Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Control Engineering

Modeling and Optimization of Traffic Flow in Urban Areas

Doctoral Thesis

Michal Kutil

Prague, January 2010

Ph.D. Programme: Electrical Engineering and Information Technology
Branch of study: Control Engineering and Robotics

Supervisor: ***Doc. Dr. Ing. Zdeněk Hanzálek***

This thesis is dedicated to my wife
and family members, with love.

Acknowledgements

I would like to give my great thanks to my thesis advisor Zdeněk Hanzálek for his patient support throughout the years it has taken me to create this thesis. I would also like to express my thanks to all my colleagues helping to create a friendly atmosphere. I would also like to thank to many anonymous reviewers of journals and international conferences where preliminary versions of this thesis were submitted. Their comments significantly contributed to the quality of this thesis. Last, but not least, thanks belongs to all my family.

This work was supported by the Ministry of Education of the Czech Republic under project 1M0567.

Czech Technical University in Prague *Michal Kutil*
January 2010

Nomenclature

$\alpha\|\beta\|\gamma$	Graham and Błażewicz notation
$\alpha_{s\to d}$	distribution rate from the source street s to the destination d
a_{uv}	cost of edge from node u to node v
$\mathbf{A}$	linear state space matrix
b_i	demand of commodity i
$\mathbf{B}, \mathbf{B_h}$	linear state space matrices
c_j	completion time—time when the execution of the task is finished
cap_{uv}	capacity of edge from node u to node v
$\mathcal{C}_i$	set of transitions P_i° indexes
C	intersection cycle time
C_i	intersection i-th cycle time
CP	length of the critical path
δ	maximal allowed difference of t_{sw}
d	destination index of street
d_j	due date—time limit by which the task should be completed
$\widetilde{d}_j$	deadline—time limit by which the task has to be completed
$d_{s\to d}$	time for which one vehicle flows throw the intersection
$\mathtt{D}$	matrix of communication delay
D_j	tardiness
ϵ	small constant
$\mathcal{E}$	set of graph edges
E	mean value of waiting times
E°	mean value of waiting times in equilibrium point
E_i	mean value of waiting times in i-th queue
$f_i(u, v)$	flow of commodity i from node u to node v
$\mathbf{F}$	non-linear function
G	graph
h	incoming vehicle flow
$\mathbf{h}$	vector of incoming vehicles flows
h°	incoming vehicle flow in equilibrium point

h_i	incoming vehicle flow into the i-th queue
i,j	indexes
J	objective cost function
k	index (especially used for discrete time)
K_i	i-th commodity
l	index
ℓ_{uv}	number of lanes in the street from the intersection u to the v
l_{uv}	length of the unit vehicle including the distance between them
L_j	lateness of task j
μ	minimal duration of the phase split
M_0	initial marking
n	number of vehicles in the queue—queue length
n°	number of vehicles in the queue in equilibrium point
n_i	number of vehicles in the i-th queue
$\mathcal{O}$	asymptotic time complexity
π_i	priority of transition T_i
p_i	processing time—time necessary for execution of task i
$\mathcal{P}$	Petri net set of places
P_i	i-th Petri net place
${}^\circ P_i$	set of input transitions of P_i
P_i°	set of output transitions of P_i
$Post$	Petri net postcondition matrix
Pre	Petri net precondition matrix
q	vehicles flow rate
$\mathbf{q}$	vehicles flow vector
q°	vehicles flow in equilibrium point
q_i	i-th vehicles flow
q_i^{max}	maximum feasible flow
ρ	density
r_j	release date—time at which a task becomes ready for execution
$\mathcal{R}$	total number of processors
R	Petri net sextuple

R_0^+	set of non-negative real numbers
s	source index of street
s_j	start time
$sink_i$	i-th sink node of graph
$source_i$	i-th source node of graph
$\mathcal{S}$	maximum number of time units
S	sum of the waiting times of all vehicles currently in the queue
τ_{ik}	split—time interval of phase k for which the vehicle flow can go through the intersection i
t	time
t_{sw}	switching time—time to pass from one phase to the other phase
$t_{s\to d}$	vehicle travel time from the s street to d street
$\mathcal{T}$	Petri net set of transitions
T_i	i-th Petri net transition
${}^\circ T_i$	set of input places of T_i
T_i°	set of output places of T_i
T^{ph}	NMPC prediction horizon
$\mathbb{T}_i$	task i
uv	unit vehicle
u, v	graph nodes
v_i	speed of continuous Petri net transition T_i
v_i^{max}	maximum permissible speed of continuous transitions T_i
v_i^s	sum of the speeds supplying the place P_i
$\mathcal{V}$	set of graph nodes
V	vector of maximal firing speeds
V_i	maximal speed of transition T_i
$V_{s\to d}$	maximum vehicle speed from the street s to d
$V_{s\to}$	common maximum speed from the street s
w	vehicle speed
$w_{s\to d}$	vehicle speed crossing the intersection from the street s to d
W_{uv}	maximal alloved vehicle speed
$\mathbf{x}$	state vector of extended queue model

$\mathbf{x_i}$	state vector of extended i-th queue model
$\mathbf{x_c}$	state vector of complete queue model
$\mathbf{x_M}$	simple intersection state space vector
x_i	number of vehicles that have been waiting for i time units
$\varkappa_{ijk}$	binary variable of SAT scheduling
φ_{ij}	offset—time delay between phases of two intersections (i, j)
ψ	number of commodities

Abbreviations

CCPN	Constant speed Continuous Petri Net
CNF	Conjunctive Normal Form
DSP	Digital Signal Processing
HPN	Hybrid Petri Net
ITS	Intelligent Transportation System
ILP	Integer Linear Programing
LP	Linear Programming
LQR	Linear Quadratic Regulator
MMCF	Minimum cost Multi-Commodity Flow
NMPC	Nonlinear Model Predictive Controller
PN	Petri Net
SAT	SATisfiability of boolean expression problem
TORSCHE	TORSCHE Scheduling Toolbox for Matlab
UML	Unified Modeling Language
ZOH	Zero-Order Hold

Modeling and Optimization of Traffic Flow in Urban Areas

Ing. Michal Kutil
Czech Technical University in Prague, 2010

Thesis Advisor: Doc. Dr. Ing. Zdeněk Hanzálek

This thesis presents models of simple and general traffic intersections. The simple intersection model is based on the new queue model, where the state variables represent the queue lengths and the mean waiting times in the queues. The general traffic intersection model is based on a constant speed continuous Petri net. The continuous Petri net tool offers more flexibility of modeling general intersections and possibility interconnects them to the complex urban traffic region model in future. The Model is innovative, firstly, by the free space modeling together with the opposite direction of the vehicular flow, secondly by the continuous Petri net use only leading to a smaller state space. For this purpose, we show a new method for conflict resolution in a continuous Petri net based on the maximal speed proportion. The control of intersections is prosed firstly for simple intersection, where the waiting times of the individual vehicles in the various streets of the intersection are taken into account to some degree. Secondly for the urban traffic region model, where the scheduling is used to find an optimal control of light controlled intersections. The performance of all depicted models and their control was evaluated in simulations and compared with real data from the traffic in Prague.

Goals and Objectives

This thesis will focus on modeling and optimization techniques to improve efficiency of light controlled intersections in urban area. The objective of this optimization is to decrease congestions, accidents and environmental load. From this purpose the goals of this work were set as follows:

1. Describe urban traffic intersection and propose the modern control method to optimize controlling of intersection with regards to drivers waiting time balancing.

2. Make a model of general light controlled intersection based on constant speed continuous Petri net.

3. Improve control of light controlled intersection traffic region.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The urban traffic is significantly increasing in the large cities. Fig. 1.1 shows the progress of registered vehicles in Prague during this decade [Prag 09]. As the number of vehicles in the street (given by the *density* ρ) is increasing, the average *speed* w is decreasing. Similar to computer networks, from a certain state increasing density ρ decreases the traffic *flow rate* q, which corresponds to a throughput of vehicles through the city. The cut-off point is given by the traffic stream model [Haef 98] describing relationships among these parameters. See Fig. 1.2, where the fundamental diagram (graphical representation of traffic stream model) of real data from the Prague street "Zborovská" is shown. Large number of vehicles and traffic flow decreasing to zero brings

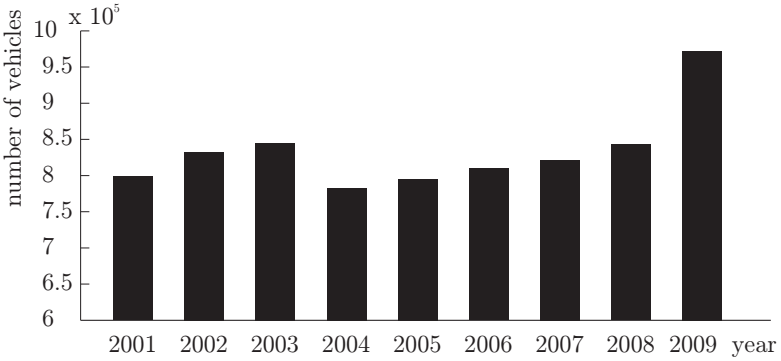
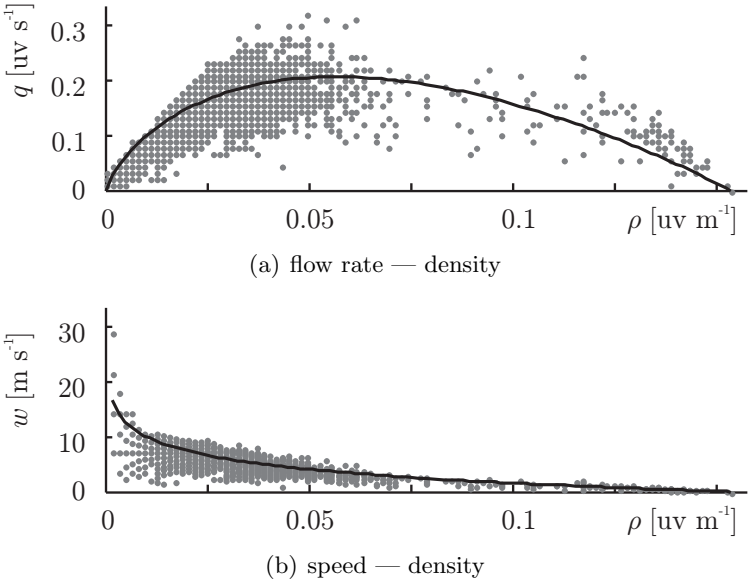


Fig. 1.1: Number of registered vehicles in Prague (different methodology was used in the years between 2004 and 2008)



(a) flow rate — density

(b) speed — density

Fig. 1.2: Fundamental diagram of real data measured in Prague street

more congestion, accidents, longer delays and increasing environmental load.

Described negative effects led to the study of *intelligent transportation systems* (ITS) [Figu 01], dated back to the 1980s. ITS make use of leading edge information and telecommunication technologies to provide traffic and transport information. It increases the efficiency of traffic management, improves the overall capacity of the road system, enhances road safety and reduces vehicle wear, transportation times, and fuel consumption. ITS encompasses traffic control and surveillance, public transport information, electronic tool collection, fleet management, car navigation systems, etc. [Man 00].

1.1 Related Work

A lot of work has been done on intelligent transportation systems. Review of road traffic control strategies, including the traffic modeling, road traffic control and freeway traffic control was summarized by Markos Papageorgiou et al. [Papa 03].

Traffic stream models [Cast 96], [Masu 07] consider traffic flow as a heterogeneous system which can be described from a macroscopic or microscopic point of view [Tolb 05]. The macroscopic point of view is described by global variables as the flow rate, the flow density and the flow average velocity [Gazi 02]. On the other hand, microscopic point of view focuses on the individual vehicles behavior on the road [Nage 03].

The light controlled intersections are characterized by several parameters: the number of light phases, phase *split* and *offset* time [Gube 08]. The term phase means the state of traffic lights on the intersection. Phase split defines the time interval of phase for which the vehicle flow can go through the intersection. The offset is a certain time delay between phases of two successive intersections [Papa 03].

The control of a single intersection (belonging to the class of road traffic control) is usually based on a *fixed-time* strategy or on a *traffic-response* strategy. In a fixed-time strategy (see, e.g., the TRANSYT tool [Robe 69]), the light control phases are scheduled offline. This approach is optimal only in the case of the under-saturated intersections. The light control phases are derived from the historical data measured for a given intersection. There are typically several light control phases for each intersection, depending on the given time of the day [Febb 06]. The traffic-response strategies are based on feedback from the current state of the traffic (e.g. the SCOTT tool [Hunt 82]).

The optimization of the traffic flow over several light controlled intersections in urban traffic region can be improved by the synchronization of lights. The three main synchronization strategies are: synchronized, green wave and random strategy [He 06, Naga 07]. In the synchronized traffic light strategy, all traffic lights switch from red (green) to green (red) simultaneously. In the green wave traffic light strategy, the lights switch with a certain time delay between two successive lights. In the random switching traffic lights strategy, all traffic lights change independently in an allowed range.

1.2 Outline and Contribution

This thesis describes use of appropriate modeling and optimization techniques to improve efficiency of light controlled intersections in urban area. The main contributions are:

- Simple traffic intersection model (extended by the mean waiting time) used for balancing of waiting time (Section 2.3).
- General light controlled intersection model based on a constant speed continuous Petri net. The model is innovative:
  - first, by the use of continuous Petri net leading to a smaller state space (Subsection 3.3.1).
  - second, by the modeling of free space together with the opposite direction of the vehicle flow (Subsection 3.3.1),
- New method for conflict resolution in a continuous Petri net based on the maximal speed proportion (Subsection 3.2.1) used to improve the behavior of the intersection model.
- Design of a tool architecture for development of scheduling and optimization algorithms in the Matlab environment (Section 4.2)
- Original algorithm for scheduling of light controlled intersections (Section 5.3, Section 5.4).

This thesis is organized as follows: Chapter 2 presents a dynamic model of a simple traffic intersection, where the vehicles waiting time is balanced by the NMPC and LQR use. The Chapter 3 proposes a general light controlled intersection model based on a constant speed continuous Petri net. In contrast of previous chapter the continuous Petri net tool gives us more flexibility of modeling general intersections. The next two chapters of thesis describe the optimization of the traffic flow control in urban traffic region made by the operation research theory use. The TORSCHE Scheduling Toolbox for Matlab, presented in Chapter 4, is used to find the optimal offset and the split of the light controlled intersections in the urban traffic region (Chapter 5). The Chapter 6 concludes the thesis.

Chapter 2

Simple Light Controled Intersection Model

This chapter presents a novel dynamical model of a simple traffic intersection, where the state variables represent the queue lengths and the mean waiting times in the queues. Including the mean waiting times in the model allows for a more fair traffic control, where the waiting times of the individual vehicles in the various streets of the intersection are taken into account to some degree. The model is linearized and its parameters are estimated using real traffic data measured during one day in Prague. For the balancing of the waiting times, two different controllers are considered: a linear quadratic regulator and a nonlinear model predictive controller. The controllers are evaluated in simulations where real traffic data is used for the incoming flows.

2.1 Introduction

In urban traffic control, it is common to decompose the traffic infrastructure into microregions that describe particular streets and intersections. This chapter focuses on a dynamic model of a simple intersection, describing the evolution of the traffic situation by nonlinear difference equations. The main idea in this model is to introduce a simplification that allows a mathematical description of the traffic flow without the use of discrete variables. This simplifies the description of the system state space and opens up the possibility to use optimization and standard control algorithms.

The objective is to develop controllers for balancing the vehicle waiting times in the different streets of the intersection. For this purpose, we use real traffic data from Prague [Homo 05] to tune the intersection model and then develop two controllers: a linear quadratic regulator (LQR) and a nonlinear model predictive controller (NMPC).

Those controllers are offen used to control the number of vehicles in the queues—see the traffic-response strategies OPAC [Gart 83], PRODYN [Henr 83], RHODES [Sen 97], and TUC [Diak 02].

We use real data from detectors placed approximately 100 m in front of the intersection. In order to derive the incoming flow we process these data via Kalman filter [Homo 05], since this approach enables to estimate the queues of the length superior to 100 m. In addition to the classical store-and-forward strategies [Gazi 63, Abou 09] our model also incorporates the vehicle waiting time [Bonn 95], which is a crucial input to the controllers designed in this chapter. A similar approach was taken in [Henr 04], where non-linear difference state equations were used to model and control web server traffic.

2.2 Extended Queue Model

Classical traffic control strategies use the single variable n—the *number of vehicles in the queue*, measured in *unit vehicles* [uv]—as an input to the control law with the objective to minimize this value. If we want to increase the quality of the traffic control from the driver's point of view, we can add another objective: the waiting time. The waiting time is the time spent by the vehicle in the queue.

Let us first assume that we are able to track every vehicle and its waiting time in the queue. This will be referred to as the *complete queue model*. The state vector of this model can be written in the form

$$\mathbf{x_c} = (x_1, x_2, \ldots, x_i)^T \tag{2.1}$$

where x_i denotes the number of vehicles that have been waiting in the queue for i time units. The disadvantages of this model are the state equation complexity and the unbounded state vector. In fact, the complexity of the model prohibits the application of standard control techniques.

We next consider an approximate queue model with only two state variables. The first variable, n, is the number of vehicles in the queue, while

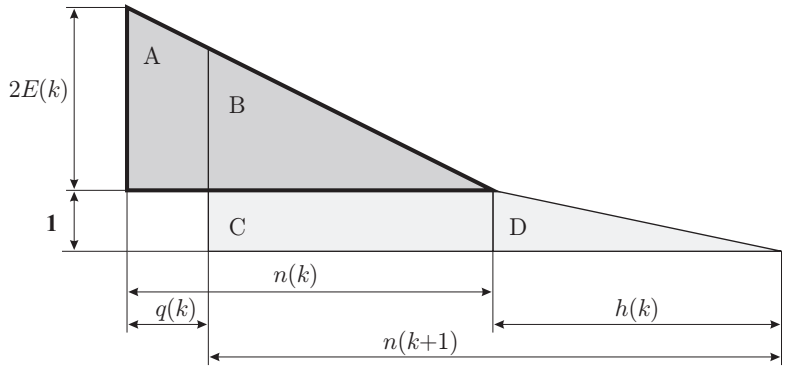


Fig. 2.1: Extended queue model evolution

the second variable, E [s], is the *mean value of waiting times*. E is given by S/n, where S is the sum of the waiting times of all vehicles currently in the queue. The state vector of this model is written in the form $\mathbf{x} = (n, E)^T$. This model will be referred to as the *extended queue model*.

2.2.1 Geometrical Interpretation

We here derive difference state equations for evolution of the extended queue model. The extended queue model evolution is dependent on the vehicles flows and the length of the time unit. We assume there is an (time-varying) incoming vehicle flow h [uv·s^{-1}] and an outgoing flow q [uv·s^{-1}] of vehicles leaving the queue.

A geometrical interpretation of the extended queue model is given in Fig. 2.1. The current state at time k is given by the bold triangle, where the base represents the queue length n and the height represents the longest waiting time in the queue. The longest waiting time is assumed to be twice the mean waiting time E. The area of the bold triangle represents the sum of waiting times over all vehicles: $S = E \cdot n$.

Next, we consider the evolution of the state from time k to $k + 1$. The incoming flow during this time interval is assumed to be $h(k)$, while the outgoing flow is $q(k)$. Studying Fig. 2.1, we have the following geometrical interpretation of the state evolution.

1. The outgoing flow $q(k)$ corresponds to the removal of the polygon A from the main triangle. The remaining vehicles are thus given by the triangle B.

2. All vehicles staying in the queue increase their waiting time by 1 unit. This corresponds to the addition of the rectangle C.

