

PARAMICS Plugin Document – BOTTLENECK ramp metering control

Lianyu Chu

PATH ATMS Center
University of California, Irvine

Plugin Compatibility: V4
Release date: 3/20/2003

522 Social Science Tower
Irvine, CA 92697-3600
URL: <http://www.its.uci.edu/>



Table of Contents

Table of Contents.....	2
1. Introduction.....	3
2 Plugin implementation.....	4
2.1 Algorithm description.....	4
2.2 Development framework.....	5
2.3 Pseudo codes.....	6
3 Step-by-step user manual.....	7
3.1 Adding detectors.....	7
3.2 Preparation of “bottleneck_control” file.....	7
3.3 Loading plugin.....	10
3.4 Output file.....	12
3.5 Error checking.....	12
3.6 Calibration of the BOTTLENECK algorithm.....	12
4 Technical supports.....	14
4.1 Release notes.....	14
4.2 Contact information.....	14

1. Introduction

The BOTTLENECK algorithm has been applied in Seattle, Washington for several years (Jacobsen, *et al.*, 1989). This plugin is to implement the BOTTLENECK ramp-metering algorithm in PARAMICS.

2 Plugin implementation

2.1 Algorithm description

The BOTTLENECK algorithm has three components: a local algorithm computing local-level metering rates based on local conditions, a coordination algorithm computing system-level metering rates based on system capacity constraints, and adjustment to the metering rates based on local ramp conditions.

The local metering algorithm employed by the BOTTLENECK algorithm is occupancy control. The metering rate for the occupancy control is selected from a predetermined, finite set of discrete metering rates, on the basis of occupancy levels upstream of the given metered ramp. Historical data collected from the given detector station are used to approximate volume-occupancy relationships, which will be used to calculate the predetermined set of metering rates.

The coordination algorithm is the unique aspect of BOTTLENECK. The freeway segment under control is divided into several sections, each of which is defined by the stretch of freeway between two adjacent mainline loop stations. A section is identified as a bottleneck if it satisfies two conditions, i.e. capacity condition and vehicle storage condition. The capacity condition can be described as:

$$O_{down}(i, t) \geq O_{threshold}(i) \quad (3)$$

where $O_{down}(i, t)$ is the average occupancy of the downstream detector station of section i over the past one-minute period $(t-1, t)$; $O_{threshold}(i)$ is a pre-defined loop station occupancy threshold when it is operating near capacity. The vehicle storage condition can be formulated as:

$$Q_{reduction}(i, t) = (Q_{up}(i, t) + Q_{on}(i, t)) - (Q_{off}(i, t) + Q_{down}(i, t)) \geq 0 \quad (4)$$

where $Q_{reduction}(i, t)$ is the number of vehicles stored in section i during the past minute. $Q_{up}(i, t)$ and $Q_{down}(i, t)$ are the volume entering section i across the upstream detector station and the volume exiting section i across the downstream detector station during the past minute, respectively; $Q_{on}(i, t)$ is the total volume entering section i from on-ramps during the past minute; $Q_{off}(i, t)$ is the total volume exiting section i to off-ramps during the past minute.

The number of vehicles stored in the bottleneck section $Q_{reduction}(i, t)$ should be reduced. Each section needs to define an area of influence that consists of a number of upstream on-ramps for the volume reduction. The amount of volume reduction from an on-ramp is determined by a weighting factor, pre-defined according to how far it is to the downstream detector station of the bottleneck section and the historical demand pattern

from the on-ramp. If on-ramp j involves in the volume reduction of any bottleneck section, its system-level metering rate is calculated as:

$$r(j, t) = Q_{on}(j, t-1) - \underset{i=1}{\overset{n}{MAX}}(Q_{reduction}(i, t) \bullet ((WF_j)_i / \sum_j (WF_j)_i)) \quad (5)$$

where $\underset{i=1}{\overset{n}{MAX}}$ is defined as the operator of selecting the maximum volume reduction if the on-ramp is located within more than one section's area of influence. $Q_{on}(j, t-1)$ is the entrance volume from on-ramp j during the past minute; $(WF_j)_i$ is the weighting factor of on-ramp j within the area of influence for section i ; $Q_{reduction}(i, t) \bullet ((WF_j)_i / \sum_j (WF_j)_i)$ is the volume reduction of on-ramp j because of section i .

