tuning of the emergent Behaviour, deliverable 3.2, November 2014.

[COLOMBO D5.3, 2014] COLOMBO project consortium: Traffic Light Algorithm Evaluation System, deliverable 5.3, November 2014.

[FGSV, 2010a] FGSV (publisher): RiLSA: Richtlinien für Lichtsignalanlagen – Lichtzeichenanlagen für den Straßenverkehr, 2010.

[FGSV, 2010b] FGSV (publisher): Beispielsammlung zu den Richtlinien für Lichtsignalanlagen (RiLSA), 2010.

[Hoos, 2012] Hoos, H. H.: Programming by optimization. Communications of the ACM, 55(2):70–80, February 2012.

[López-Ibáñez et al., 2011] López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T. and Birattari, M.. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Universite´ Libre de Bruxelles, Belgium, 2011.

# Appendix B - Applied Parameters

## B.1 Evaluation of Swarm Version 2

### *CTS Approach*

**Table B.1: Swarm parameters for CTS**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| THRESHOLD | 2286 | 16 | 946 | 2352 | 2956 | 1179 | 2712 |
| MAX_CONGESTION_DUR | 74 | 52 | 82 | 45 | 97 | 90 | 37 |
| PHERO_MAXVAL | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 |
| DECAY_THRESHOLD | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| DECAY_CONSTANT | 0 | -0.00048 | -0.00026 | -0.00007 | -0.00064 | -0.00056 | -0.00057 |
| CHANGE_PLAN_PROBABILITY | 0.56310 | 0.19988 | 0.06433 | 0.18755 | 0.61011 | 0.26716 | 0.62490 |
| GAMMA_SP | 0.89905 | 0.59144 | 0.91290 | 0.10691 | 0.30131 | 0.15352 | 0.37381 |
| BETA_SP | 0.64313 | 0.92690 | 0.93382 | 0.52707 | 0.10708 | 0.52779 | 0.86271 |
| GAMMA_NO | 0.31000 | 0.42904 | 0.96219 | 0.24908 | 0.44529 | 0.12532 | 0.69861 |
| BETA_NO | 0.83889 | 0.75102 | 0.51353 | 0.55639 | 0.42291 | 0.84432 | 0.51162 |
| THETA_MAX | 0.66125 | 0.90225 | 0.64225 | 0.67558 | 0.80583 | 0.98378 | 0.95565 |
| THETA_MIN | 0.25956 | 0.09294 | 0.43704 | 0.22154 | 0.49855 | 0.26338 | 0.43763 |
| THETA_INIT | 0.52507 | 0.51469 | 0.58049 | 0.53891 | 0.53933 | 0.51707 | 0.51885 |
| LEARNING_COX | 0.00066 | 0.00856 | 0.00790 | 0.00375 | 0.00397 | 0.00126 | 0.00706 |
| FORGETTING_COX | 0.07134 | 0.06188 | 0.08769 | 0.01909 | 0.06121 | 0.06033 | 0.07305 |

**Table B.2: Phase stimulus function parameters for CTS**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| PHASE_STIM_COX | 0.94831 | 0.75735 | 0.63751 | 0.61177 | 0.63690 | 0.85595 | 0.48286 |
| PHASE_STIM_OFFSET_IN | 0 | 0 | 6.45502 | 0 | 0 | 7.38673 | 7.67059 |
| PHASE_STIM_OFFSET_OUT | 6.27292 | 9.59981 | 0 | 6.69171 | 7.18019 | 6.26700 | 1.16567 |
| PHASE_STIM_DIVISOR_IN | 1 | 1 | 5.35665 | 1 | 1 | 2.30886 | 4.63830 |
| PHASE_STIM_DIVISOR_OUT | 4.58452 | 3.49590 | 1 | 1.39360 | 6.12659 | 3.92120 | 3.76470 |
| PHASE_STIM_COX_EXP_IN | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| PHASE_STIM_COX_EXP_OUT | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| PHASE_STIM_OFFSET_DISPERSION_IN | 0 | 0 | 1.29923 | 0 | 0 | 2.33889 | 1.21124 |
| PHASE_STIM_DIVISOR_DISPERSION_IN | 1 | 1 | 3.76329 | 1 | 1 | 1.90149 | 1.87930 |
| PHASE_STIM_COX_EXP_DISPERSION_IN | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