3. The incoming flow $h(k)$ is represented by the addition of the triangle D. Notice that the quantisation error is zero, when we ussume a constant incoming flow of vehicles from discrete time k to time $k+1$.

The new area (B+C+D) is equivalent to $S(k+1)$, i.e., the sum of waiting times over all vehicles at time $k+1$:

$$S(k+1) = \underbrace{\frac{E(k)(n(k)-q(k))^2}{n(k)}}_{B} + \underbrace{n(k)-q(k)}_{C} + \underbrace{\frac{h(k)}{2}}_{D} \tag{2.2}$$

Finally, using the fact $E(k) = S(k)/n(k)$, we arrive at the following discrete-time state equations:

$$n(k+1) = n(k) - q(k) + h(k) \tag{2.3}$$

$$E(k+1) = \frac{\frac{E(k)(n(k)-q(k))^2}{n(k)} + n(k) - q(k) + \frac{h(k)}{2}}{n(k) - q(k) + h(k)} \tag{2.4}$$

These equations are valid only for $n(k) > 0$ and $n(k) > q(k) - h(k)$. This means that there must be some vehicles in the queue, otherwise $E(k+1)$ is equal to zero.

2.2.2 Extended Queue Model Evaluation

To evaluate the extended queue model, we compared its ability to predict the mean waiting times to that of the complete queue model. (While the complete queue model has a complex mathematical description, its behavior can be simulated for a bounded number of vehicles.) As input data to both models we used input traffic flows taken from a real traffic region [Homo 05]. The result is shown in Fig. 2.2. It is seen that the extended queue model captures the mean waiting times of the vehicles quite well, justifying its use.

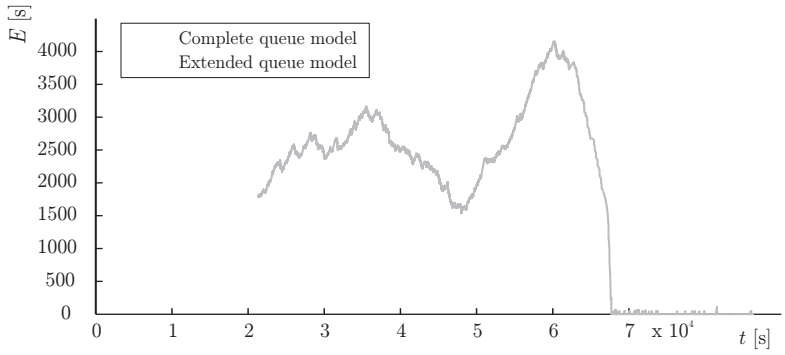


Fig. 2.2: Queue model evaluation

2.2.3 Extended Queue Model Equilibrium

For the purposes of linearization and further control synthesis, we want to find the equilibrium points, i.e., the points where $\mathbf{x}(k) = \mathbf{x}(k+1)$. The equilibrium points for our model must satisfy the conditions

$$n(k) = n(k+1) \tag{2.5}$$
$$E(k) = E(k+1) \tag{2.6}$$

Solution of these equations implies

$$q^{\circ}(k) = h^{\circ}(k) \tag{2.7}$$
$$2E^{\circ}(k) = \frac{n^{\circ}(k)}{q^{\circ}(k)} \tag{2.8}$$

(The circle mark means that the value of a given variable is the value in the equilibrium.) The condition (2.7) means that the incoming flow h must be equal to the outgoing flow q. The condition (2.8) implies that the mean value of the waiting times is proportional to the queue length and inversely proportional to the vehicle flow. This is the well known Little's law [Litt 61]. In our terminology, the condition says that "the average number of vehicles in a stable queue (over some time interval) is equal to their average incoming flow, multiplied by their average time in the queue."

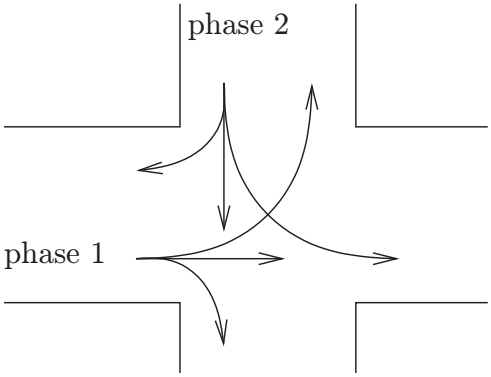


Fig. 2.3: Simple intersection model

2.3 Simple Intersection Model

The queue model described above will now be used to construct a *simple intersection model* (see Fig. 2.3). The intersection consists of two streets (i.e., two queues) and one crossing area (which is a shared resource). The outgoing flow q for each queue is controlled by the intersection lights.

The simple intersection model is described by

$$\mathbf{x_M}(k+1) = \mathbf{F}\left(\mathbf{x_M}(k), \mathbf{q}(k), \mathbf{h}(k)\right), \tag{2.9}$$

where $\mathbf{x_M}(k) = (\mathbf{x}_1(k), \mathbf{x}_2(k))^T$ contains the state vectors of the two queues. The full intersection state vector is hence given by

$$\mathbf{x_M}(k) = (n_1(k), E_1(k), n_2(k), E_2(k))^T \tag{2.10}$$

Here, $\mathbf{F}$ is a non-linear function given by (2.3) and (2.4). The vector $\mathbf{q}(k) = (q_1(k), q_2(k))^T$ represents the outgoing flow for the queues and the vector $\mathbf{h}(k) = (h_1(k), h_2(k))^T$ represents the incoming flow.

2.3.1 Linear model

A linear model is constructed via linearization of the function $\mathbf{F}$ around an equilibrium point (Subsection 2.2.3). The equilibrium point was selected

Table 2.1: Traffic data for the linearization of the intersection model.

	queue1	queue2
$q^\circ = h^\circ$	0.028	0.042
n°	20	50
$E^\circ = \frac{n^\circ}{2q^\circ}$ (2.8)	360	600

as an average point in the real traffic situation, described by the data in Table 2.1.

The linearized model can be written as

$$\mathbf{x_M}(k+1) = \mathbf{A}\mathbf{x_M}(k) + \mathbf{B}\mathbf{q}(k) + \mathbf{B_h}\mathbf{h}(k) \tag{2.11}$$

where $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{B_h}$ are given by

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.05 & 0.99 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0.02 & 0.99 \end{bmatrix},$$

$$\mathbf{B} = \begin{bmatrix} -1 & 0 \\ -17 & 0 \\ 0 & -1 \\ 0 & -11 \end{bmatrix}, \quad \mathbf{B_h} = \begin{bmatrix} 1 & 0 \\ -17 & 0 \\ 0 & 1 \\ 0 & -11 \end{bmatrix}.$$

2.4 Control of The Simple Intersection Model

The goal of the control is to find an optimal switching time for the traffic lights in the intersection, such that the difference in average waiting times between the two queues is minimized. In this section, two controllers will be designed and compared.

In general, an incoming flow of vehicles arriving at an intersection must be separated into several *phases*. The phase separation, designed by the traffic engineers, determines a direction of vehicles driving through the intersection. A repetitive sequence of phases form a *cycle time*. The phases have fixed

order in the cycle time and our goal is to find their optimal split. The *split* τ_{vj} defines the time interval of phase j for which the vehicle flow can go through the intersection v from one or more streets [Papa 03].

The simple intersection model defined above includes the two control phases. Each phase allows vehicles to flow only from one street, see Fig. 2.3. Our control algorithms consider a constant sum of the phase splits, i.e. constant cycle time C. In this section, the cycle time is assumed to be 90 s. The time when the first phase passes to the second one will be denoted the *switching time* t_{sw}. The switching time can be used to define a control law for the model (2.9) as follows:

$$\mathbf{q}(k) = \begin{cases} (q_1^{max}, 0)^T & \text{if } k \in \langle iC,\ iC + t_{sw}), \\ (0, q_2^{max})^T & \text{if } k \in \langle iC + t_{sw},\ (i+1)C), \end{cases} \tag{2.12}$$

Here, q_j^{max} is the maximum feasible outgoing flow from queue j and $i = 0, 1, 2, 3, \ldots$ is the index of the cycle time.

2.4.1 Linear Quadratic Regulator

In this subsection, a linear quadratic regulator (LQR) (e.g., [Kwak 72],[Astr 97]) will be used for the intersection control. The objective is to minimize the difference in the waiting times of the vehicles. This means that a vehicle entering a queue should wait the same time, regardless of which queue it is entering. This can be expressed as minimization of the cost function

$$J = \sum_k \left(E_1(k) - E_2(k)\right)^2. \tag{2.13}$$

Using (2.10), the cost function can be rewritten as

$$J = \sum_k \mathbf{x_M}(k)^T \mathbf{Q} \mathbf{x_M}(k) \tag{2.14}$$

where:

$$\mathbf{Q} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}.$$

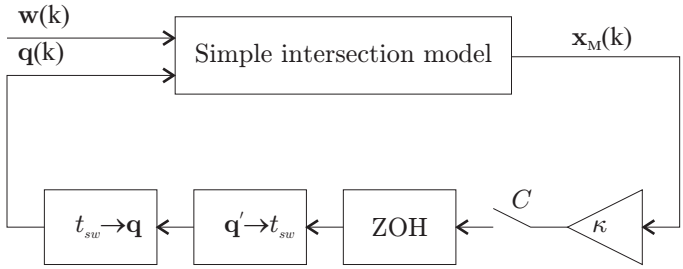


Fig. 2.4: LQR control loop

Assuming a control law in the form

$$\mathbf{q}'(k) = (q_1'(k), q_2'(k))^T = \kappa\ \mathbf{x_M}(k) \tag{2.15}$$

and solving the LQR Riccati equation gives the optimal feedback gain

$$\kappa = \begin{bmatrix} -0.001 & -0.020 & 0.000 & -0.012 \\ 0.001 & 0.016 & -0.001 & -0.062 \end{bmatrix}$$

This control law produces a potentially unbounded result $\mathbf{q}'(k)$, which cannot be directly applied to the intersection traffic control. Instead, from this result we compute the switching time t_{sw} as

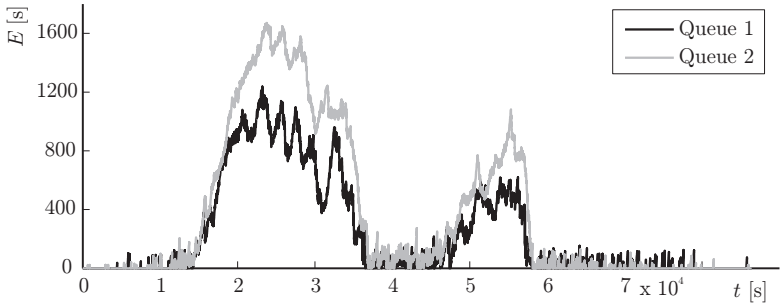
$$t_{sw}(k) = T\frac{q_1'(k)}{q_1'(k) + q_2'(k)} \tag{2.16}$$

The final control law $\mathbf{q}(k)$ is obtained by combining this expression and (2.12). The control law is computed at the start of the cycle time and is held for the whole cycle time. Control loop schema is shown on Fig. 2.4.

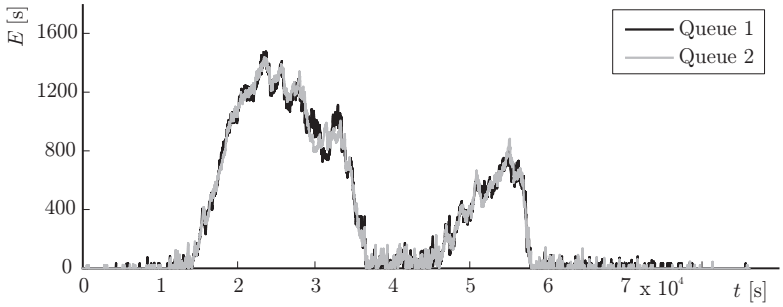
The LQR was applied to the simple intersection model control (2.9). The simulated response to real input data during one day (i.e. 86400 seconds) is shown in Fig. 2.5(a). The resulting average waiting times in the two queues are significantly different, the error caused by the linearization of the model.

2.4.2 Non-Linear Model Predictive Controller

Next, we consider controlling the waiting times in the simple intersection model using a non-linear model predictive controller (NMPC) [Find 02,



(a) LQR



(b) NMPC

Fig. 2.5: Mean waiting times

Magn 03]. The same cost function (2.13) was used as an objective function. For the NMPC algorithm we must select a control horizon and a prediction horizon. The *prediction horizon* T^{ph} is a time interval that the controller uses for simulating the nonlinear model. The *control horizon* is a time interval over which the controller optimizes the control signal. In our case, both horizons were set to the 90 s, which is equal to the cycle time C. Control loop schema is shown on Fig. 2.6.

For convex problems, the NMPC can find an optimal switching time t_{sw} by convex optimization [Boyd 04]. Our optimization problem is not convex, however. Instead, we find the optimal t_{sw} by enumerating all $t_{sw} \in (0, C)$ and simulating the response. The simulated intersection model response when applying the NMPC control law is shown on the Fig. 2.5(b).

NMPC allows tuning of the control law to be modified in a number of

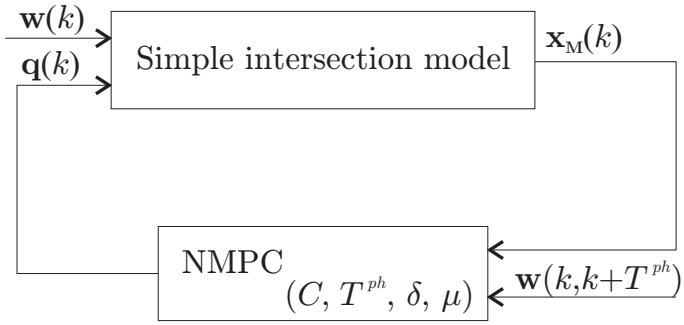


Fig. 2.6: NMPC control loop

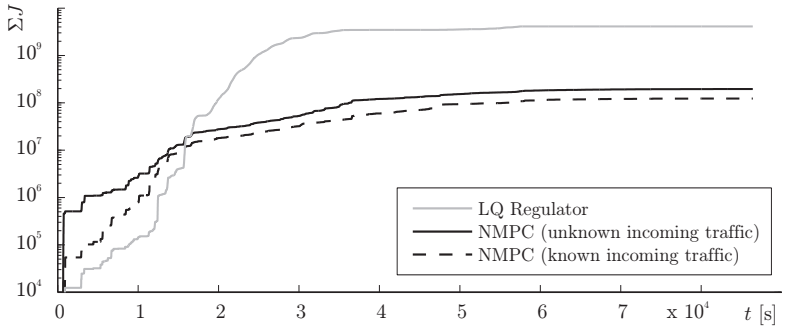


Fig. 2.7: Evolution of accumulated objective function

ways. For example, we can extend the controller by taking into account future incoming flow. In practice, we can measure this traffic in a previous, neighboring intersection (as shown by [Lei 01]) and forward this information to the next intersection controller. In this way, the predictive controller can prepare a much better control action. Trying this approach on the simple intersection model, adding feedforward traffic information to the NMPC is able to reduce the cost function J by about 37%. The accumulative value of the cost function J for different controllers is depicted in Fig. 2.7. We can see that the NMPC yields much better results than LQR, and that feedforward from the incoming traffic improves the result even further.

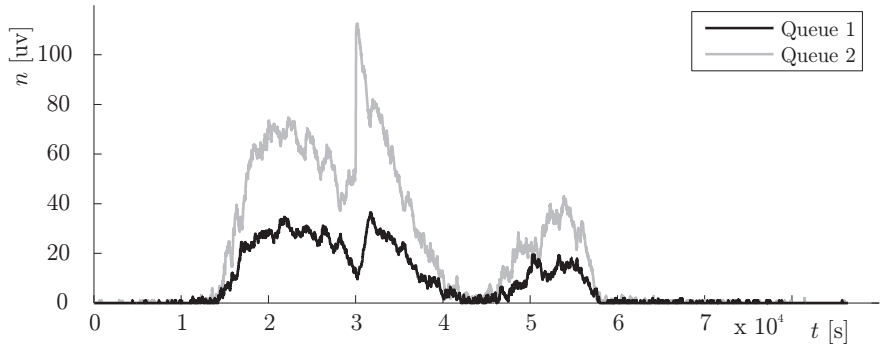
To evaluate the sensitivity of the NMPC to the incoming flow, the fol-

lowing experiment was performed. In addition to the original incoming flow, Queue 2 was subjected to one additional vehicle per second from time 30000 s to time 30100 s. Fig. 2.8(a) shows that the increase in the number of vehicles in Queue 1 is partially compensated by the NMPC, which leads to an increase in the number of vehicles in Queue 2. From the principle (see Equation 2.3) the number of vehicles in Fig. 2.8(a) holds for both models. The mean value of the waiting times in the extended queue model (shown in Fig. 2.8(b)) is used to calculate the switching time t_{sw} by the NMPC. The same t_{sw} is applied to the complete queue model (see Fig. 2.8(c)) showing that E in both queues is quite well balanced.

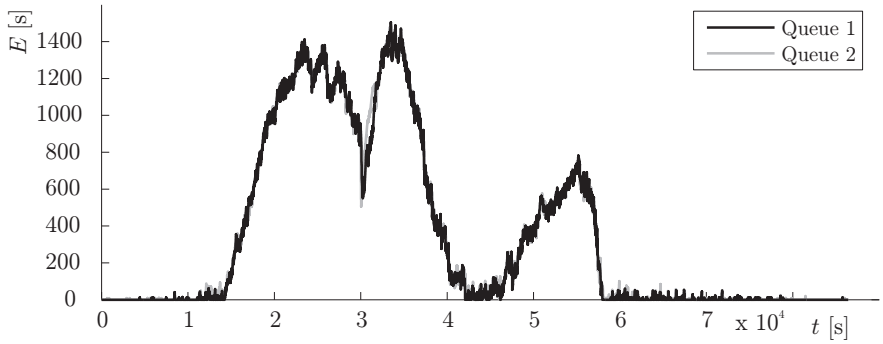
Table 2.2 shows the complexity of the NMPC calculations in terms of the number iterations for the one-day experiment. The left half of the table shows the complexity results for a prediction horizon of 90 s. The first row in the first column refers to the complexity of experiments reported up to now. In general, the control performance can be increased by prolongation of the predictive horizon. In the right half of the table, the complexity results for a predictive horizon of 180 s is shown. In all cases, the time complexity is negligible with respect to the cycle time. Nevertheless, we propose two simple approaches for reducing the problem complexity. First, the t_{sw} does not need to be an integer variable (as assumed in the first row in Table 2.2), but can be assumed to achieve a value divisible by 2 (the second row) or by 5 (the third row). Second, for practical reasons, $t_{sw}(k+1)$ does not need to vary from 0 to 90 s (as assumed in the columns with $\delta = 90$, $\mu = 0$), but could be allowed to vary only from $\max\{t_{sw}(k) - \delta, \mu\}$ to $\min\{t_{sw}(k) + \delta, 90 - \mu\}$ where δ stands for maximal allowed difference of t_{sw} and μ defines a minimal

Table 2.2: NMPC complexity in number of iterations

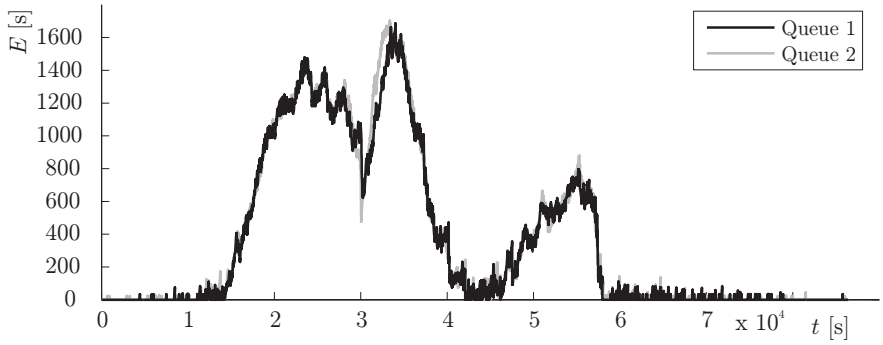
	predictive horizon 90 s			predictive horizon 180 s		
	$\delta = 90$ $\mu = 0$	$\delta = 30$ $\mu = 10$	$\delta = 5$ $\mu = 10$	$\delta = 90$ $\mu = 0$	$\delta = 30$ $\mu = 10$	$\delta = 5$ $\mu = 10$
1	**87451**	41328	9905	7958041	1980255	110621
2	45167	21927	5995	2122849	548138	39316
5	19220	10055	3587	384400	113237	13724



(a) Number of vehicles in the queue



(b) Mean waiting times in the extended queue model



(c) Mean waiting times in the complete queue model

Fig. 2.8: NMPC with extra vehicles at $t = 3 \cdot 10^4\, s$

phase split. Both approaches lead to a significant reduction of the search space for the NMPC.