The more restrictive of the local rate and the system rate will be selected for further adjustments, including queue adjustment, ramp volume adjustment and advanced queue override. The queue adjustment and advanced queue override are used for preventing traffic spillback onto arterials. Ramp volume adjustment copes with the condition that more vehicles have entered the freeway compared to the number of vehicles assumed to enter, which may be caused by HOV traffic or HOV lane violators. The metering rate to be finally implemented should be within the range of the pre-specified minimum and maximum metering rates.

2.2 Development framework

Figure 1 illustrates the hierarchical development framework of advanced ramp-metering algorithm plugins in PARAMICS. The advanced ramp-metering algorithm plugin is built on top of two basic plugin modules, i.e., ramp metering controller and loop data aggregator. The on-ramp signals in the simulation network are controlled by the ramp metering plugin, through which metering rates can be queried and set by other plugin modules. The loop data aggregator emulates the real-world loop data collection, typically with a thirty-second polling interval, and broadcast the latest loop data to the dynamic memory. At each time increment, the advanced ramp-metering algorithm plugin can access the dynamic memory and obtains the required loop data through interface functions provided by the loop data aggregation plugin. Then the metering rate for the next control interval is calculated based on the advanced ramp-metering algorithm. The new metering rate is finally sent back to the ramp controller plugin for implementation.

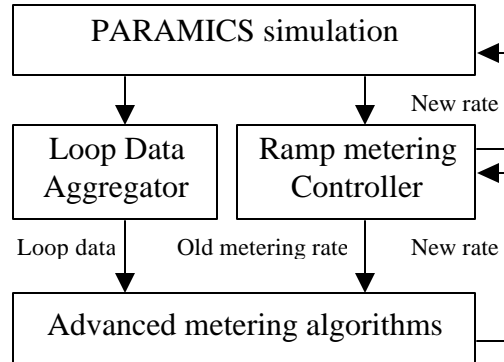


Figure 1 The hierarchical approach for the development of advanced metering algorithms

2.3 Pseudo codes

The control logic of the BOTTLENECK algorithm plugin is implemented as the following pseudo codes:

1. Communicating with ramp metering API and loop data aggregator API in order to obtain up-to-date traffic information and historical metering rates.
2. Calculating and then checking if the two conditions are met. If they are met, the system metering rate and local metering rate are calculated and the most restrictive one is selected for further adjustment. Otherwise, the local metering rate is selected for further adjustment.
3. Metering rate restriction
4. Checking if the queuing strategy needs to be activated.
5. HOV adjustment
6. Sending its computed metering rate to the ramp metering API for implementation.

3 Step-by-step user manual

3.1 Adding detectors

BOTTLENECK needs to put mainline detectors at the spacing of 0.5 to 1.0 mile to the target freeway in order to capture the traffic congestion dynamics. As a result, the freeway segment under control is divided into several sections, each of which is defined by the stretch of freeway between two adjacent mainline detector stations.

3.2 Preparation of “bottleneck_control” file

The “bottleneck_control” file includes all necessary information required by the BOTTLENECK API. Some of inputs should be calibrated based on the modeled network before implementation. An example of the file is as follows:

```

total number of bottleneck controlled sections is 13
checking control file                no
update cycle of metering rate        30
time period to accumulate detector data 60
algorithm activation time             07:00:00
algorithm deactivation time          09:00:00
report metering rate                 yes
local algorithm                      ALINEA

section                               1
upstream loop                        405n0.93ml
downstream loop                      405n1.11ml
number of influenced ramps           1
ramps                                33
reduction factors                    1.0
number of onramps                    1
onramp loops                         405n0.93ora
number of offramps                   0
offramp loops
number of unmetered ramps            0
unmetered ramp loops
desired downstream occupancy          0.20

...

section                               13
upstream loop                        405n5.74ml
downstream loop                      405n6.21ml
number of influenced ramps           2