**Table B.3: Platoon stimulus function parameters for CTS**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| PLATOON_STIM_COX | 0.23127 | 0.41502 | 0.40028 | 0.73009 | 0.28927 | 0.79987 | 0.76643 |
| PLATOON_STIM_OFFSET_IN | 0 | 0 | 0 | 8.53418 | 6.26289 | 0 | 4.15340 |
| PLATOON_STIM_OFFSET_OUT | 9.50974 | 0 | 8.32159 | 0 | 9.10271 | 4.44787 | 4.06422 |
| PLATOON_STIM_DIVISOR_IN | 1 | 1 | 1 | 1.57484 | 1.55487 | 1 | 4.64613 |
| PLATOON_STIM_DIVISOR_OUT | 4.33745 | 1 | 8.12316 | 1 | 8.62918 | 3.97277 | 8.97470 |
| PLATOON_STIM_COX_EXP_IN | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| PLATOON_STIM_COX_EXP_OUT | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| PLATOON_STIM_OFFSET_DISPERSION_IN | 5.02268 | 1.32207 | 5.53731 | 2.78214 | 2.76591 | 0 | 0.83022 |
| PLATOON_STIM_DIVISOR_DISPERSION_IN | 9.64432 | 2.16094 | 6.25619 | 3.51302 | 7.38942 | 1 | 8.65296 |
| PLATOON_STIM_COX_EXP_DISPERSION_IN | 1 | 1 | 1 | 1 | 1 | 0 | 1 |

**Table B.4: Marching stimulus function parameters for CTS**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| MARCHING_STIM_COX | 0.28822 | 0.89393 | 0.47286 | 0.72846 | 0.81310 | 0.47077 | 0.22770 |
| MARCHING_STIM_OFFSET_IN | 0 | 7.17515 | 0 | 0 | 0 | 0 | 0 |
| MARCHING_STIM_OFFSET_OUT | 0 | 9.12178 | 3.51693 | 0 | 0 | 0 | 0 |
| MARCHING_STIM_DIVISOR_IN | 1 | 2.57206 | 1 | 1 | 1 | 1 | 1 |
| MARCHING_STIM_DIVISOR_OUT | 1 | 2.28450 | 6.28178 | 1 | 1 | 1 | 1 |
| MARCHING_STIM_COX_EXP_IN | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| MARCHING_STIM_COX_EXP_OUT | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| MARCHING_STIM_OFFSET_DISPERSION_IN | 0 | 0 | 0 | 0 | 4.65686 | 0 | 0 |
| MARCHING_STIM_DIVISOR_DISPERSION_IN | 1 | 1 | 1 | 1 | 5.49312 | 1 | 1 |
| MARCHING_STIM_COX_EXP_DISPERSION_IN | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

**Table B.5: Congestion stimulus function parameters for CTS**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| CONGESTION_STIM_COX | 0.03871 | 0.43032 | 0.34338 | 0.40502 | 0.97589 | 0.56298 | 0.51318 |
| CONGESTION_STIM_OFFSET_IN | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CONGESTION_STIM_OFFSET_OUT | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| CONGESTION_STIM_DIVISOR_IN | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CONGESTION_STIM_DIVISOR_OUT | 4.82331 | 6.26977 | 2.17008 | 2.72902 | 4.99472 | 7.09408 | 2.91392 |
| CONGESTION_STIM_COX_EXP_IN | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CONGESTION_STIM_COX_EXP_OUT | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Table B.6: Swarm parameters for eCTS**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| THRESHOLDSPEED | 1.68077 | 3.82922 | 5.94737 | 7.68355 | 7.38426 | 3.31218 | 3.27949 |
| THRESHOLD | 19 | 17 | 89 | 1704 | 759 | 1521 | 2639 |
| MAX_CONGESTION_DUR | 36 | 69 | 57 | 89 | 49 | 91 | 73 |
| PHERO_MAXVAL | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 |
| DECAY_THRESHOLD | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| DECAY_CONSTANT | 0 | -0.00024 | -0.00086 | -0.00063 | -0.00080 | -0.00098 | -0.00048 |
| CHANGE_PLAN_PROBABILITY | 0.91620 | 0.24887 | 0.47044 | 0.11382 | 0.29294 | 0.75052 | 0.20604 |
| GAMMA_SP | 0.84021 | 0.88361 | 0.99487 | 0.82814 | 0.66268 | 0.67102 | 0.42206 |
| BETA_SP | 0.44776 | 0.01777 | 0.96380 | 0.58142 | 0.50617 | 0.65157 | 0.47037 |
| GAMMA_NO | 0.45372 | 0.47700 | 0.72755 | 0.37408 | 0.55870 | 0.20876 | 0.63181 |
| BETA_NO | 0.49337 | 0.15928 | 0.22190 | 0.06164 | 0.72051 | 0.65739 | 0.35381 |
| THETA_MAX | 0.66381 | 0.92479 | 0.62890 | 0.80884 | 0.60907 | 0.63932 | 0.92559 |
| THETA_MIN | 0.38341 | 0.11458 | 0.18587 | 0.22106 | 0.06073 | 0.44055 | 0.16871 |
| THETA_INIT | 0.56554 | 0.55418 | 0.53093 | 0.54198 | 0.57393 | 0.51578 | 0.53343 |
| LEARNING_COX | 0.00906 | 0.00073 | 0.00328 | 0.00210 | 0.00776 | 0.00551 | 0.00794 |
| FORGETTING_COX | 0.08382 | 0.01858 | 0.00137 | 0.06488 | 0.07635 | 0.05191 | 0.04571 |