2.5 Summary

In chapter, a new model for traffic queues has been presented. The extended queue model is described by the number of vehicles in the queue and the mean value of waiting time. The model is based on non-linear difference state equations. We have shown that the equilibrium point of the nonlinear model conforms with the Little's law.

Further, we have used the extended queue model to derive the parameters of the controllers for a simple intersection model with two queues. Two controllers were applied to the intersection model. First, we designed a linear quadratic regulator based on a linearization of the state equations around an equilibrium point. Second, we proposed the use of a nonlinear model predictive controller. The advantages and disadvantages of the controllers were discussed, and their performance was evaluated in simulations using real traffic data.

Chapter 3

General Light Controlled Intersection Model

This chapter proposes a light controlled intersection model based on a constant speed continuous Petri net. In contrast of previous chapter the continuous Petri net tool gives us more flexibility of modeling general intersections. The Model is innovative, firstly, by the free space modeling together with the opposite direction of the vehicular flow, secondly by the continuous Petri net use only leading to a smaller state space. For this purpose, we show a new method for conflict resolution in a continuous Petri net based on the maximal speed proportion. The model is compared with intersection model based on a classical discrete Petri net and it is evaluated in the simulation where real traffic data is used for the incoming flow.

3.1 Introduction

Traffic stream models consider traffic flow as a heterogeneous system which can be described from a macroscopic or microscopic point of view, [Tolb 05]. On one hand, the macroscopic point of view is described by global variables as the flow rate, the flow density and the flow average velocity, [Gazi 02]. On the other hand, microscopic point of view focuses on the individual vehicles behavior on the road.

The traffic flow model based on a discrete Petri net (PN), [Wang 93] is suitable for a microscopic description. On the other side, the model based on

a continuous Petri net is suitable for macroscopic description, [Tolb 01]. The combination of both concepts gives us model that consists of a continuous part in the street and a discrete part in the intersection. Hybrid Petri nets (HPN), [Davi 01], are used to describe these complex models, [Febb 04]. The main disadvantage of this concept based on HPN is a discrete part of the Petri net used for traffic flow, [Tolb 03] or a discrete part of the PN used for intersection control, [Febb 01], [Julv 05] which subsequently discretizes traffic flow. In these cases, the continuous flow is cut to the different flow levels by the intersection part, which is further processed by a continuous part of the HPN. Due to discrete part, processing such models is not efficient.

The aim of this chapter is to develop a new light controlled intersection model based on the *Constant speed Continuous Petri Net* (CCPN), [Davi 98]. This model divides the continuous traffic flow in consequence of the intersection structure and control without cutting the flow to different flow levels. Moreover, free space modelling together with the opposite direction of vehicle flow is considered. For this goal, we show a new method for conflict resolution in the CCPN based on maximal speed proportion. As a result, we show that the simulation based on our model is faster than based on the conventional approach.

3.2 Continuous Petri Net

A constant speed continuous Petri net is defined as a sextuple $R = \langle \mathcal{P}, \mathcal{T}, V, Pre, Post, M_0 \rangle$, where the definition of $\mathcal{P}$, $\mathcal{T}$, Pre, $Post$ are similar to those of the discrete Petri nets, as well as ${}^\circ T_i$, T_i°, ${}^\circ P_i$, P_i° notation for predecessor and successor places and transitions, both described, for example, by [Davi 04]. M_0 is the initial marking. $V : \mathcal{T} \to R_0^+ \cup \{\infty\}$ is a vector of maximal firing speeds. V_j denotes the *maximal speed* of transition T_j. Further more $v_j(t)$ denotes the *speed* of transition T_j at time t. The value of $v_j(t)$ is bounded by the interval $\langle 0, V_j \rangle$. Transition T_j is *strongly enabled* at time t if all places P_i of ${}^\circ T_j$ are marked. Place P_i is *supplied* at time t if there is at least one transition T_j in ${}^\circ P_i$, which is enabled (strongly or weakly). Transition T_j is *weakly enabled* at time t if there is place $P_i \in {}^\circ T_j$, which is not marked and is supplied, and the remaining places of ${}^\circ T_j$ are either marked or supplied. For more information, see [Hanz 03]. The following extensions of the CCPN will be used: the marking of continuous place can

take the value of 0+ and Inhibitor arcs there can be. Both of these concepts are described by [Davi 04].

3.2.1 Conflict Resolution

In a continuous Petri net, there is an actual conflict among k transitions in a set $\{T_1, T_2, \ldots, T_k\}$, if the speed of at least one of them has to be less than the maximal speed due to the speeds of the other transitions in this set.

When there is an actual conflict between two or more transitions, then the priority rule can be used. Another alternative is to use a conflict resolution method based on the *maximal speed proportion*. The maximal speed proportion means that the flow which runs through the place is divided into the subsequent transitions in proportion to their maximal speeds, if it is feasible in regard to other constraints. The method is based on a flow sharing rule, observing the *maximum firing speed* described by [Hanz 03] and can be used only in simple Petri nets. A simple Petri net is a net in which each transition can only be concerned with one conflict at most.

In order to illustrate the conflict resolution method we have shown an example in Fig. 3.1. There are three places P_1, P_2 and P_3 with zero marking. These places are supplied by the transitions T_1, T_2 and T_3 with a speed $v_1 = 35$, $v_2 = 40$ and $v_3 = 18$, respectively. Place P_2 is supplied with a smaller speed than the sum of the maximal speed of the transitions P_2° i.e. $v_2 < V_4 + V_5$. Therefore, there is an actual conflict between transitions T_4 and T_5.

A geometrical interpretation of the conflict resolution is given in Fig. 3.2.

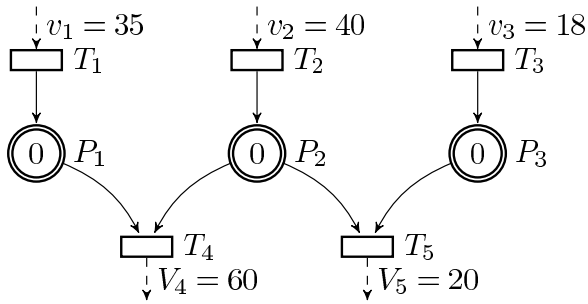


Fig. 3.1: Continuous Petri net example with actual conflict

The axes show the speeds of transitions T_4 and T_5. A gray polygon bounds the area where the possible speed v_4 and v_5 can be placed. This polygon is bounded by v_1, v_2 and v_3 from one side and by the axes from the other side. All combinations of the speeds v_4 and v_5 which are in proportion to the maximal speed V_4 and V_5 are located on the line λ. Point A, in Fig. 3.2(a), is where v_4 and v_5 are at a maximum and are located on line λ. The conflict resolution is given by this point A and the speed transitions for T_4 and T_5 are $v_4 = 30$ and $v_5 = 10$, respectively.

Let us consider the modification of speed v_1 from $v_1 = 35$ to $v_1 = 25$ (see Fig. 3.2(b)). This situation corresponds to point B where v_4 and v_5 are at a maximum and are located closest to line λ. The conflict resolution is given by this point and the speed of the transitions T_4 and T_5 is $v_4 = 25$ and $v_5 = 15$, respectively. Note that the speed is not in proportion to the maximal speed V_4 and V_5, but it is the maximum firing speed combination.

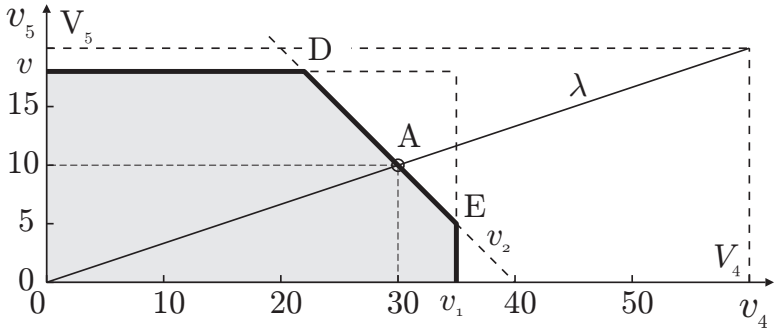
Let us assume v_1 to be equal to 15 (see Fig. 3.2(c)). This situation corresponds to point C where v_4 and v_5 are at a maximum and are located closest to line λ. The conflict resolution is given by this point and the speed of the transitions T_4 and T_5 is $v_4 = 15$ and $v_5 = 18$, respectively. Note that there is no actual conflict in this case.

We can generalize that the conflict resolution corresponds to the point located on the line segment DE closest to line λ. In the special case when point D is equal to E i.e. $|DE| = 0$ the transition speed is placed to this point, see Fig. 3.2(c) for point C.

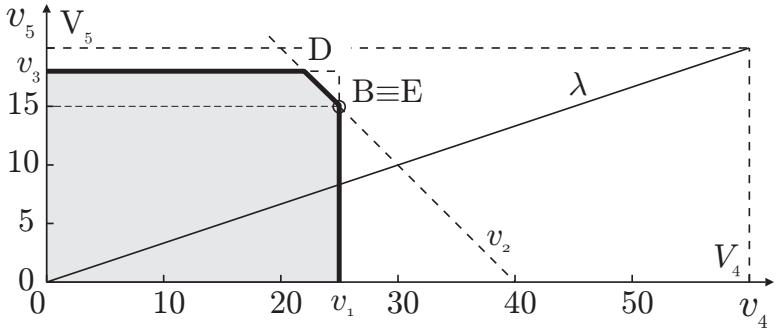
This geometrical interpretaion of speed computing can be formulated as follow: consider place P_i and the set of transitions P_i° having a conflict. The set of transitions P_i° indexes is denoted as $\mathcal{C}_i$. Variable $v_i^s(t)$ denotes the sum of the speeds supplying the place P_i at time t. For each transition T_j where $j \in \mathcal{C}_i$ initialize *maximum permissible transition speed* $v_j^{max}(t)$ as a minimum from the maximal speed V_j and separately sum the speed of transitions which supplied the place from ${}^\circ T_j$ with zero marking. Set the transition speed $v_j(t) = 0$ and modify it by the iterative Algorithm 3.1.

3.2.2 Linear Programming for Conflict Resolution

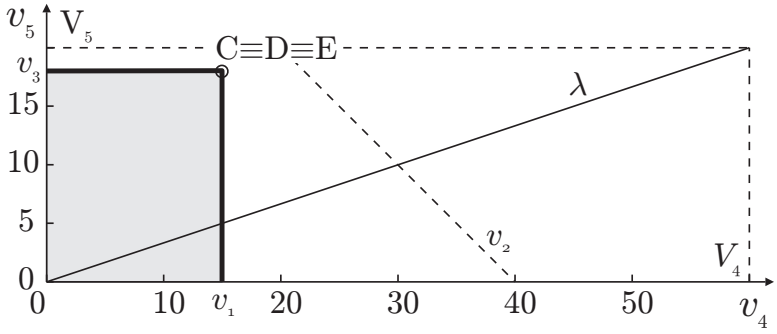
This subsection shows how linear programming (LP) can be used for conflict resolution computation. At first, we will use the example in Fig. 3.1 to show how a conflict resolution can be solved by linear programming at time t. In



(a) Proportional resolution



(b) Resolution close to proportion



(c) Resolution close to proportion without the transition T_2 effect

Fig. 3.2: Geometrical interpretation of the conflict resolution

Algorithm 3.1 Transition speed computing

while $v_i^s(t) > 0$ and $\mathcal{C}_i$ is not empty **do**
  $\mathcal{C}'_i = \mathcal{C}_i$
  for all j such that $j \in \mathcal{C}'_i$ **do**
    $v'_j(t) = \min\left(v_j^{max}(t) - v_j(t), v_i^s(t)\frac{V_j}{\sum_{k\in\mathcal{C}'_i} V_k}\right)$
    $v_j(t) = v_j(t) + v'_j(t)$
    if $v_j(t) = v_j^{max}(t)$ **then**
      $\mathcal{C}_i = \mathcal{C}_i \setminus j$
    end if
  end for
  $v_i^s(t) = v_i^s(t) - \sum_{j\in\mathcal{C}'_i} v'_j(t)$
end while

this example, we are looking for the speed of T_4 and T_5 with the maximal speed proportion condition. The LP problem using the variables defined above is:

$$\max\left(v_4(t) + v_5(t) - \epsilon\left|v_5(t) - v_4(t)\frac{V_5}{V_4}\right|\right), \tag{3.1}$$

subject to:

$$\begin{aligned} v_4(t) &\geq 0, \\ v_5(t) &\geq 0, \\ v_4(t) &\leq V_4, \\ v_5(t) &\leq V_5, \\ v_4(t) &\leq v_1(t), \\ v_5(t) &\leq v_3(t), \\ v_4(t) + v_5(t) &\leq v_2(t), \end{aligned} \tag{3.2}$$

where ϵ is a small constant for which $0 < \epsilon < \min(1, \frac{V_4}{V_5})$ holds. The fraction $\frac{V_5}{V_4}$ is the slope of line λ in Fig. 3.2.

The objective function (3.1) can be rewritten with the use of a new variable. This variable z_{45} replaces the absolute value, which includes the v_4 and v_5 variables as follows:

$$\max\left(v_4(t) + v_5(t) - \epsilon z_{45}\right), \tag{3.3}$$

on the assumption that we add two new constraints:

$$\begin{array}{rcl} v_5(t) - v_4(t)\frac{V_5}{V_4} & \leq & z_{45}, \\ v_5(t) - v_4(t)\frac{V_5}{V_4} & \geq & -z_{45}. \end{array} \tag{3.4}$$

This LP problem gives us a solution in conformity of Fig. 3.2.

Moreover, if there is a conflict among more transitions T_j where $j \in \mathcal{C}_i$ then it can be formulated as an LP problem as follows:

$$\max \left(\sum_{j \in \mathcal{C}_i} v_j(t) - \epsilon \sum_{\substack{k,l \in \mathcal{C}_i \\ k<l}} z_{kl} \right), \tag{3.5}$$

subject to:

$$\begin{array}{rcll} v_j(t) & \geq & 0, & \forall j \in \mathcal{C}_i \\ v_j(t) & \leq & v_j^{max}(t), & \forall j \in \mathcal{C}_i \\ \sum\limits_{j \in \mathcal{C}_i} v_j(t) & \leq & v_i^s(t), & \\ v_l(t) - v_k(t)\frac{V_l}{V_k} & \leq & z_{kl}, & \forall k,l \in \mathcal{C}_i, k < l \\ v_l(t) - v_k(t)\frac{V_l}{V_k} & \geq & -z_{kl}, & \forall k,l \in \mathcal{C}_i, k < l. \end{array} \tag{3.6}$$

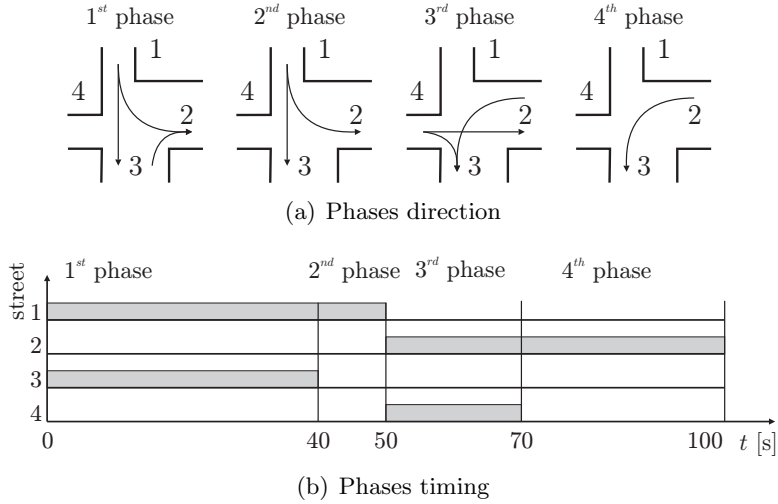
where ϵ is a small constant number which $0 < \epsilon < \min\limits_{\substack{k,l \in \mathcal{C}_i \\ k<l}} (1, \frac{V_k}{V_l})$ holds.

The solution of the LP problem mentioned above gives us the same results as Algorithm 3.1. Nevertheless, it is not intended for all continuous Petri nets generally. It can be used only for a CCPN where the number of subsequent transitions which are under consideration for the objective function (3.5) for each P_i° are the same, which is our case.

3.3 Light Controlled Intersection Model

In this section we will study the light controlled intersection. The intersection under consideration (see Fig. 3.3(a)) includes four input and two output flows.

The light controlled intersection is characterized by several parameters: the number of phases, the phases split and a list of streets from which the vehicles flow. Our model considers a constant sum of the phase time intervals,



(a) Phases direction

(b) Phases timing

Fig. 3.3: Intersection description

i.e. a constant *cycle time* C in seconds. The next parameter is the *distribution rate*. The distribution rate $\alpha_{s\to d}$ indicates the proportion of the vehicles which are crossing the intersection from the source street s to the destination street d, where the variables s and d denote the indexes of the street numbers. The sum of the distribution rates for each input street must be equal to one. For each distribution rate, we consider the *real vehicle speed* $w_{s\to d}$ in kilometers per hour. Further more, $t_{s\to d}$ denotes the *duration* how long the vehicles flow with the considered distribution rate.

The intersection under consideration includes four control phases. The source and destination streets are marked in Fig. 3.3(a). The duration of the vehicle flow through the intersection from a street can be found in Fig. 3.3(b). For this intersection, the cycle time C is assumed to be 100 s. The summary of all the parameters under consideration is presented in the left part of Table 3.1.

We can see that some of the source-destination pairs are there twice differing in duration $t_{s\to d}$. This difference is given by a give way rule in addition to the light control. For example, see Fig 3.3(a) where, during the 3rd phase the flow from street 2 gives way to the flow from the street 4. If there is a flow from street 4 then the duration is equal to 30 s, otherwise it is

Table 3.1: Intersection parameters

s	d	$\alpha_{s\to d}$	$w_{s\to d}$ [m·s^{-1}]	$t_{s\to d}$ [s]	$d_{s\to d}$ [s/uv]	$V_{s\to d}$ [uv/s]	$V_{s\to}$ [uv/s]
1	2	0.2	8.3	50	0.60	0.83	1.23
1	3	0.8	13.9	50	0.36	1.39	
1	2	0.2	8.3	10	0.60	0.17	0.25
1	3	0.8	13.9	10	0.36	0.28	
2	3	1	8.3	50	0.60	0.83	0.83
2	3	1	8.3	30	0.60	0.50	0.50
3	2	1	5.6	40	0.90	0.44	0.44
4	2	0.4	13.9	20	0.36	0.56	0.29
4	3	0.6	5.6	20	0.90	0.22	

equal to 50 s.