```

ramps	95 92
reduction factors	0.37 0.63
number of onramps	1
onramp loops	405n5.74ora
number of offramps	0
offramp loops	
number of unmetered ramps	0
unmetered ramp loops	
desired downstream occupancy	0.20

total number of entrance ramps is 7

ramp	33
mainline detector	405n0.93ml
queue detector	405n0.93orspill
on-ramp detector	405n0.93orb
HOV	0
control type	1
rate restriction	240 900
desired occupancy	0.20
regulator	70.0
metering level	
occupancy threshold	
metering cycle	
...	

There are three parts of this control file. The first part is the basic information of the BOTTLENECK algorithm.

The option of “checking control file” is used for checking if there are any mistakes in the control file. If “yes”, this API will print out the information obtained from “bottleneck_control” file when this API is loaded.

“update cycle of metering rate” is the time interval of metering rate calculation.

The option “time period to accumulate detector data” determines if more than one observation of loop data will be used for metering rate calculation. For example, if “time period to accumulate detector data” is 60 and “metering rate update cycle” is 30, the loop data at time interval t-1 and t-2 will be accumulated first and then used for calculating the metering rate at time t. The value of “time period to accumulate detector data” must be integer times of the value of “metering rate update cycle”.

During the time period between “algorithm activation time” and “algorithm deactivation time”, the ramp-metering algorithm is functional.

If “report metering rate” is yes, the metering information of all BOTTLENECK controlled ramps will be output (every update cycle) to a file named “BOTTLENECK-rampRate.txt”.

The current applied “local algorithm” should be also specified. The local algorithms can be either OCCUPANCY_CONTROL or ALINEA (Therefore, ALINEA algorithm has been integrated in this API).

The second part is the section information. “upstream loop” and “downstream loop” are the names of upstream and downstream loop detectors.

“number of influenced ramps” is the number of on-ramps in the area of influence of this section. “ramps” is the node name of influenced on-ramps. If there is more than one on-ramp, please use a space between any two on-ramp names.

“reduction factors” is the correspondent reduction factors of on-ramps specified in the last row. “number of onramps” is the number of on-ramp in the section. If there is more than one on-ramp in the section, their names should be specified in the row of “onramp loops”. If there is no on-ramp in the section, please leave blank nothing in the row of “onramp loops”.

“number of offramps” is the number of off-ramps in the section. If there is more than one off-ramp in the section, their names should be specified in the row of “offramp loops”. Otherwise, leave blank nothing. “number of unmetered ramps” is the number of unmetered on-ramps in the section. If there is more than one unmetered on-ramp in the section, their names should be specified in the row of “unmetered ramp loops”. Otherwise, leave blank.

“desired downstream occupancy” is the occupancy threshold of the downstream loop detector station.

The third part of the file is the entrance ramp part. If a queue detector is specified, the queuing strategy is involved in BOTTLENECK (queuing strategy is required by BOTTLENECK). If there is no queue detector or no queuing strategy involved in ramp metering, please specify as “N/A”. If the specified detector cannot be found in the “detectors” file, a warning message will be given and the entrance ramp will be operated without any queue strategy.

If ALINEA is specified as the local algorithm in the first part of the file, please fill in the rows of “desired occupancy” and “regulator” and leave rows of “metering level”, “occupancy threshold” and “metering cycle” blank. If OCCUPANCY_CONTROL is the local algorithm, please fill in the rows of “metering level”, “occupancy threshold” and “metering cycle” and leave rows of “desired occupancy” and “regulator” blank.