**Table B.7: Phase stimulus function parameters for eCTS**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| PHASE_STIM_COX | 0.38613 | 0.92102 | 0.06972 | 0.77366 | 0.51349 | 0.60667 | 0.79157 |
| PHASE_STIM_OFFSET_IN | 9.07106 | 4.82293 | 5.38024 | 5.90930 | 3.38435 | 0 | 0 |
| PHASE_STIM_OFFSET_OUT | 3.27315 | 0 | 6.05142 | 3.37421 | 5.70946 | 7.83451 | 8.08585 |
| PHASE_STIM_DIVISOR_IN | 7.33554 | 5.87788 | 6.57716 | 8.14472 | 7.10317 | 1 | 1 |
| PHASE_STIM_DIVISOR_OUT | 1.57587 | 1 | 8.90865 | 2.59388 | 3.79228 | 6.61618 | 6.74067 |
| PHASE_STIM_COX_EXP_IN | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| PHASE_STIM_COX_EXP_OUT | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| PHASE_STIM_OFFSET_DISPERSION_IN | 0 | 9.41361 | 0 | 0 | 0 | 2.56113 | 0 |
| PHASE_STIM_DIVISOR_DISPERSION_IN | 1 | 9.03074 | 1 | 1 | 1 | 6.40747 | 1 |
| PHASE_STIM_COX_EXP_DISPERSION_IN | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

**Table B.8: Platoon stimulus function parameters for eCTS**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| PLATOON_STIM_COX | 0.49234 | 0.52762 | 0.36037 | 0.50613 | 0.16162 | 0.97981 | 0.60469 |
| PLATOON_STIM_OFFSET_IN | 0 | 0 | 0 | 1.60241 | 8.55448 | 5.24537 | 3.92872 |
| PLATOON_STIM_OFFSET_OUT | 0 | 3.12768 | 1.65016 | 7.90473 | 0 | 9.40789 | 1.11303 |
| PLATOON_STIM_DIVISOR_IN | 1 | 1 | 1 | 2.92659 | 6.14448 | 1.59528 | 5.88815 |
| PLATOON_STIM_DIVISOR_OUT | 1 | 3.42072 | 5.02953 | 1.48896 | 1 | 1.99464 | 6.85703 |
| PLATOON_STIM_COX_EXP_IN | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| PLATOON_STIM_COX_EXP_OUT | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| PLATOON_STIM_OFFSET_DISPERSION_IN | 0 | 0 | 0 | 0.78901 | 0 | 2.42238 | 6.52803 |
| PLATOON_STIM_DIVISOR_DISPERSION_IN | 1 | 1 | 1 | 8.83769 | 1 | 4.01840 | 5.46346 |
| PLATOON_STIM_COX_EXP_DISPERSION_IN | 0 | 0 | 0 | 1 | 0 | 1 | 1 |