From these intersection parameters, we can compute the parameters necessary for the continuous Petri net. The first one is the delay $d_{s\to d}$, which denotes the time for which one vehicle flows throw the intersection in seconds per unit vehicle:

$$d_{s\to d} = \frac{l_{\text{uv}}}{w_{s\to d}}. \tag{3.7}$$

Where the constant l_{uv} is the length of the unit vehicle including the distance between the vehicles in meters. In this paper, we will consider l_{uv} to be equal to 5 m. The second parameter is the maximum average speed $V_{s\to d}$ over the period T in unit vehicles per second:

$$V_{s\to d} = \frac{w_{s\to d} t_{s\to d}}{l_{\text{uv}} C}. \tag{3.8}$$

The speeds with a common source can be combined with respect to the distribution rate to one common speed as follows:

$$V_{s\to} = \frac{\sum\limits_{d} \alpha_{s\to d}}{\sum\limits_{d} \frac{\alpha_{s\to d}}{V_{s\to d}}}. \tag{3.9}$$

The summary of all these parameters is presented in the right part of Table 3.1.

3.3.1 Continuous Petri Net Intersection Model

The continuous Petri net intersection model described in Fig. 3.3 is shown in Fig. 3.4. The net is divided to six parts. First, part A includes eight places, which are used as an interface for the inputs into the intersection from the streets. For each street, there are two places, the first one for vehicles waiting in the queue before the intersection (P_{101}, P_{102}, P_{103}, P_{104}), the second one for the free space in the street (P_{151}, P_{152}, P_{153}, P_{154}). The *free space* denotes the tokens which represent the available space for the vehicles, which can flow into the street. The free space modelling together with the opposite direction of vehicular flow is an innovative approach to this model. The distribution rate through the intersection is shown in parts B, C, D and E, each one for one input street. These parts of the network are very similar, thus only part C will be described. The transition T_{40} consumes vehicles from street 4 and the free space is returned back to the place P_{154}. Vehicles are divided up into distribution rate acceptance, to the places P_{40} and P_{43}. The weights of the arcs are equal to the distribution rate $\alpha_{4\to2}$ and $\alpha_{4\to3}$. From places P_{40} and P_{43}, the vehicles flow through the transitions T_{42} and T_{44} to the output streets represented by places P_{202} and P_{203}, respectively. The free space from the output streets close the loop l_1 (l_2) and flow to the place P_{154} through transition T_{40}. The speed of all the transitions in the loop is the same. It is bounded by maximal speed which is taken from Table 3.1, i.e. for example, the maximal speed V_{42} and V_{43} are equal to $V_{4\to2}$.

Transitions T_{12}, T_{17}, T_{32}, T_{42} and T_{14}, T_{19}, T_{20}, T_{21}, T_{44}, respectively are in conflict (places P_{252} and P_{253}). This conflict is solved by the above described (see Subsection 3.2.1) conflict resolution method based on maximal speed proportion. This method guarantees the same free space distribution as in a real traffic intersection.

Parts D and E include two parallel similar sub networks. Input transitions to those sub networks T_{10}, T_{15} and T_{20}, T_{21}, respectively, are in conflict (places P_{101} and P_{102}). This conflict is structural but not effective, because only one of them may be fired at the same moment. It is given by the arcs and the inhibitor arcs between places P_{103} and P_{104} respectively and the mentioned transitions.

See Fig. 3.3(a) there is no flow into street 2 during the 4^{th} phase. During this time the free space is accumulated in place P_{252}. Thirty percent (split of the 4^{th} phase in the cycle time C) of the free space is saved to the temporary

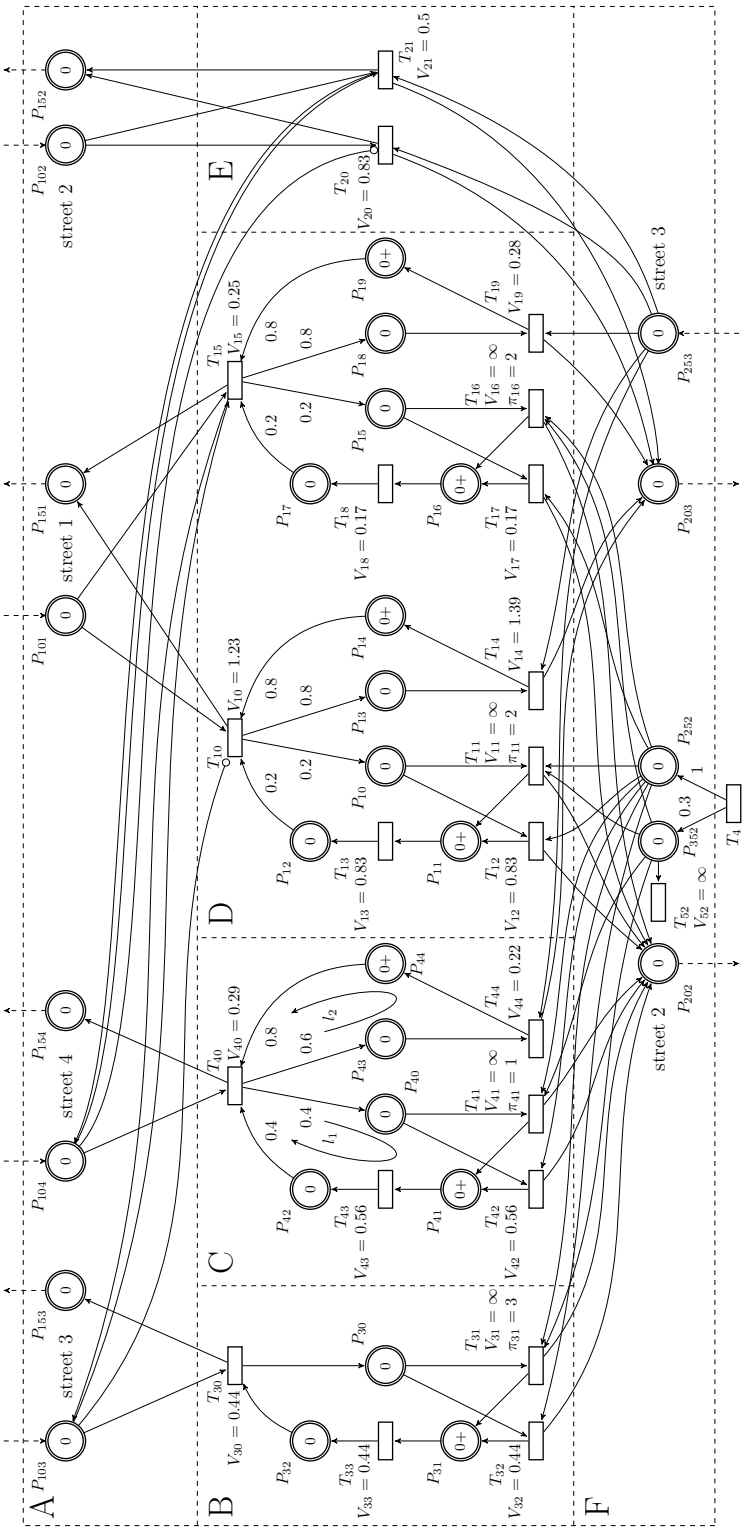


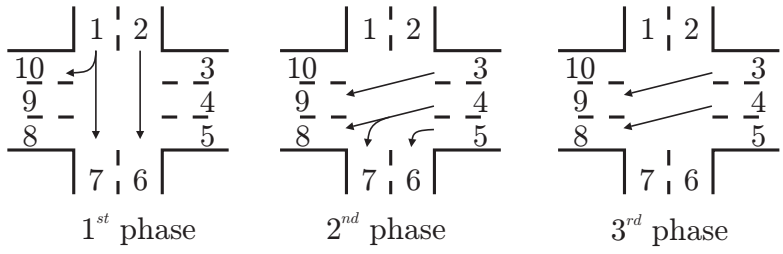
Fig. 3.4: Light controled intersection continuous Petri net model

place P_{352}, by the arc from T_4 to P_{352} with weight equal to 0.3. Transition T_4 is a part of the street model. This free space must be utilized during the next traffic phase. The order of the phases is given (see Fig. 3.3(a)) and together with the give way rule it designates the priority (π_i) for transitions T_{11}, T_{16}, T_{31}, T_{41}. Those transitions consume marking from places P_{352} and P_{252} and the conflict is solved by the priority.

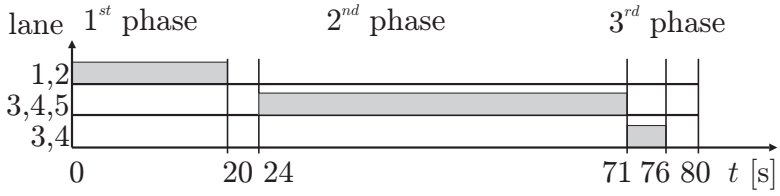
3.4 Performance Evaluation

In this section the performance evaluation of the previously described intersection model simulation is shown. Prague intersection of "V Botanice" and "Zborovská" streets (GPS location: 50°4′31.484″N, 14°24′27.36″E) is used as a real system for the simulation. The street "V Botanice" includes three traffic lanes and the street "Zborovská" includes two traffic lanes. The intersection scheme and control phases are shown in Fig. 3.5.

The real data for the performance evaluation of the simulation is measured by the inductive loop detectors. Inductive loop detectors monitor the traffic



(a) Phases direction



(b) Phases timing

Fig. 3.5: Prague intersection diagram

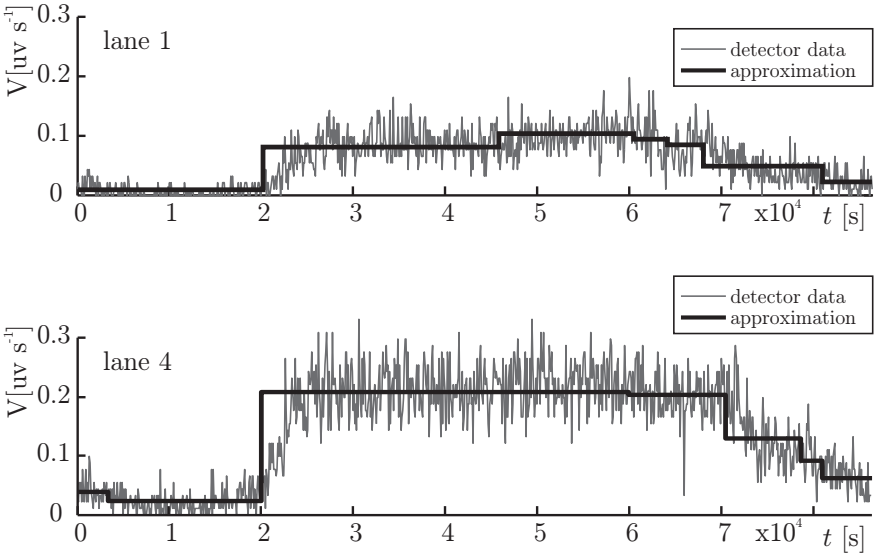


Fig. 3.6: Real data output from the inductive loop detectors

conditions on the road for each lane separately[1]. Data from the detectors is produced as time-averaged over a 90 s long period. See Fig. 3.6 for an example of the data from the detectors in unit vehicles per second during a one day period for lane numbers 1 and 4 (other lane detectors produce a similar output). This data from the detectors was subsequently processed by a step-wise filter. The step-wise filter is based on smoothing and produces a constant level of the output data during the variable time interval (see the "approximation" line in Fig. 3.6). Data from this filter is used as the input for the simulation during one day.

The intersection model uses the same principle described in Section 3.3. The parameters are summarized in Table 3.2. The places which are intended for *vehicles waiting in the queue before the intersection* (see to part A in Fig. 3.4) are connected by the arc from the new *source transitions*. Each source transition is connected to just one place. The maximal speed of the source transition is set to the value given by the approximation for the cor-

[1]The detectors distance from the intersection is 35 m for the input lanes and 50 m for the output lanes. This is a sufficient distance to minimize the effect of the intersection control to continuity flow.

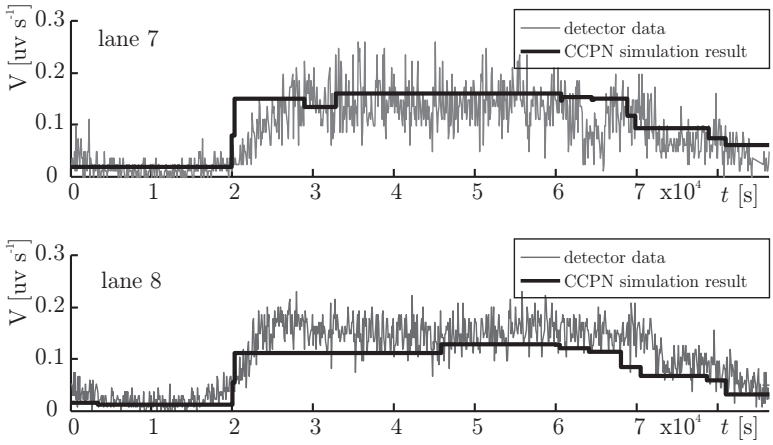
Table 3.2: Intersection parameters

s	d	$\alpha_{s\to d}$	$w_{s\to d}$ [m·s^{-1}]	$t_{s\to d}$ [s]	$d_{s\to d}$ [s/uv]	$V_{s\to d}$ [uv/s]	$V_{s\to}$ [uv/s]
1	10	0.21	7.2	20	0.69	0.36	0.58
1	7	0.79	13.9	20	0.36	0.69	
2	6	1.0	13.9	20	0.36	0.69	0.69
3	9	1.0	13.9	52	0.36	1.81	1.81
4	8	0.77	13.9	52	0.36	1.81	1.81
4	7	0.23	7.8	47	0.64	0.91	
5	6	1	7.8	47	0.64	0.91	0.91

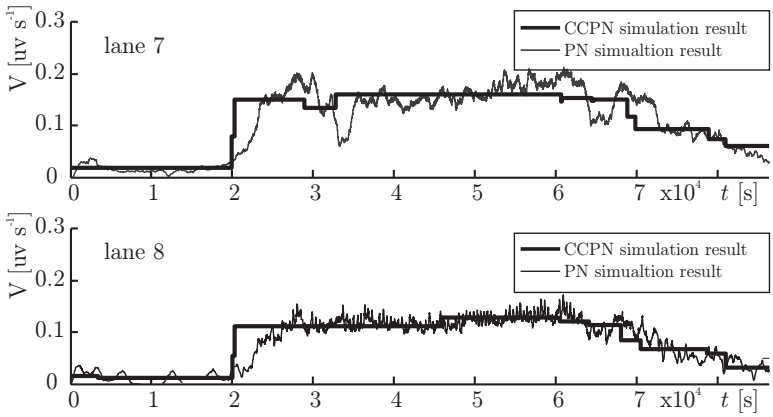
respondent lane. Thus, the intersection places are supplied by the speed which is equivalent to the real data measured in the streets. Note that the data from the step-wise filter is not constant during the whole simulation time. From this point of view, the simulation time must be split into several shorter intervals during which the data is constant. These intervals are simulated separately and the simulation result of the first interval is used as an initial condition for the second interval and so on for the whole simulation time.

The result of the simulation is shown in Fig. 3.7. There are two representative outputs from lanes 7 and 8 in the unit vehicles per second. The simulation results for the other lanes are similar, hence they are not shown. Fig. 3.7(a) shows the real data from the detector (thin line) versus the result of the CCPN model simulation (thick line). Fig. 3.7(b) shows the result of the CCPN model simulation versus the result of the discrete PN model (thin line). The simulation based on the discrete PN uses the same real data which is used for preprocessing by the step-wise filter described above. The parameters of both continuous Petri net models are the same. The delay time for the discrete PN model is taken from the column $d_{s\to d}$ of Table 3.2. The figures show that the simulation model corresponds to the real system.

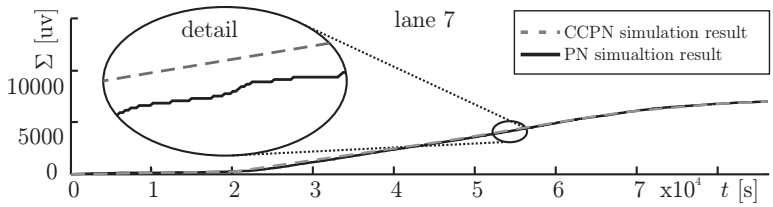
The quality of the simulation can be analyzed from Fig. 3.7(c), where the sum of all the vehicles in lane 7 during the simulation time is shown. We can see that the number of vehicles which flow through the intersection is similar for both models. The model based on the CCPN computes its state continuously in contrast to the discrete PN model where each vehicle in the



(a) Real data from detector and the CCPN simulation



(b) Data from the CCPN simulation and the discrete PN simulation



(c) Sum of all the vehicles during the simulation time

Fig. 3.7: Prague intersection simulation results

intersection initiates the state update, implies the computation performance. The total real simulation time for the model based on the discrete PN is 4927 s and the model uses 149523 states from the state space. On the other side, the total real simulation time for the model based on the CCPN is 2 s and model uses 34 states. We used standard PC (Intel Core2 CPU T7200 @2.00 GHz, 4GB RAM) for the simulations.

3.5 Summary

In the chapter, a new method for conflict resolution in CCPN has been presented. The conflict resolution method is based on the maximal speed proportion. Next, the iterative algorithm for this problem and its solution by LP was shown. LP gives us an effective method to solve complex continuous Petri net models.

Furthermore, the described method for conflict resolution was used in a light controlled intersection model based on CCPN. The model describes the traffic flow from the macroscopic point of view. This model is innovative, firstly, by the free space modeling together with the opposite direction of the vehicular flow. Secondly, by the constant speed continuous Petri net use only, i.e. without a discrete part for intersection control. The performance evaluation shows that the real time of the simulation is much better than for an equivalent model based on the discrete PN. The accuracy of the simulation results was successfuly compared with real data from traffic in Prague.

Chapter 4

TORSCHE Scheduling Toolbox for Matlab

This chapter presents a Matlab based Scheduling toolbox TORSCHE (Time Optimization of Resources, SCHEduling) intended mainly as an open source research tool to handle with optimization and scheduling problems. The toolbox offers a collection of data structures that allow the user to formalize various scheduling problems. The toolbox algorithms can utilize interfaces for Integer Linear Programming or the satisfiability of boolean expression problem solvers. With respect to the close relationship between the scheduling and graph theory, the toolbox provides graph algorithms with uniform interface.

4.1 Introduction

TORSCHE (Time Optimization of Resources, SCHEduling) Scheduling Toolbox for Matlab is a freely (GNU GPL) available toolbox developed at the Czech Technical University in Prague. The toolbox is designed for researches in operational research or industrial engineering and for undergraduate courses. The current version of the toolbox covers the following areas: scheduling on monoprocessor/dedicated processors/parallel processors, open shop/flow shop/job shop scheduling, cyclic scheduling and real-time scheduling. Furthermore, particular attention is dedicated to graphs and graph algorithms due to their large interconnection with the scheduling theory. The

toolbox offers a transparent representation of the scheduling/graph problems, various scheduling/graph algorithms, a useful graphical editor of graphs, interfaces for mathematical solvers (Integer Linear Programming, satisfiability of the boolean expression) and an interface to a MATLAB/Simulink based simulator and a visualization tool. The scheduling problems and algorithms are categorized by notation $(\alpha|\beta|\gamma)$ proposed by Graham and Błażewicz [Blaz 83]. This notation, widely used in the scheduling community, greatly facilitates the presentation and discussion of scheduling problems.

The toolbox is supplemented by several examples of real applications, e.g. the scheduling of Digital Signal Processing (DSP) algorithms on a hardware architecture with pipelined arithmetic units, scheduling the movements of hoists in a manufacturing environment, scheduling of light controlled intersections in urban traffic and response-time analysis in real-time systems. The toolbox is equipped with sets of benchmarks from the research community (e.g. DSP algorithms, the Quadratic Assignment Problem). TORSCHE is an open source tool available at (http://rtime.felk.cvut.cz/scheduling-toolbox/)

In the off-line scheduling area, some tools for the development of scheduling algorithms already exist. The term *off-line* scheduling means all parameters of the scheduling problem are known a priori [Pine 08]. A scheduling system developed at the Stern School of Business is called LEKIN [Pine 02]. It was created as an educational tool and it provides six basic workspace environments: single machine, parallel machines, flow shop, flexible flow shop, job shop, and flexible job shop. Another tool is LiSA [Andr 03]. It is a software-package for entering, editing and solving off-line scheduling problems while the main focus is on shop-scheduling and one-machine problems. The graphical user interface is written in Tcl/Tk for machine and operating system independence. All algorithms are implemented externally while the parameters are passed through files. The commercial tool ILOG Scheduler from the software package ILOG CP Optimizer [ILOG 09] is based on constraints programming. It provides extensions for scheduling problems in manufacturing, transportation and workforce scheduling.

There are more tools for on-line scheduling, where *on-line* means the parameters of the tasks become known on the task arrival/occur. One example is the MAST tool [Gonz 08] built to mainly support the timing analysis of real-time applications. Close to the on-line scheduling tools

are tools for real-time scheduling. For example TrueTime [Ande 05] is a Matlab/Simulink-based simulator for real-time control systems. TrueTime facilitates co-simulation of controller task execution in real-time kernels, network transmissions, and continuous plant dynamics.

4.2 Tool Architecture and Basic Notation

TORSCHE Scheduling Toolbox is written in Matlab object oriented programming language (backward compatible with Matlab environment version 2007) and it is used in the Matlab environment as a toolbox. The toolbox includes two complementary parts. The first one is intended for solving problems from the scheduling theory. Problems from this area or their parts can, very often, be reformulated to another problem which can be directly solved by a graph algorithm. For this purpose the second part of the toolbox is dedicated to the graph theory algorithms.

4.2.1 Scheduling Part

The main classes of scheduling part are *Task*, *PTask*, *Taskset* and *Problem*. The UML class diagram with relationships in it is shown in Fig. 4.1. A task represented by the class of the same name is a unit of work to be scheduled on the given set of processors. The class includes task parameters as processing time, release date, deadline, etc. The instance of the class (variable `T1` depicted below) is returned by the constructor method with the following syntax rule:

```
T1 = task([Name,]ProcTime[,ReleaseTime[,Deadline ...
          [,DueDate[,Weight[,Processor]]]]])
```

Input variables correspond to the public class properties. Variables contained inside the square brackets are optional. The class *Task* provides the following properties (also graphically depicted in Fig. 4.2):

Processing time (`ProcTime`, p_j) is time necessary for task execution (also called computation time).

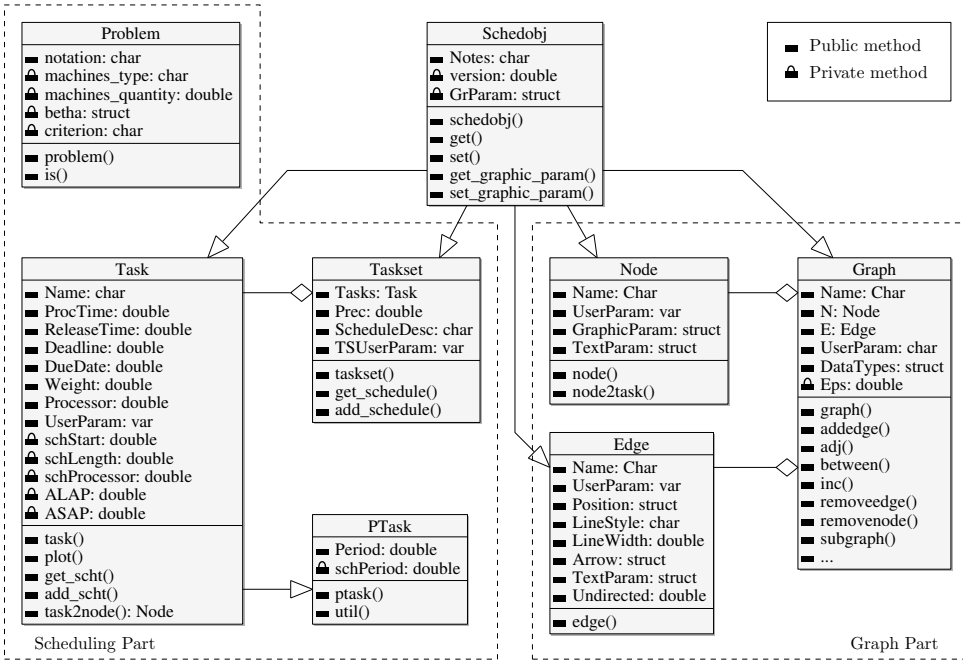


Fig. 4.1: TORSCHE architecture by the UML Class Diagram

Release date (`ReleaseTime`, r_j) is the moment at which a task becomes ready for execution (also called the arrival time, ready time, request time).

Deadline (`Deadline`, $\widetilde{d}_j$) specifies a time limit by which the task has to be completed, otherwise the scheduling is assumed to fail.

Due date (`Duedate`, d_j) specifies a time limit by which the task should be completed, otherwise the criterion function is charged by a penalty.

Weight (`Weight`) expresses the priority of the task with respect to other tasks (also called the priority).

Processor (`Processor`) specifies the dedicated processors on which the task must be executed.

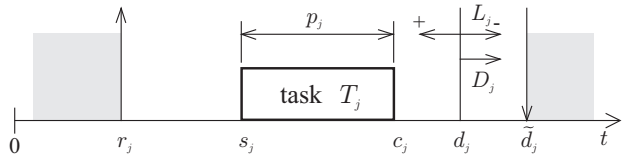


Fig. 4.2: Graphics representation of task parameters

The resulting schedule is represented by the following properties:

Start time (`schStart`, s_j) is the time when the execution of the task is started.

Completion time (c_j) is the time when the execution of the task is finished.

Lateness $L_j = c_j - d_j$.

Tardiness $D_j = \max\{c_j - d_j, 0\}$.

The private properties are mainly intended for the final task schedule representation, which are set-up inside the scheduling algorithms (e.g. by method `add_scht`). The values from the private properties are used, for example, by the method `plot` for the Gantt chart drawing.

Class *PTask* (see Fig. 4.1) is a derived class from the *Task* class in order to represent a periodic task in the on-line scheduling problems (e.g. in response-time analysis). This class extends the *Task* class with support to store, plot and analyze the utilization methods.

The instances of the classes *Task* and *PTask* can be aggregated into a set of tasks. A set of tasks is represented by the class *Taskset* which can be obtained as the return value of the constructor `taskset`, for example:

```
TS = taskset(tasks[,prec])
```

where the variable `tasks` is an array of instances of the *Task* class. Furthermore, the relations between the tasks can be defined by precedence constraints in the optional parameter `prec`. The parameter `prec` is an adjacency matrix defining a graph where the nodes correspond to the tasks and the

edges are precedence constraints between these tasks. For simple scheduling problems, the object *Taskset* can be directly created from a vector of the tasks processing times. In this case, the tasks are created automatically inside the object constructor. There are also other ways how to create an instance of the set of tasks in order to simplify the user interface as much as possible.

Another class, *Problem*, is used for the classification of deterministic scheduling problems in Graham and Błażewicz notation [Blaz 83]. This notation consists of three parts $(\alpha|\beta|\gamma)$. The first part describes the processor environment (e.g. number and type of processors), the second part describes the task characteristics of the scheduling problem (e.g. precedence constrains, release time). The last part denotes the optimality criterion (e.g. schedule makespan minimization). The following example shows the notation string used as an input to the class constructor:

```
prob = problem('P|prec|Cmax')
```

This instance of the class *Problem* represents the scheduling problem on parallel identical processors where the tasks have precedence constraints and the objective is to minimize the schedule makespan.

All of the above mentioned classes are designed to be maximally effective for users and developers of scheduling algorithms. The toolbox includes dozens of scheduling algorithmsscheduling algorithm which are stored as Matlab functions with at least two input parameters and at least one output parameter. The first input parameter has to be an instance of the *Taskset* class containing the tasks to be scheduled. The second one has to be an instance of the *Problem* class describing the required scheduling problem. The output parameter is an instance of the *Taskset* class containing the resulting schedule. A typical syntax of the scheduling algorithm call is:

```
TSout = algorithmname(TS,problem[,processors[,parameters]])
```

where:

`TSout` is the instance of the *Taskset* with the resulting schedule,
`algorithmname` is the algorithm name,
`TS` is the instance of the *Taskset* to be scheduled,

`problem` is the instance of the *Problem* class,
`processors` is the number of processors to be used,
`parameters` denotes additional parameters, e.g. algorithm strategy, etc.

The typical work-flow of scheduling problem solution is shown on an UML Interaction Overview Diagram (see Fig. 4.3). There are several sequence diagrams (sd) used. The first two "Create Taskset 1" and "Create Taskset 2" show the constitution of a *Taskset* instance by both of the above described ways. The third one, called "Classification", shows the constitution of a *Problem* instance. The following sequence diagram "Scheduling" presents the call of the scheduling algorithm. The scheduling algorithm is described separately in the "Scheduling Algorithm" diagram, which is divided into three parts. The first one is checking of the input parameters ("Read Properties"). The second one is constituted by the solver of a scheduling algorithm and the final part stores the resulting schedule into the instance of the *Taskset* ("Schedule to the Tasks"). The last diagram "Gantt Chart" presents the final schedule conversion to a Gantt chart, i.e. the graphical representation of a schedule.

Furthermore, the toolbox contains objects to handle problems like open shop, flow shop and job shop, it also supports limited buffers and transport robots. For more details please see the toolbox documentation [Kuti 07].

4.2.2 Graph Part

A lot of scheduling problems can be solved with the assistance of the graph theory. Therefore, the second part of the toolbox is aimed at graph theory algorithms. All algorithms are available as a method of the main class *Graph* which is used to describe the directed graph.

There are several different ways to create an instance of the class *Graph*. The graph is generally described by an adjacency matrix. In this case, the *Graph* object is created by the command with the following syntax:

```
G = graph('adj',A)
```

where the variable A is an adjacency matrix. Similarly, it is possible to create the *Graph* by an incidence matrix. Another way how to create the *Graph* object is based on a matrix of edge weights.

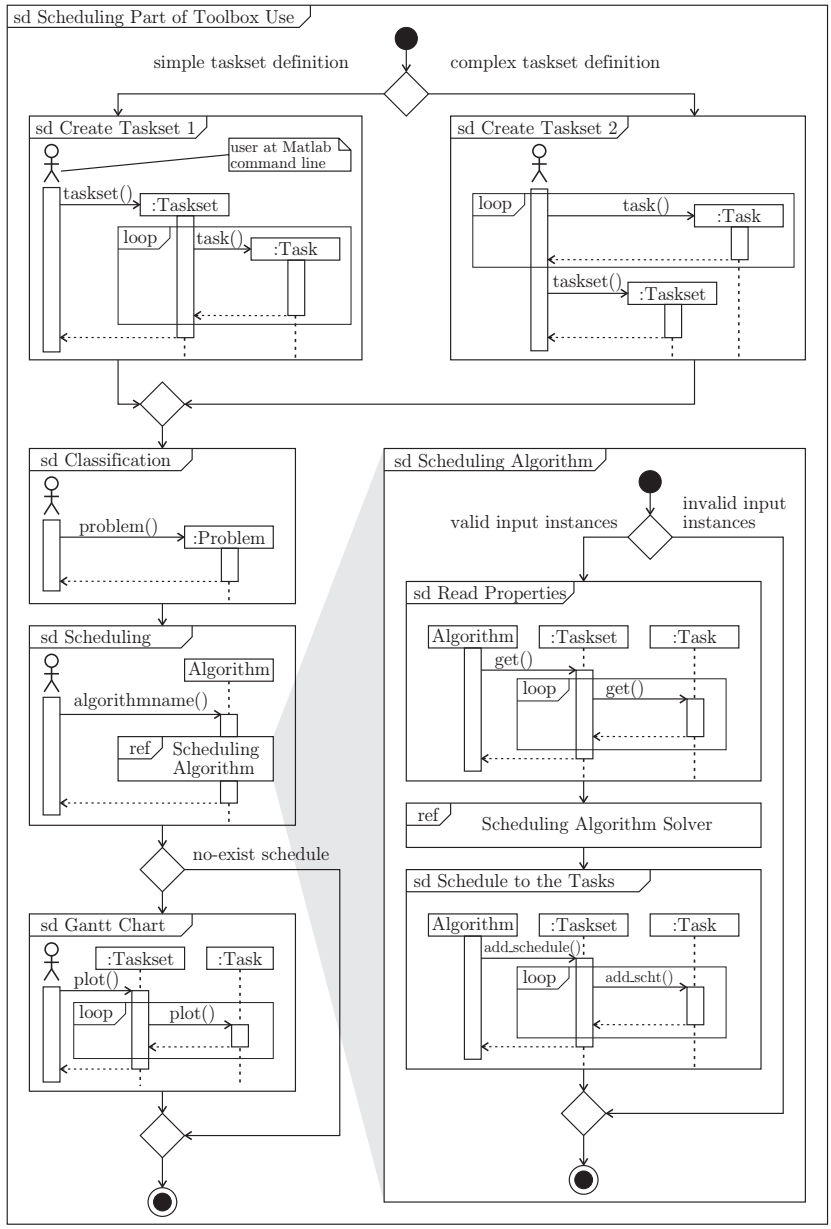


Fig. 4.3: UML Interaction Overview Diagram of a typical toolbox work-flow of the scheduling problem solution

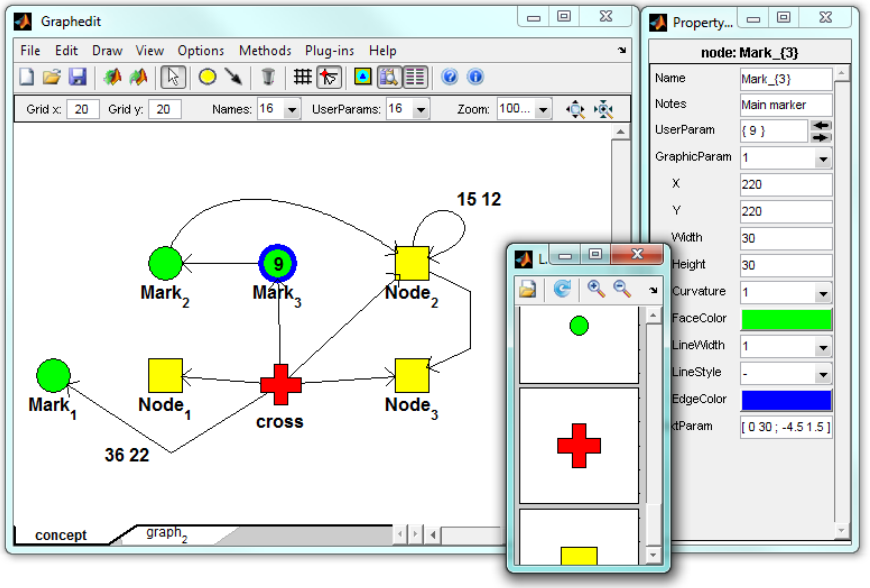


Fig. 4.4: Graphedit (main window, library browser, property editor)

The toolbox is equipped with a simple but powerful editor of graphs called *Graphedit* based on the System Handle Graphics of Matlab. It allows one to construct directed graphs with various user parameters on nodes and edges by simple and an intuitive way. Fig. 4.4 shows the *Graphedit* with a graph with various graphical representations of nodes, various edge paths and two setting windows: property editor and node library. The constructed graph can be easily used to create an instance of the class *Graph* which can be exported to the Matlab workspace or saved to a binary mat-file. In addition, *Graphedit* contains a system of plug-ins which allow one to extend its functionality by the user.

Moreover, due to the close relationship between the scheduling and graph algorithms, each object *Graph* can be transformed to an object *Taskset* and vice versa. Obviously, the nodes from the graph are transformed to the tasks in the *Taskset* and the edges are transformed to the precedence constraints and vice versa according to the user specification.

4.3 Implemented Algorithm

The biggest part of the toolbox is constituted by scheduling algorithms. There are a large variety of algorithms (see Appendix A) solving both simple problems (on a single processor) and practically oriented issues. This section shows several of them demonstrated on real applications.

4.3.1 List Scheduling Algorithm

List Scheduling (LS) is a basic and popular combinatorial algorithm intended for scheduling of set of tasks on a single and parallel processors. In this algorithm tasks are fed from a pre-specified list and, whenever a processor becomes idle, the first available task on the list is scheduled and removed from the list. Where the availability of a task means that the task has been released and if there are precedence constraints, all its predecessors have already been processed [Leun 04]. The algorithm terminates when all tasks from the list are scheduled. Its time complexity is $\mathcal{O}(n)$. In multiprocessor case, the processor with minimal time is taken in every iteration of algorithm and the others are relaxed.

The fact, which is obvious from the principle of algorithm is that, there aren't any requirements of knowledge about past or future of content of the list. Therefore, this algorithm is capable to solve offline as well as online no-clairvoyance scheduling problems. There are many strategy algorithms like the Earliest Starting Time first, the Earliest Completion Time first, the Longest Processing Time first and etc., based on simple ordering of tasks in the list by various parameters.

Algorithm strategy do not guarantee to find optimal solutions for any instance of an optimization problem. On condition of appropriate choose of strategy it often provide acceptable results with very good time and memory complexity.

The Earliest Starting Time first (EST) is a strategy in which the tasks are arranged in nondecreasing order of release time r_j before the application of List Scheduling algorithm. The time complexity of EST is $\mathcal{O}(n \cdot log(n))$.

The Earliest Completion Time first (ECT) is a strategy in which the tasks are arranged in nondecreasing order of completion time C_j in each iteration of List Scheduling algorithm. The time complexity of ECT is $O\left(n^2 \cdot log(n)\right)$.

The Longest Processing Time first (LPT) is a strategy in which the

tasks are arranged in non-increasing order of processing time p_j before the application of List Scheduling algorithm. The time complexity of LPT is $\mathcal{O}(n \cdot log(n))$.

The Shortest Processing Time first (SPT) is a strategy in which the tasks are arranged in non-decreasing order of processing time p_j before the application of List Scheduling algorithm. The time complexity of SPT is $\mathcal{O}(n \cdot log(n))$.

List Scheduling is implemented in TORSCHE Scheduling Toolbox by function `listsch` which also allows to user to use any of implemented strategy algorithms and visualize process of scheduling step by step in text form in MABLAB workspace (see Fig 4.5 for flowchart of function). Moreover, the last version is able to solve scheduling problems on unrelated parallel processors. The syntax of `listsch` function is:

```
TS = listsch(T,problem,processors [,strategy] [,verbose])
TS = listsch(T,problem,processors [,option])
```

where:

`T`	is the instance of the *Taskset* class without schedule,
`TS`	is the instance of the *Taskset* class with schedule,
`problem`	is the instance of the *Problem* class,
`processors`	is the number of processors,
`strategy`	is the strategy (like LPT, SPT, EST, ...),
`verbose`	is a level of verbosity,
`option`	is the optimization option setting.