**Table B.9: Marching stimulus function parameters for eCTS**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| MARCHING_STIM_COX | 0.52731 | 0.07788 | 0.47457 | 0.91951 | 0.71746 | 0.46805 | 0.99234 |
| MARCHING_STIM_OFFSET_IN | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MARCHING_STIM_OFFSET_OUT | 0 | 7.44128 | 0 | 0 | 0 | 0 | 0 |
| MARCHING_STIM_DIVISOR_IN | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| MARCHING_STIM_DIVISOR_OUT | 1 | 6.55910 | 1 | 1 | 1 | 1 | 1 |
| MARCHING_STIM_COX_EXP_IN | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MARCHING_STIM_COX_EXP_OUT | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| MARCHING_STIM_OFFSET_DISPERSION_IN | 8.28401 | 7.37094 | 9.32354 | 0 | 3.15110 | 0 | 0 |
| MARCHING_STIM_DIVISOR_DISPERSION_IN | 8.05067 | 3.26749 | 5.45412 | 1 | 7.75286 | 1 | 1 |
| MARCHING_STIM_COX_EXP_DISPERSION_IN | 1 | 1 | 1 | 0 | 1 | 0 | 0 |

**Table B.10: Congestion stimulus function parameters for eCTS**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| CONGESTION_STIM_COX | 0.25750 | 0.12404 | 0.31061 | 0.79457 | 0.50218 | 0.18848 | 0.87743 |
| CONGESTION_STIM_OFFSET_IN | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CONGESTION_STIM_OFFSET_OUT | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| CONGESTION_STIM_DIVISOR_IN | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CONGESTION_STIM_DIVISOR_OUT | 7.39529 | 3.54158 | 2.79387 | 3.23641 | 4.23448 | 6.68400 | 5.40738 |
| CONGESTION_STIM_COX_EXP_IN | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CONGESTION_STIM_COX_EXP_OUT | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Table B.11: Swarm parameters for Queue Length**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| THRESHOLD | 21 | 87 | 30 | 23 | 59 | 34 | 96 |
| MAX_CONGESTION_DUR | 111 | 99 | 94 | 83 | 77 | 43 | 54 |
| PHERO_MAXVAL | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 | 10.0 |
| DECAY_THRESHOLD | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| DECAY_CONSTANT | 0 | -0.00059 | -0.00037 | -0.00050 | -0.00097 | -0.00021 | -0.00053 |
| CHANGE_PLAN_PROBABILITY | 0.30968 | 0.98329 | 0.29733 | 0.83297 | 0.08420 | 0.51921 | 0.92985 |
| GAMMA_SP | 0.37533 | 0.45263 | 0.56512 | 0.68577 | 0.25556 | 0.18977 | 0.65799 |
| BETA_SP | 0.72073 | 0.97747 | 0.07477 | 0.53279 | 0.35757 | 0.53612 | 0.93784 |
| GAMMA_NO | 0.68093 | 0.55863 | 0.43234 | 0.28924 | 0.01054 | 0.34721 | 0.82441 |
| BETA_NO | 0.57019 | 0.92568 | 0.76028 | 0.31633 | 0.44090 | 0.82418 | 0.27398 |
| THETA_MAX | 0.81242 | 0.67795 | 0.84441 | 0.86341 | 0.90373 | 0.93551 | 0.64931 |
| THETA_MIN | 0.23823 | 0.35032 | 0.36506 | 0.39909 | 0.15850 | 0.42800 | 0.16290 |
| THETA_INIT | 0.55040 | 0.55107 | 0.57100 | 0.54278 | 0.53082 | 0.53731 | 0.51029 |
| LEARNING_COX | 0.00063 | 0.00735 | 0.00830 | 0.00276 | 0.00708 | 0.00078 | 0.00603 |
| FORGETTING_COX | 0.03428 | 0.01266 | 0.07100 | 0.04012 | 0.04528 | 0.04044 | 0.04598 |

**Table B.12: Phase stimulus function parameters for Queue Length**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| PHASE_STIM_COX | 0.75010 | 0.77239 | 0.84197 | 0.21059 | 0.45691 | 0.40265 | 0.05030 |
| PHASE_STIM_OFFSET_IN | 4.31842 | 8.55250 | 0 | 5.88827 | 8.59662 | 6.69996 | 2.57579 |
| PHASE_STIM_OFFSET_OUT | 0 | 7.25255 | 6.32022 | 0 | 3.17940 | 1.70121 | 6.09824 |
| PHASE_STIM_DIVISOR_IN | 1.99541 | 9.70966 | 1 | 3.93334 | 4.03216 | 4.46807 | 5.46237 |
| PHASE_STIM_DIVISOR_OUT | 1 | 3.74442 | 5.54528 | 1 | 4.97195 | 2.95241 | 1.89409 |
| PHASE_STIM_COX_EXP_IN | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| PHASE_STIM_COX_EXP_OUT | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| PHASE_STIM_OFFSET_DISPERSION_IN | 4.90397 | 0 | 9.42686 | 0 | 5.21293 | 4.85951 | 0 |
| PHASE_STIM_DIVISOR_DISPERSION_IN | 5.54608 | 1 | 6.45766 | 1 | 6.25044 | 5.52505 | 1 |
| PHASE_STIM_COX_EXP_DISPERSION_IN | 1 | 0 | 1 | 0 | 1 | 1 | 0 |