This subsection concludes by the example. The example solves a problem of manufacturing of a chair by two workers (cabinetmakers). Their goal is to make four legs, seat and backrest of the chair and assembly all of these parts within minimal time. Material, which is needed to create backrest, will be available in 20 time units and assemblage is divided into two stages ($\text{assembly}_{1/2}$ and $\text{assembly}_{2/2}$). Fig. 4.6 shows the representation of mentioned problem by the graph.

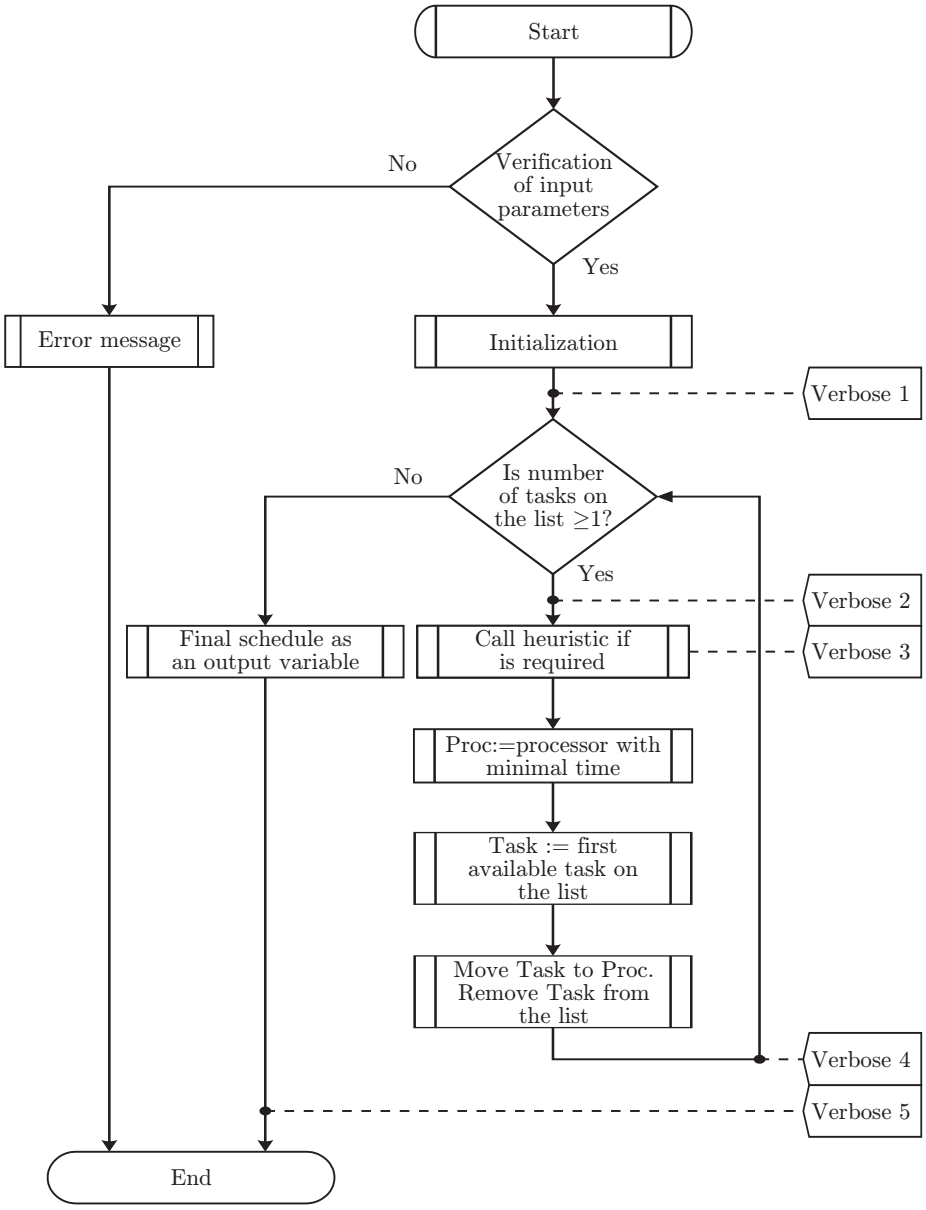


Fig. 4.5: Flow chart of `listsch` function

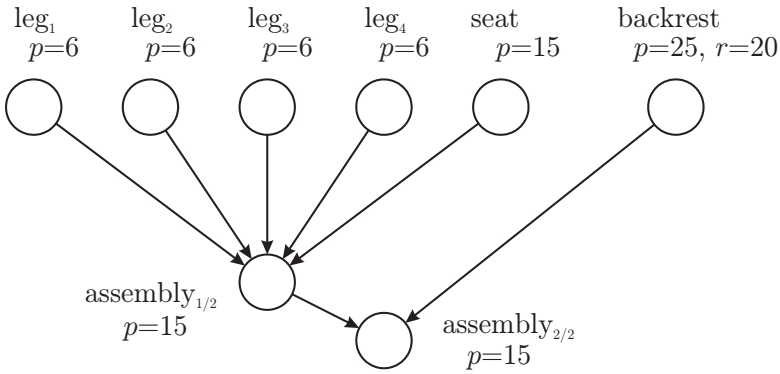


Fig. 4.6: Graph representation of chair manufacturing

Solution of the scheduling problem is shown in following steps:

1. Create desired tasks.

```
>> T1 = task('leg1',6)
Task "leg1"
 Processing time: 6
 Release time:    0

>> T2 = task('leg2',6);
>> T3 = task('leg3',6);
>> T4 = task('leg4',6);
>> T5 = task('seat',6);
>> T6 = task('backrest',25,20);
>> T7 = task('assembly1/2',15);
>> T8 = task('assembly2/2',15);
```

2. Define precedence constraints by precedence matrix prec. Matrix has size $n \times n$ where n is the number of tasks.

```
>> prec = [0 0 0 0 0 1 0 0;...
           0 0 0 0 0 1 0 0;...
           0 0 0 0 0 1 0 0;...
           0 0 0 0 0 1 0 0;...
           0 0 0 0 0 1 0 0;...
           0 0 0 0 0 0 0 1;...
           0 0 0 0 0 0 0 1;...
           0 0 0 0 0 0 0 0];
```

3. Create an object of taskset from recently defined objects.

```
>> T = taskset([T1 T2 T3 T4 T5 T6 T7 T8],prec)
Set of 8 tasks
 There are precedence constraints
```

4. Define solved problem.

```
>> p = problem('P|prec|Cmax')
P|prec|Cmax
```

5. Call List Scheduling algorithm with taskset and problem created recently and define number of processors and desired heuristic.

```
>> TS = listsch(T,p,2,'SPT')
Set of 8 tasks
 There are precedence constraints
 There is schedule: List Scheduling
   Solving time: 1.1316s
```

6. Visualize the final schedule by standard plot function, see Fig. 4.7.

```
>> plot(TS)
```

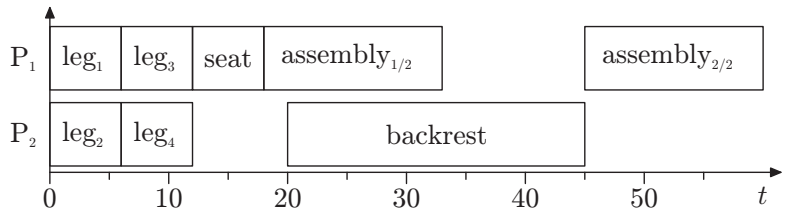


Fig. 4.7: The manufacture scheduling in the Gantt chart form

4.3.2 SAT Based Scheduling Algorithm

This subsection presents a satisfiability of boolean expression based algorithm for the scheduling problem $P|prec|C_{max}$, i.e. the scheduling of tasks with precedence constraints on the set of parallel identical processors while minimizing the schedule makespan. The main idea is to formulate the scheduling problem in the form of CNF (conjunctive normal form) clauses [Cram 06, Memi 02].

In the case of the $P|prec|C_{max}$ problem, each CNF clause is a function of the Boolean variables in the form χ_{ijk}. If the task $\mathbb{T}_i$ is started at time

unit j on the processor k then $\varkappa_{ijk} = true$, otherwise $\varkappa_{ijk} = false$. For each task $\mathbb{T}_i$, where $i = 1 \ldots n$, there are $\mathcal{S} \times \mathcal{R}$ Boolean variables, where $\mathcal{S}$ denotes the maximum number of time units and $\mathcal{R}$ denotes the total number of processors.

The Boolean variables are constrained by the three following rules (modest adaptation of [Memi 02]):

1. For each task, exactly one of the $\mathcal{S} \times \mathcal{R}$ variables has to be equal to *true*. Therefore two clauses are generated for each task $\mathbb{T}_i$. The first guarantees having at most one variable equal to *true*:
$(\bar{\varkappa}_{i11} \vee \bar{\varkappa}_{i21}) \wedge \cdots \wedge (\bar{\varkappa}_{i11} \vee \bar{\varkappa}_{i\mathcal{S}\mathcal{R}}) \wedge \cdots \wedge (\bar{\varkappa}_{i(\mathcal{S}-\infty)\mathcal{R}} \vee \bar{\varkappa}_{i\mathcal{S}\mathcal{R}})$.
The second guarantees having at least one variable equal to *true*: $(\bar{\varkappa}_{i11} \vee \bar{\varkappa}_{i21} \vee \cdots \vee \bar{\varkappa}_{i(\mathcal{S}-\infty)\mathcal{R}} \vee \bar{\varkappa}_{i\mathcal{S}\mathcal{R}})$.

2. If there are a precedence constraints such that $\mathbb{T}_u$ is the predecessor of $\mathbb{T}_v$, then $\mathbb{T}_v$ cannot start before the execution of $\mathbb{T}_u$ is finished. Therefore, $\varkappa_{ujk} \rightarrow (\bar{\varkappa}_{v1l} \wedge \cdots \wedge \bar{\varkappa}_{vjl} \wedge \bar{\varkappa}_{v(j+1)l} \wedge \cdots \wedge \bar{\varkappa}_{v(j+p_u-1)l})$ for all possible combinations of processors k and l, where p_u denotes the processing time of task $\mathbb{T}_u$.

3. At any time unit, there is at most one task executed on a given processor. For the couple of tasks with a precedence constrain this rule is ensured already by the clauses in the rule number 2. Otherwise the set of clauses is generated for each processor k and each time unit j for all couples $\mathbb{T}_u$, $\mathbb{T}_v$ without precedence constrains in the following form:
$(\varkappa_{ujk} \rightarrow \bar{\varkappa}_{vjk}) \wedge (\varkappa_{ujk} \rightarrow \bar{\varkappa}_{v(j+1)k}) \wedge \cdots \wedge (\varkappa_{ujk} \rightarrow \bar{\varkappa}_{v(j+p_u-1)k})$.

The toolbox cooperates with the *zChaff* solver [Mosk 01] to decide whether the set of clauses is satisfiable. If it is, the schedule within $\mathcal{S}$ time units is feasible. An optimal schedule is found in an iterative manner. First, the List Scheduling algorithm (see 4.3.1) is used to find the initial value of $\mathcal{S}$. Then the algorithm iteratively decreases the value of $\mathcal{S}$ by one and tests the feasibility of the solution. The iterative algorithm finishes when the solution is not feasible.

An example of the $P|prec|C_{max}$ problem can be taken from the digital signal processing area. A typical scheduling problem is to optimize the speed of a computation loop, e.g constituting the Jaumann wave digital filter [Groo 92]. The goal is to minimize the computation time of the filter loop, shown as a directed acyclic graph in Fig. 4.8. The nodes in the graph represent the tasks (i.e. operations of the loop) and the edges represent the

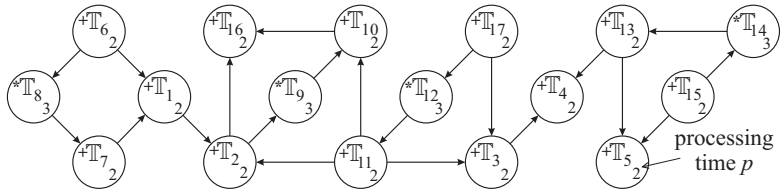


Fig. 4.8: Jaumann wave digital filter

precedence constraints. The nodes are labeled by the operation type ("+" or "∗") and processing time p_i. The example in Fig. 4.8 considers two parallel identical processors, i.e. two general arithmetic units.

Fig. 4.9 shows the consecutive steps performed in the toolbox. The first step defines the set of the tasks with the precedence constraints for the scheduling algorithm `satsch`. The resulting schedule is displayed by the `plot` command. The optimal schedule is depicted in Fig. 4.10.

```
>> procTime = [2,2,2,2,2,2,2,3,3,2,2,3,2,3,2,2,2];
>> prec = sparse([6,7,1,11,11,17,3,13,13,15,8,6,2, 9,11,12,17,14,15,2 ,10],...
                 [1,1,2, 2, 3, 3,4, 4, 5, 5,7,8,9,10,10,11,12,13,14,16,16],....
                 [1,1,1, 1, 1, 1,1, 1, 1, 1,1,1,1, 1, 1, 1, 1, 1, 1, 1, 1],17,17);
>> TS = taskset(procTime,prec);
>> TS = satsch(TS,problem(P|prec|Cmax),2)
Set of 17 tasks
 There are precedence constraints
 There is schedule: SAT solver
   SUM solving time: 0.06s
   MAX solving time: 0.04s
   Number of iterations: 2
>> plot(TS)
```

Fig. 4.9: Solution of the scheduling problem $P|prec|C_{max}$ in the toolbox

4.3.3 Minimum Cost Multi-commodity Flow Problem

Various optimization problems (e.g. routing) from the graph and network flow theory can be reformulated on the minimum cost multi-commodity flow (MMCF) problem. The objective of the MMCF is to find the cheapest possible ways of sending a certain amount of flows through the network. Therefore,

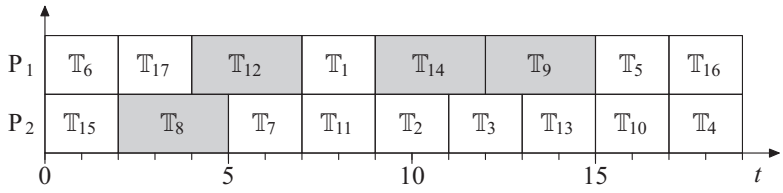


Fig. 4.10: The optimal schedule of the Jaumann filter

TORSCHE includes a `multicommodityflow` function.

The MMCF problem is defined by a directed flow network graph $G(\mathcal{V}, \mathcal{E})$, where the edge $(u, v) \in \mathcal{E}$ from node $u \in \mathcal{V}$ to node $v \in \mathcal{V}$ has a capacity cap_{uv} and a cost a_{uv}. There are ψ commodities $K_1, K_2, \ldots, K_\psi$ defined by $K_i = (source_i, sink_i, b_i)$ where $source_i$ and $sink_i$ stand for source and sink node of commodity i, and b_i is the volume of the demand. The flow of commodity i along the edge (u, v) is $f_i(u, v)$. The objective is to find an assignment of the flow $f_i(u, v)$ which minimizes the total cost $J = \sum_{\forall(u,v)\in E} \left(a_{uv} \cdot \sum_{i=1}^{\psi} f_i(u, v) \right)$ and satisfies the following constraints:

$$\begin{array}{rcll} \sum_{i=1}^{\psi} f_i(u,v) & \leq & cap_{uv} & \forall (u,v) \in \mathcal{E}, \\ \sum_{u\in\mathcal{V}} f_i(u,w) & = & \sum_{v\in\mathcal{V}} f_i(w,v) & w \in \mathcal{V} \setminus \{source_i, sink_i\}, \\ & & & \forall i = 1 \ldots \psi, \\ \sum_{w\in\mathcal{V}} f_i(source_i, w) & = & \sum_{w\in\mathcal{V}} f_i(w, sink_i) = b_i & \forall i = 1 \ldots \psi. \end{array} \tag{4.1}$$

The function `multicommodityflow` solves MMCF problem by the transformation to the linear programming problem [Kort 06]. Let $\mathfrak{P}$ be the set of all paths from the source node $source_i$ to the sink node $sink_i$ for all commodities $i = 1, 2, 3 \ldots \psi$. Let M be a 0-1-matrix whose columns correspond to the elements $\mathfrak{p}$ of $\mathfrak{P}$ and whose rows correspond to the edges of G, where $M_{(u,v),\mathfrak{p}} = 1$ iff $(u, v) \in \mathfrak{p}$. Similarly, let N be a 0-1-matrix whose columns correspond to the elements of $\mathfrak{P}$ and whose rows correspond to the commodities, where $N_{i,\mathfrak{p}} = 1$ iff $source_i$ and $sink_i$ are the start and end nodes of path $\mathfrak{p}$. Then the LP problem using the variables defined above is:

$$\min \left(a_{\mathfrak{P}} \cdot y \right), \tag{4.2}$$

subject to:

$$\begin{array}{rcl} y & \geq & 0, \\ M \cdot y & \leq & cap, \\ N \cdot y & = & b, \end{array} \tag{4.3}$$

where b is a vector of b_i, cap is a vector of cap_i and $a_{\mathfrak{P}}$ is a constant vector of a path costs. Each of its elements is defined as $a_{\mathfrak{p}} = \sum_{(u,v)\in\mathfrak{p}} a_{uv}$. Assignment $f_i(u,v)$ of multi-commodity flow to the edge is given by the vector y and set of paths $\mathfrak{P}$.

An example of minimum cost multi-commodity flow problem for a real application is shown in the Section 5.2.

4.4 Summary

This chapter presents the TORSCHE Scheduling Toolbox for Matlab covering: scheduling on monoprocessor/dedicated processors/parallel processors, open shop/flow shop/job shop scheduling, cyclic scheduling and real-time scheduling. The toolbox already has several real applications. It has been used for the development of a new method for re-configuration of the tasks or a process in an embedded avionics application [Muni 09]. Simulations in TORSCHE also helped to develop a method optimizing the jitter of tasks in a real-time system [Liu 09]. Recently, TORSCHE has become a part of a textbook for courses in scheduling "Scheduling: Theory, Algorithms, and Systems" written by M. Pinedo [Pine 08]. The actual version of the toolbox with documentation and screencasts is freely available at http://rtime.felk.cvut.cz/scheduling-toolbox/.

Chapter 5

Traffic Flow Optimization

This chapter proposes consecutive steps to find the optimal offset and the split of the light controlled intersections in the urban traffic region. The optimization takes into account the street length, number of lanes and maximal allowed vehicle speed in consideration to respect the green wave strategy control and constant intersection cycle time. The solution of this problem is shown on the urban traffic region in Prague. The problem is formalized and solved by the TORSCHE.

5.1 Introduction

The light controlled intersections are characterized by several parameters: the number of light phases, phase split, offset time and a list of streets from which the vehicles flow [Gube 08]. The term phase means state of traffic lights on the intersection. The number of phases and the list of streets are partially given by the urban architecture of the intersection and partially by the intersection control strategy (i.e. one-way street, directional roadway marking). Both of these parameters are constant. On the other hand, the split and offset can be changed dynamically during a day. The *split* τ_{vj} defines the time interval of phase j for which the vehicle flow can go through the intersection v from one or more streets [Papa 03]. The *offset* φ_{uv} is a certain time delay between phases of two successive intersections u and v. When the offset is zero, all lights in the region turn on and off at the same time. It is called the synchronized strategy. In the *green wave* strategy, the

Fig. 5.1: Traffic region in Prague (GPS: 50°4′27.446″N, 14°24′25.951″E) [Goog 09]

traffic light changes with time delay between the light phases of two successive intersections. As a result, signals switch as the green wave [Naga 07].

The goal of this chapter is to find the offset φ_{uv} respecting the green wave strategy and the split τ_{vj} which minimizes the total time spend on the road and considers a constant intersection *cycle time* $C_v = \sum_{\forall j} \tau_{vj}$ of intersection v such that $C_1 = C_2 = \cdots = C$. The problem is formalized and solved by the TORSCHE (Chapter 4). The solution of this problem is shown on the example of the light controlled intersections in an urban traffic region in Prague (see Fig. 5.1) and consists of the following steps:

- The model of an urban traffic region as the oriented graph made up.
- Source and destination nodes of graph mark for requirement traffic flows.
- The multi-commodity flow compute, which implies splits τ_{vj}.

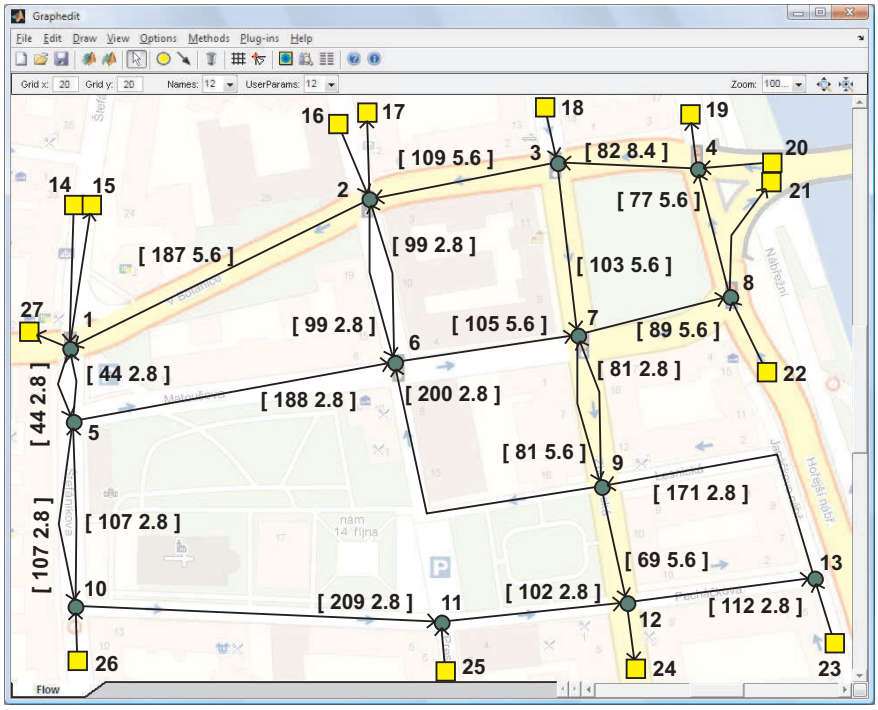


Fig. 5.2: Traffic region model

- Offset φ_{uv} from the street length and vehicle speed compute.
- By the scheduling technique find a schedule for intersection control.

5.2 Traffic Region Model

In the first step, a traffic region is modeled as an oriented graph $G(\mathcal{V}, \mathcal{E})$. Nodes $\mathcal{V}$ of the graph represent the intersections and edges $\mathcal{E}$ represent the streets. See Fig. 5.2 where the *Graphedit* tool of TORSCHE is utilized to construct the graph. Sink and source nodes are drawn as rectangles. The edges include two parameters; the first one is cost a_{uv} and the second one is capacity cap_{uv} of the street (u, v). The cost is given by the street length in meters. The capacity of the street is given by the number of lanes ℓ_{uv} in the street as $cap_{uv} = \ell_{uv} \cdot W_{uv}/l_{\mathrm{uv}}$ where W_{uv} is a maximal allowed vehicle speed

Table 5.1: Required traffic region multi-commodity flow instances

K_i	K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8
$source_i$	14	14	14	14	16	16	16	16
$sink_i$	24	17	21	27	24	21	15	27
$b_i[10^{-3}s^{-1}]$	4.9	1.9	3.1	30.1	10.1	12.1	1.2	6.6
K_i	K_9	K_{10}	K_{11}	K_{12}	K_{13}	K_{14}	K_{15}	K_{16}
$source_i$	18	18	18	18	18	20	20	20
$sink_i$	24	17	21	15	27	19	24	17
$b_i[10^{-3}s^{-1}]$	32.7	3.3	9.3	2.1	11.3	8.9	13.4	8.1
K_i	K_{17}	K_{18}	K_{19}	K_{20}	K_{21}	K_{22}	K_{23}	K_{24}
$source_i$	20	20	22	22	22	22	23	23
$sink_i$	15	27	19	21	15	27	24	17
$b_i[10^{-3}s^{-1}]$	7.9	41.7	11.7	88.5	4.7	25.1	8.6	4.3
K_i	K_{25}	K_{26}	K_{27}	K_{28}	K_{29}	K_{30}	K_{31}	K_{32}
$source_i$	23	23	25	25	26	26	26	26
$sink_i$	21	27	24	21	24	21	15	27
$b_i[10^{-3}s^{-1}]$	6.5	3.6	9.6	0.4	15.9	1.9	5.7	6.5

in the street in ms^{-1} and l_{uv} is the *unit vehicle* length including distance between vehicles. Let us assume that, in our case, the speed is $W_{uv} = 13.8\,\mathrm{ms}^{-1}$ (50 km/h) and the unit vehicle length is $l_{\mathrm{uv}} = 5\,\mathrm{m}$, then the capacity of one lane street is $2.8\,\mathrm{s}^{-1}$. The final graph is exported from the *Graphedit* tool to the Matlab workspace as graph object `G`.

In the second step, the multi-commodity flow method in the following form is called:

```
>> Gm = multicommodityflow(G,source,sink,b)
```

where the vectors `source`, `sink` and `b` define the required multi-commodity flow as is it described in Subsection 4.3.3. These variables are shown in Table 5.1. The graph `Gm` includes an assignment of optimal multi-commodity flow to the edges. We assume the drivers to make their decisions in a similar way. Table 5.2 shows a part of the assignment $f_i(u, v) : u = 6 \vee v = 6$. The complete result can be obtained from the *Graph* object by the command:

```
>> F = get(Gm,'edl')
```

Table 5.2: Multi-commodity flow assignment

u	K_i v	K_2	K_3	K_5	K_6	K_{24}	K_{26}	K_{30}	$\sum_{i=1}^{\psi} f_i(u,v)$ $[10^{-3}s^{-1}]$
		$f_i(u,v)$ $[10^{-3}s^{-1}]$							
2	6	0	0	10.1	12.1	0	0	0	22.2
5	6	1.9	3.1	0	0	0	0	1.9	6.9
9	6	0	0	0	0	4.3	3.6	0	7.9
6	2	1.9	0	0	0	4.3	3.6	0	9.8
6	7	0	3.1	10.1	12.1	0	0	1.9	27.2

5.3 Tasks Definition for Intersection

In the next step, the phase j split τ_{vj} and the offset φ_{uv} for each light controlled intersection $v \in \mathcal{V}$ are found. Continuous vehicle flow from the street (u, v) over a given number of intersection phases can be formalized as one task $\mathbb{T}_{uv}$ from the scheduling point of view. For example there exist three tasks $\mathbb{T}_{26}$, $\mathbb{T}_{56}$ and $\mathbb{T}_{96}$ for intersection 6. The number of phases is given by the urban architecture of the intersection and by the intersection control strategy. The intersections are the resources from the scheduling point of view and the tasks are dedicated there by the engineering skills.

The processing time p_{uv} of task $\mathbb{T}_{uv}$ is calculated by Algorithm 5.1 (Part of the Algorithm shows the solution of the consecutive steps in TORSCHE for intersection 6). This algorithm computes the processing time from the assignment of MMCF $f_i(u, v)$, from the cycle time C and from the precedence constraints of the tasks (defined by the intersection control strategy). Table 5.3 shows processing time of tasks for three intersections (6,7 and 8).

Table 5.3: Processing time of tasks and offset for intersections 6,7,8

u, v	5,6	2,6	9,6	6,7	3,7	9,7	7,8	22,8
p_{uv}	21.3	68.7	24.4	29.6	60.4	7.5	18.4	71.6
φ_{uv}	-	-	-	9.5	-	-	8	-

Algorithm 5.1 Processing time computation

1. Create tasks $\mathbb{T}_{uv}$, with temporary processing time $p'_{uv} = \sum_{i=1}^{\psi} f_i(u, v)$.

```
>> T56 = task('T(5,6)', 0.0069)
Task "T(5,6)"
 Processing time: 0.0069
 Release time:    0
>> T26 = task('T(2,6)', 0.0222);
>> T96 = task('T(9,6)',0.0079);
```

2. Group the tasks into a *taskset* and add precedence constrains.

```
>> prec6 = [0 1 1; 0 0 0; 0 0 0];
>> TS6 = taskset([T56 T26 T96],prec6);
Set of 3 tasks
 There are precedence constraints
```

3. Compute a length of critical path CP_v by the *asap* (as soon as possible) function.

```
>> TS6.asap;
>> asapStart = asap(TS6,'asap');
>> CP6 = max(asapStart + TS6.ProcTime)
CP6 =
    0.0291
```

4. From the length of the critical path and cycle time C we obtain processing time p_{uv} as a linear proportion of flow: $p_{uv} = p'_{uv} \cdot C/CP_v$.

```
>> C = 90;
>> TS6.ProcTime = TS6.ProcTime * C / CP6;
```

5. We can display intersection phases by the plot function, see to Fig. 5.4(b).

```
>> TS6.asap;
>> plot(TS6,'asap',1,'prec',0)
```

5.4 Scheduling with Communication Delay

The intersection phase offset and split is computed for the green wave strategy. The green wave strategy, specified by the engineering skills, extends the tasks precedence constraints by the relationships between successive intersection tasks. The precedence constraints must be in the out-tree form. Each of those relationships defines the offset φ_{uv} as a time, which a vehicle needs to pass from intersection u to intersection v. The φ_{uv} is given by the street length a_{uv} and vehicle speed W_{uv} as $\varphi_{uv} = a_{uv}/W_{uv}$ (see Table 5.3).

The split can be found by an algorithm for scheduling with a communication delay [Chre 95]. The *scheduling with communication delay* problem extends the precedence constraints in the classical scheduling by the communication delay between dependent tasks assigned to distinct processors. In our case the communication delay is equal to the offset φ_{uv}. Let `D` be a matrix of communication delays, where the elements are φ_{uv} in the case that the offset between intersections u and v is considered, zero otherwise. We can classify our instances as tasks with precedence constraints in an out-tree form, communication delays, unlimited number of processors and no duplication of tasks. In Graham and Błażewicz notation it can be denoted as $P_\infty|\textit{out-tree}, c_{jk}|C_{max}$. This problem can be solved in $\mathcal{O}(n)$ by the allgorithm presented in [Chre 89] which is implemented in the TORSCHE toolbox (see Chapter 4) as a function `chretienne`.

Fig. 5.3 shows the problem solution for three intersections (6, 7 and 8). First, the taskset object `TSall` with eight tasks corresponding to the intersection control is defined. The tasks and precedence constraints among them are shown in Fig. 5.4(a). The precedence constraints given by the green-wave strategy are drawn as solid lines. Consequently, matrix `D` and the notation of the problem `prob` is defined. Finally, the scheduling problem is solved by the algorithm `chretienne` and the resulting Gantt chart is shown in Fig. 5.4(b). The figure shows the tasks for the three considered intersections including the processing time p_{uv}, split τ_{vj} and offset φ_{uv}. The split is given by the processing time of the scheduled tasks. The tasks are periodically repeated with a cycle time C.

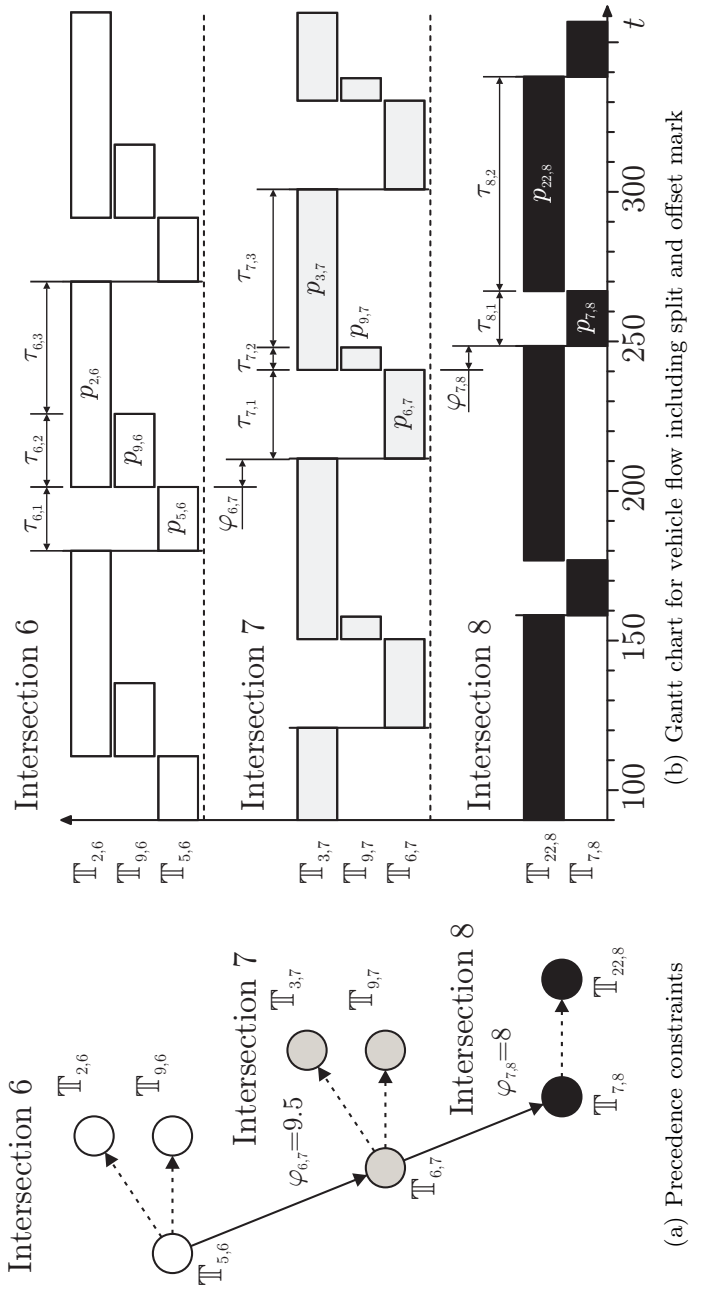
```
>> T56 = task('T(5,6)', 21.3);
>> T26 = task('T(2,6)', 68.7);
>> T96 = task('T(9,6)', 24.4);
>> prec6 = [0 1 1; 0 0 0; 0 0 0];
>> TS6 = taskset([T56 T26 T96],prec6);
>> T67 = task('T(6,7)', 29.6);
>> T37 = task('T(3,7)', 60.4);
>> T97 = task('T(9,7)',  7.5);
>> prec7 = [0 1 1; 0 0 0; 0 0 0];
>> TS7 = taskset([T67 T37 T97],prec7);
>> T78 = task('T(7,8)',  18.4);
>> T228 = task('T(22,8)', 71.6);
>> prec8 = [0 1; 0 0];
>> TS8 = taskset([T78 T228],prec8);
```

```
>> TSall = [TS6 TS7 TS8];
>> TSall.Prec(1,4) = 1;
>> TSall.Prec(4,7) = 1;

>> D = zeros(size(TSall.Prec));
>> D(1,4) = 9.5;
>> D(4,7) = 8;

>> prob = ...
  problem('Pinf|prec,out-tree,cjk|Cmax');

>> TSall = chretienne(TSall,p,Inf,D);
>> plot(TSall);
```

Fig. 5.3: Solution of the scheduling problem in the toolbox

5.5 Summary

In this chapter, the offset φ_{uv} and the split τ_{vj} of intersections were found. The offset is respecting the green wave strategy and the split is optimal under considered criterion (minimizes the time spend on the road — from the scheduling point of view minimizes C_{max}). The data from real light controlled intersections in an urban traffic region in Prague was used to demonstrate consecutive steps of the problem solution.



(a) Precedence constraints

(b) Gantt chart for vehicle flow including split and offset mark

Fig. 5.4: The intersections (6, 7 and 8) control

Chapter 6

Conclusions

6.1 Summary and Contributions

In this thesis, the modeling and optimization techniques to improve efficiency of light controlled intersections in urban area have been studied. The general procedures to make intersection models were shown as well optimization techniques to their control. All results were verified on real data from Prague traffic region.

The extended queue model based on number of vehicles in the queue and the mean value of waiting time was presented. Further, we have used the model to derive the parameters of the controllers for a simple intersection model. The advantages and disadvantages of the presented controllers were discussed.

General light controlled intersection model was based on CCPN. The new method for conflict resolution in CCPN has been presented. The conflict resolution method is based on the maximal speed proportion. Next, the iterative algorithm for this problem and its solution by LP was shown. The light controlled intersection model describes the traffic flow from the macroscopic point of view. This model is innovative, firstly, by the free space modeling together with the opposite direction of the vehicular flow. Secondly, by the constant speed continuous Petri net uses only.

The last part of thesis describes the optimization of traffic flow in urban region. The new algorithm for the offset and the split of intersections was presented. The offset is respecting the green wave strategy and the split is

optimal under considered criterion. The solution of this problem is solved by the TORSCHE. The performance of all depicted models and optimizations techniques was evaluated in simulations and compared with real data from the traffic in Prague.

The contributions of the thesis are summarized in Section 1.2. In our opinion, the main contributions are:

1. Formalization of the traffic intersection model (extended by the mean waiting time) used to drivers waiting time balancing.

2. CCPN based general light controlled intersection model verified on real data.

3. Algorithm for optimization of light controlled intersection traffic urban region.

6.2 Future Research

Current work aims at incorporating several intersections based on CCPN into a complex traffic region model. In order to model the traffic with higher precision (i.e. incorporating logarithmic stream model capturing the output flow as non-monotonic function of the flow density) we are developing a model based on continuous Petri Nets. As future work we would like to include additional practical constraints to the problem (e.g., supervisory systems performing high-level optimization on the model).