**Table B.13: Platoon stimulus function parameters for Queue Length**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| PLATOON_STIM_COX | 0.15232 | 0.19725 | 0.92732 | 0.59358 | 0.20794 | 0.71429 | 0.60516 |
| PLATOON_STIM_OFFSET_IN | 5.12785 | 0 | 0 | 8.65336 | 9.47939 | 9.90401 | 7.14154 |
| PLATOON_STIM_OFFSET_OUT | 6.19013 | 8.72867 | 8.47703 | 4.15995 | 0 | 7.73514 | 1.60644 |
| PLATOON_STIM_DIVISOR_IN | 9.45824 | 1 | 1 | 7.58846 | 2.12720 | 7.33914 | 4.45178 |
| PLATOON_STIM_DIVISOR_OUT | 9.09681 | 8.91562 | 9.63246 | 9.35815 | 1 | 5.44070 | 4.23272 |
| PLATOON_STIM_COX_EXP_IN | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| PLATOON_STIM_COX_EXP_OUT | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| PLATOON_STIM_OFFSET_DISPERSION_IN | 0 | 0 | 0 | 4.65889 | 0 | 0 | 6.73804 |
| PLATOON_STIM_DIVISOR_DISPERSION_IN | 1 | 1 | 1 | 1.87947 | 1 | 1 | 7.39910 |
| PLATOON_STIM_COX_EXP_DISPERSION_IN | 0 | 0 | 0 | 1 | 0 | 0 | 1 |

**Table B.14: Marching stimulus function parameters for Queue Length**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| MARCHING_STIM_COX | 0.41918 | 0.76407 | 0.28616 | 0.42176 | 0.35560 | 0.42681 | 0.72774 |
| MARCHING_STIM_OFFSET_IN | 0 | 5.72932 | 0 | 0 | 0 | 0 | 0 |
| MARCHING_STIM_OFFSET_OUT | 0 | 0 | 0 | 0 | 0 | 2.41108 | 0 |
| MARCHING_STIM_DIVISOR_IN | 1 | 9.06427 | 1 | 1 | 1 | 1 | 1 |
| MARCHING_STIM_DIVISOR_OUT | 1 | 1 | 1 | 1 | 1 | 8.31224 | 1 |
| MARCHING_STIM_COX_EXP_IN | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| MARCHING_STIM_COX_EXP_OUT | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| MARCHING_STIM_OFFSET_DISPERSION_IN | 0 | 9.52732 | 0 | 0 | 0 | 0 | 0 |
| MARCHING_STIM_DIVISOR_DISPERSION_IN | 1 | 7.33751 | 1 | 1 | 1 | 1 | 1 |
| MARCHING_STIM_COX_EXP_DISPERSION_IN | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

**Table B.15: Congestion stimulus function parameters for Queue Length**

| Parameter | Value | | | | | | |
|---|---|---|---|---|---|---|---|
| | PR 100 | PR 50 | PR 25 | PR 10 | PR 5 | PR 2.5 | PR 1 |
| CONGESTION_STIM_COX | 0.79057 | 0.66866 | 0.62293 | 0.51124 | 0.14569 | 0.17642 | 0.85468 |
| CONGESTION_STIM_OFFSET_IN | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CONGESTION_STIM_OFFSET_OUT | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| CONGESTION_STIM_DIVISOR_IN | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| CONGESTION_STIM_DIVISOR_OUT | 5.20624 | 3.57336 | 4.93739 | 6.72991 | 9.71830 | 7.74525 | 6.25005 |
| CONGESTION_STIM_COX_EXP_IN | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CONGESTION_STIM_COX_EXP_OUT | 1 | 1 | 1 | 1 | 1 | 1 | 1 |