Appendices

Appendix A

TORSCHE Algorithms

A.1 Scheduling Algorithms

algorithm	command	problem
Algorithm for $1\|r_j\|C_{max}$	`alg1rjcmax`	$1\|r_j\|C_{max}$
Bratley's Algorithm	`bratley`	$1\|r_j, \widetilde{d}_j\|C_{max}$
Horn's Algorithm	`horn`	$1\|pmtn, r_j\|L_{max}$
Hodgson's Algorithm	`alg1sumuj`	$1\|\|\sum U_j$
Algorithm for $1\|\|\sum w_j D_j$	`alg1sumwjdj`	$1\|\|\sum w_j D_j$
Algorithm for $P\|\|C_{max}$	`algpcmax`	$P\|\|C_{max}$
Dynamic Prog. and $P\|\|C_{max}$	`algpcmaxdp`	$P\|\|Cmax$
McNaughton's Algorithm	`mcnaughtonrule`	$P\|pmtn\|C_{max}$
Algorithm for $P\|r_j, prec, \widetilde{d}_j\|C_{max}$	`algprjdeadlinepreccmax`	$P\|r_j, prec, \widetilde{d}_j\|C_{max}$
Hu's Algorithm	`hu`	$P\|in\text{-}tree, p_j = 1\|C_{max}$
Brucker's algorithm	`brucker76`	$P\|in\text{-}tree, p_j = 1\|L_{max}$
List Scheduling	`listsch`	$P\|prec\|C_{max}$
Coffman's and Graham's Algorithm	`coffmangraham`	$P2\|prec, p_j = 1\|C_{max}$
SAT Scheduling	`satsch`	$P\|prec\|C_{max}$
Johnson's Algorithm	`johnson`	$F2\|\|C_{max}$
Gonzales Sahni's Algorithm	`gonzalezsahni`	$O2\|\|C_{max}$
Jackson's Algorithm	`jackson`	$J2\|n_j \leq 2\|C_{max}$
Algorithm cpshopscheduler	`cpshopscheduler`	$J, F, O\|\|C_{max}$
Alg. for $F2, R1\|p_{ij} = 1, t_j\|C_{max}$	`algf2r1pijtjcmax`	$F2, R1\|p_{ij} = 1, t_j\|C_{max}$
Alg. for $F\|\|C_{max}$ with lim. buffers	`fslb`	$F\|\|C_{max}$
Alg. for $O\|p_{ij} = 1\|\sum T_i$	`algopij1sumti`	$O\|p_i = 1\|\sum T_i$
Positive and Negative Time-Lags	`spntl`	$SPNTL$
Cyclic scheduling (General)	`cycsch`	$CSCH$
SAT Scheduling	`satsch`	$P\|prec\|C_{max}$

A.2 Graph Algorithms

algorithm	command
Minimum spanning tree	`spanningtree`
Dijkstra's algorithm	`dijkstra`
Floyd's algorithm	`floyd`
Tarjan's algorithm	`tarjan`
Minimum Cost Flow	`mincostflow`
Critical Circuit Ratio	`criticalcircuitratio`
Hamilton circuit	`hamiltoncircuit`
Christofides	`christofides`
MWPM	`mwpm`
Multicommodity flow	`multicommodityflow`
All path	`allpath`
K shortest path	`xbestpath`
Commodity flow	`commodityflow`
Graph coloring	`graphcoloring`
Quadratic Assignment Problem	`qap`

A.3 Other Optimization Algorithms

algorithm	command
Knapsack problem	`knapsack`
Knapsack problem graph	`knapsack_graph`

Bibliography

[Abou 09] K. Aboudolas, M. Papageorgiou, and E. Kosmatopoulos. "Store-and-forward based methods for the signal control problem in large-scale congested urban road networks". *Transportation Research Part C: Emerging Technologies*, Vol. 17, No. 2, pp. 163 – 174, 2009. Selected papers from the Sixth Triennial Symposium on Transportation Analysis (TRISTAN VI).

[Ande 05] M. Andersson, D. Henriksson, and A. Cervin. *TrueTime 1.3– Reference Manual*. Department of Automatic Control, Lund University, Sweden, Lund University, Sweden, 2005. http://www.control.lth.se/truetime/.

[Andr 03] M. Andresen, H. Bräsel, F. Engelhardt, and F. Werner. *LiSA - A Library of Scheduling Algortihms*. Otto-von-Guericke-Universität Magdeburg, 2003. http://lisa.math.uni-magdeburg.de/.

[Astr 97] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems: Theory and Design*. Prentice Hall, November 20 1997.

[Blaz 83] J. Błażewicz, J. Lenstra, and A. R. Kan. "Scheduling subject to resource constraints. Classification and complexity". *Discrete Applied Mathematics*, Vol. 5, No. 5, pp. 11–24, 1983.

[Bonn 95] J. A. Bonneson and P. T. Mccoy. "Average duration and performance of actuated signal phases". *Transportation Research Part A: Policy and Practice*, Vol. 29, No. 6, pp. 429 – 443, 1995.

[Boyd 04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, March 2004.

[Cast 96] D. J. M. Castillo. "A car-following model based on the Lighthill-Whitham Theory". In: J. Lesort, Ed., *Proceedings of the 13th International Symposium of Transportation and Traffic Theory*, pp. 517–538, 1996.

[Chre 89] P. Chrétienne. "A Polynomial Algorithm to Optimally Schedule Tasks on a Virtual Distributed System under Tree-like Precedence Constraints". *European Journal of Operational Research*, Vol. 43, No. 43, pp. 225–230, 1989.

[Chre 95] P. Chrétienne, E. G. Coffman, J. K. Lenstraand, and Z. Liu. *Scheduling theory and its applications*. John Wiley & Sons Ltd, Baffins Lane, Chichester, West Sussex PO19 1UD, England, 1995.

[Cram 06] Y. Crama and P. L. Hammer. "Boolean Functions: Theory, Algorithms and Applications". 2006. http://www.rogp.hec.ulg.ac.be/Crama/Publications/BookPage.html.

[Davi 01] R. David and H. Alla. "On Hybrid Petri Nets". *Discrete Event Dynamic Systems*, Vol. 11, No. 1-2, pp. 9–40, 2001.

[Davi 04] R. David and H. Alla. *Discrete, Continuous, and Hybrid Petri Nets*. Springer, November 23 2004.

[Davi 98] R. David and H. Alla. "A modeling and analysis tool for discrete events systems: continuous Petri net". *Performance Evaluation*, Vol. 33, pp. 175–199, 1998.

[Diak 02] C. Diakaki, M. Papageorgiou, and K. Aboudolas. "A multivariable regulator approach to traffic-responsive network wide signal control". *Control Engineering Practice*, Vol. 10, No. 2, pp. 183–195, February 2002.

[Febb 01] A. D. Febbraro, D. Giglio, and N. Sacco. "Modular representation of urban traffic systems based on hybridPetri nets". In: *Intelligent Transportation Systems*, pp. 866–871, Oakland, CA, USA, 2001.

[Febb 04] A. D. Febbraro, D. Giglio, and N. Sacco. "Urban traffic control structure based on hybrid Petri nets". *IEEE Transactions on Intelligent Transportation Systems*, Vol. 5, pp. 224–237, 2004.

[Febb 06] A. D. Febbraro and D. Giglio. “Urban traffic control in modular/switching deterministic-timed Petri nets”. In: *11th IFAC Symposium on Control in Transportation Systems*, 2006.

[Figu 01] L. Figueiredo, I. Jesus, J. Machado, J. Ferreira, and J. Santos. “Towards the development of intelligent transportation systems”. In: *IEEE Intelligent Transportation Systems Proceedings*, p. 29, 2001.

[Find 02] R. Findeisen and F. Allgöwer. “An Introduction to Nonlinear Model Predictive Control”. In: *21st Benelux Meeting on Systems and Control*, March 2002.

[Gart 83] N. H. Gartner. “OPAC: A demand-responsive strategy for traffic signal control”. *Transportation Research Record*, Vol. 906, pp. 75–81, 1983.

[Gazi 02] D. C. Gazis. *Traffic theory.* Kluwer, 2002.

[Gazi 63] D. C. Gazis and R. Potts. “The oversaturated intersection”. In: *Proc. 2nd Int. Symp. Traffic Theory*, pp. 221–237, 1963.

[Gonz 08] M. Gonzalez *et al.* “MAST (Modeling and Analysis Suite for Real-Time Applications)”. http://mast.unican.es/, 2008.

[Goog 09] Google. “Prague Google Maps”. http://maps.google.com/, December 2009.

[Groo 92] S. H. de Groot, S. Gerez, and O. Herrmann. “Range-chart-guided iterative data-flow graph scheduling”. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, Vol. 39, pp. 351–364, 1992.

[Gube 08] S. Guberinić, G. Šenborn, and B. Lazić. *Optimal Traffic Control: Urban Intersections.* CRC Press, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, 2008.

[Haef 98] L. E. Haefner and M.-S. Li. “Traffic Flow Simulation for an Urban Freeway Corridor”. *Transportation Conference Proceedings*, 1998.

[Hanz 03] Z. Hanzálek. "Continuous Petri Nets and Polytopes". In: *IEEE International Conference on Systems Man & Cybernetics, Washington, D.C.*, October 2003.

[He 06] H. He, L. Dong, and S. Dai. "Simulation of traffic flow with traffic light strategies via a modified cellular automaton model". *Journal of Shanghai University (English Edition)*, Vol. 10, No. 3, pp. 189–191, 2006.

[Henr 04] D. Henriksson, Y. Lu, and T. F. Abdelzaher. "Improved Prediction for Web Server Delay Control". In: *ECRTS*, pp. 61–68, IEEE Computer Society, 2004.

[Henr 83] J. J. Henry, J. L. Farges, and J. Tuffal. "The PRODYN real time traffic algorithm". In: *4th IFAC-IFIP-IFORS Conference on Control in Transportation System*, BadenBaden, Germany, September 1983.

[Homo 05] J. Homolová and I. Nagy. "Traffic Model of a Microregion". In: *IFAC World Congress*, 2005.

[Hunt 82] P. B. Hunt, D. I. Robertson, and R. D. Bretherton. "The SCOOT on-line traffic signal optimization technique". *Traffic Eng. Control*, Vol. 23, pp. 190–192, 1982.

[ILOG 09] ILOG. *ILOG CP Optimizer*. IBM Corporation, ILOG Europe, 9 rue de Verdun, BP 85, 94253 Gentilly Cedex, 2009. http://www.ilog.com/products/cpoptimizer/.

[Julv 05] J. Júlvez and R. Boel. "Modelling and Controlling Traffic Behaviour with Continuous Petri Nets". In: *16th IFAC World Congress 2005*, Elsevier, 07 2005.

[Kort 06] B. H. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer-Verlag, Berlin, Heidelberg, third Ed., 2006.

[Kuti 07] M. Kutil, P. Šůcha, M. Sojka, and Z. Hanzálek. *TORSCHE Scheduling Toolbox for Matlab: User's Guide*. Centre for Applied

Cybernetics, Department of Control Engineering, Czech Technical University in Prague, October 2007.
http://rtime.felk.cvut.cz/scheduling-toolbox/.

[Kwak 72] H. Kwakernaak. *Linear Optimal Control Systems*. John Wiley & Sons, Inc., New York, NY, USA, 1972.

[Lei 01] J. Lei and U. Ozguner. "Decentralized hybrid intersection control". In: *Proceedings of the 40th IEEE Conference on Decision and Control*, pp. 1237–1242, 2001.

[Leun 04] J. Y.-T. Leung. *Handbook of Scheduling*. Chapman & Hall/CRC, 2004.

[Litt 61] J. D. C. Little. "A Proof of the Queueing Formula $L = \lambda W$". *Operations Research*, Vol. 9, pp. 383–387, 1961.

[Liu 09] Z. Liu, H. Zhao, P. Li, and J. Wang. "An Optimization Model for IO Jitter in Device-Level RTOS". In: *ITNG '09: Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, pp. 1528–1533, IEEE Computer Society, Washington, DC, USA, 2009.

[Magn 03] L. Magni, G. De Nicolao, R. Scattolini, and F. Allgwer. "Robust model predictive control for nonlinear discrete-time systems.". *Int. J. Robust Nonlinear Control*, Vol. 13, No. 3-4, pp. 229–246, 2003.

[Man 00] I. T. K. Man. "Intelligent Transport Systems". In: *Better air Quality Motor Vehicle Control & Technology Workshop 2000*, 2000.

[Masu 07] S. Masukura, T. Nagatani, K. Tanaka, and H. Hanaura. "Theory and simulation for jamming transitions induced by a slow vehicle in traffic flow". *Physica A: Statistical Mechanics and its Applications*, Vol. 379, pp. 263–273, 2007.

[Memi 02] S. O. Memik and F. Fallah. "Accelerated SAT-based Scheduling of Control/Data Flow Graphs". In: *ICCD '02: Proceedings of the 2002 IEEE International Conference on Computer Design: VLSI*

in Computers and Processors, p. 395, IEEE Computer Society, Washington, DC, USA, 2002.

[Mosk 01] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. "Chaff: Engineering an Efficient SAT Solver". In: *Proceedings of the 38th Design Automation Conference (DAC'01)*, pp. 530–535, ACM, 2001.

[Muni 09] A. C. Muniyappa. *Computer Safety, Reliability, and Security*. Springer, Berlin Heidelberg, 2009.

[Naga 07] T. Nagatani. "Vehicular traffic through a sequence of green-wave lights". *Physica A: Statistical Mechanics and its Applications*, Vol. 380, pp. 503 – 511, 2007.

[Nage 03] K. Nagel, P. Wagner, and R. Woesler. "Still flowing: Approaches to traffic flow and traffic jam modeling". *Operations research*, Vol. 51, No. 5, pp. 681–710, 2003.

[Papa 03] M. Papageorgiou, C. Diakaki, V. Dinopoulou, A. Kotsialos, and Y. Wang. "Review of Road Traffic Control Strategies". *PROCEEDINGS - IEEE*, Vol. 91, pp. 2043–2067, 2003.

[Pine 02] M. Pinedo *et al. LEKIN® - Flexible Job-Shop Scheduling System*. New York University, Leonard N. Stern School of Business New York, NY, 2002.
http://www.stern.nyu.edu/om/software/lekin/.

[Pine 08] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 233 Spring Street, NewYork, NY 10013, USA, third Ed., 2008.

[Prag 09] Prague city. "Statistika registru silničních vozidel v hl.m. Praze". http://doprava.praha-mesto.cz/, December 2009.

[Robe 69] D. Robertson. "TRANSYT method for area traffic control". *Traffic Eng. Control*, Vol. 10, p. 276, 1969.

[Sen 97] S. Sen and L. K. Head. "Controlled Optimization of Phases at an Intersection". *Transportation Science*, Vol. 31, pp. 5–17, 1997.

[Tolb 01] C. Tolba. "Continuous Petri Nets Models for the Analysis of Traffic Urban Networks". In: *Proceedings of IEEE Systems, Man, and Cybernetics Conference, Arizona, USA*, pp. 1323–1328, 2001.

[Tolb 03] C. Tolba, P. Thomas, A. ElMoudni, and D. Lefebvre. "Performances evaluation of the traffic control in a single crossroad by Petri nets". In: *IEEE Emerging Technologies and Factory Automation*, 2003.

[Tolb 05] C. Tolba, D. Lefebvre, P. Thomas, and E. A. Moudni. "Continuous and timed Petri nets for the macroscopic and microscopic traffic flow modelling". *Simulation Modelling Practice and Theory*, Vol. 13, No. 5, pp. 407–436, July 2005.

[Wang 93] H. Wang, G. F. List, and F. Dicesare. "Modeling and Evaluation of Traffic Signal Control Using Timed Petri Nets". In: *IEEE International Conference on Systems, Man and Cybernetic*, 1993.

Index

Curriculum Vitae

Michal Kutil was born in Kutná Hora, Czech Republic, in 1980. He received the his master's degree in Electrical Engineering from the Czech Technical University (CTU) in Prague in 2004. From 2004 he was a Ph.D. student at the Czech Technical University and from 2005 he has had the position of a full time researcher at the Center for Applied Cybernetics at CTU. His research interests include scheduling and urban traffic flow modeling and control.

His teaching activities at CTU cover courses on Introduction to control, Control Systems, Systems and control, Distributed Control Systems and Industrial Informatics and Internet. He has supervised several students' projects and diploma theses.

Michal Kutil completed a three-month stay at Lund University, Department of Automatic Control (Sweden) where he worked on a simple intersection model control.

Research results of Michal Kutil have been presented at prestigious international conferences and workshops including the 11th IFAC Symposium on Control in Transportation Systems in Delft (Netherlands), 12th IFAC Symposium on Control in Transportation in California (USA), 17th IFAC World Congress in Seoul (Korea), etc.

List of Author's Publications

Papers submitted to international journals

Michal Kutil, Přemysl Šůcha, Roman Čapek, and Zdeněk Hanzálek. Torsche scheduling toolbox for matlab (**co-authorship 25%**). *Transactions on Mathematical Software*, submitted 2010.

Michal Kutil and Zdeněk Hanzálek. Traffic Intersection Model Based on Constant Speed Continuous Petri Net (**co-authorship 50%**). *Transactions on Intelligent Transportation Systems*, submitted 2009.

Michal Kutil, Zdeněk Hanzálek, and Anton Cervin. Minimization the Waiting Times in Traffic Intersection Control (**co-authorship 35%**). *Control Engineering Practice*, submitted 2009.

International Conference Papers

M. Kutil and Z. Hanzálek. Light controlled intersection model based on the continuous petri net (**co-authorship 50%**). In *12th IFAC Symposium on Transportation Systems*, pages 519–525, Laxenburg, 2009. IFAC.

M. Kutil, P. Šůcha, and Z. Hanzálek. Scheduling and Simulation in TORSCHE Toolbox. In *17th IFAC World Congress-Workshop on Embedded Control Systems: from design to implementation (**co-authorship 35%**)*, Seoul, 2008. Seoul National University.

V. Navrátil and M. Kutil. Torsche Scheduling Toolbox and Graph Theory. In *Summary Volume of 16th International Conference on Process Control '07*

*(**co-authorship 50%**)*, page 152, Bratislava, 2007. Slovak University of Technology.

M. Kutil, Z. Hanzálek, and A. Cervin. Balancing the Waiting Times in a Simple Traffic Intersection Model (**co-authorship 50%**). In *11th IFAC Symposium on Control in Transportation Systems*, pages 313–318, New York, 2006. IFAC.

P. Šůcha, M. Kutil, M. Sojka, and Z. Hanzálek. TORSCHE Scheduling Toolbox for Matlab (**co-authorship 25%**). In *IEEE Symposium on Computer-Aided Control System Design 2006*, pages 277–282, Piscataway, 2006. IEEE.

M. Stibor and M. Kutil. Torsche Scheduling Toolbox: List Scheduling (**co-authorship 50%**). In *Proceedings of Process Control 2006*, Pardubice, 2006. University of Pardubice.

R. Láska and M. Kutil. AGMAWEB - Automatically generated Matlab Web Server presentations (**co-authorship 50%**). In *15th International Conference on Process Control 05*, Bratislava, 2005. Slovak University of Technology.

M. Kutil. Control of model using Internet (**co-authorship 100%**). In *8th International Student Conference on Electrical Engineering, POSTER 2004, May 20 2004, Prague*, Praha, 2004. ČVUT v Praze, FEL.

J. Fuka, M. Kutil, and F. Vaněk. SARI - Internet Textbook for Basic Control Education (**co-authorship 30%**). In *Proceedings of Process Control '03*, pages 106–1–106–4, Bratislava, 2003. Slovak University of Technology.

F. Vaněk and M. Kutil. Application in Control (**co-authorship 10%**). In *Proceedings of the 5th International Scientific - Technical Conference*, pages R171–1–R171–5, Pardubice, 2002. University of Pardubice.

Other publications

P. Šůcha, M. Kutil, and Z. Hanzálek. TORSCHE Scheduling Toolbox for Matlab (**co-authorship 35%**). In *ARTIST Graduate Course on Embedded Control Systems*, pages 121–129, Stockholm, 2008. Royal Institute of Technology.

P. Šůcha and M. Kutil. TORSCHE Scheduling Toolbox for Matlab(**co-authorship 50%**). In *Graduate Course on Embedded Control Systems*, pages 347–361, Prague, 2006. CTU, Faculty of Electrical Engineering, Department of Control Engineering.

M. Kutil. Scheduling Toolbox for Use with Matlab (**co-authorship 100%**). In *CTU Reports - Proceedings of Workshop 2006*, volume A, pages 160–161, Praha, 2006. Česká technika - nakladatelství ČVUT.

M. Kutil. Splnitelnost Booleovských formulí (**co-authorship 100%**). Research Report K13135/05/231, ČVUT, FEL, Praha, 2005.

J. Fuka and M. Kutil. Internet Texbook SARI and Remote Scale Models (**co-authorship 50%**). In *Proceedings of Workshop 2005*, pages 308–309, Prague, 2005. CTU.

M. Kutil. Scheduling Toolbox First Preview (**co-authorship 100%**). In *MATLAB 2004 - Sborník příspěvků 12. ročníku konference*, volume 1-2, pages 297–303, Praha, 2004. VŠCHT.