The following is an example of the use of OCCUPANCY_CONTROL as the local algorithm. The example of the use of ALINEA_CONTROL as the local algorithm is shown in last example.

```

...

local algorithm          OCCUPANCY_CONTROL

...

total number of entrance ramps is 7

ramp                    33
mainline detector      405n0.93ml
queue detector         405n0.93orspill
on-ramp detector       405n0.93orb
HOV                    0
control type           1
rate restriction       240 900
desired occupancy
regulator
metering level         6
occupancy threshold   0.15 0.17 0.20 0.25 0.40
metering cycle         4.0 5.0 7.0 9.0 12.0 15.0

...

```

3.3 Loading plugin

The names of this plugin files are:

```

bottleneck.dll: Modeller Plugin
bottleneck-p.dll: Processor Plugin

```

The BOTTLENCK plugin depends on other two plugins, ramp metering control and loop data aggregator. These two plugins should be specified earlier than this plugin in the “plugins” or “programming” file, i.e.:

```

loop_agg.dll
ramp_controller.dll
bottleneck_controller.dll

```

If you want to implement BOTTLENCK with another queue override strategy, which can be implemented by the on-ramp queue control plugin, you will need to leave the “queue detector” line in the “bottleneck_control” file blank (which disables the internal queue

override strategy of BOTTLENECK). Then you will need to add on-ramp queue control plugin to the “plugins” or “programming” file with the following sequence in order to give the on-ramp queue control strategy higher priority than BOTTLENECK.

```

loop_agg.dll
ramp_controller.dll
queue_control.dll
bottleneck_controller.dll

```

In addition, in order to correctly load and run this plugin, please satisfy the following requirements:

(1) For ramp metering control plugin: on-ramp signals controlled by the BOTTLENECK algorithm should be specified in the “ramp_control” file. For example, the correspondent on-ramp signal 33 needs to be specified in the “ramp_control” file:

```

on-ramp signal      33
name                405N & ICD 1 @ 0.93
demand detector     405n0.93orb
number of control plans  2
from 6:0 to 9:0      METER_ON with 1 veh per 6 sec
from 15:0 to 19:0    METER_ON with 1 veh per 6 sec

```

(2) For the loop data aggregator plugin: all loops involved in the “bottleneck_control” file should be specified in the “loop_control” file for aggregated data collection. In addition, the “report cycle” in the “loop_control” file should be the same as the “metering rate update interval” specified in the second row of “bottleneck_control” file. For example, all relevant loops of ramp 33 needs to be specified in the “loop_control” file:

```

detector count      42
report cycle       30
activation time      06:00:00
deactivation time    10:00:00
gather smoothed data no
output to files      yes

```

...

```

name 405n0.93ml
gather interval 00:00:30

```

```

name 405n0.93orb
gather interval 00:00:30

```

...

3.4 Output file

If “report metering rate” in the “bottleneck_control” file is specified as “yes”, the metering information of all BOTTLENECK controlled ramps will be output (every update cycle) to a file named “moe-BOTTLENECK.txt”. It can be found in the subdirectory:

network/Log/run-xxx

where network is the name of the current working directory, and xxx is a three-digit sequence number.

3.5 Error checking

If any mistakes happened in the “bottleneck_control” file or the other two supporting plugins, i.e. loop data aggregator and ramp metering control, this plugin will be disabled. The report window of PARAMICS will show whether this plugin is working.

Through enabling the option: “checking control file” in the “bottleneck_control” file, you can check if there is any error in the “bottleneck_control” file.

3.6 Calibration of the BOTTLENECK algorithm

In order to maximize the performance of ALINEA metering control, parameters of the ALINEA algorithm need to be calibrated and optimized. Basically, the calibration of the BOTTLENECK algorithm involves five aspects:

- (1) Calibration of local algorithm. If ALINEA is used, its calibration requirements have been described in Section 6; if occupancy control is used, its calibration involves the analysis the allowed traffic flow from on-ramps and mainline occupancy based on a plot of historical volume-occupancy data collected at its correspondent mainline loop detector station;
- (2) Defining each section in the study network, basically the stretch of freeway between two adjacent mainline loop detector stations. The typical spacing of two adjacent loop detector stations is ½ to 1 mile;
- (3) Calibrating the threshold occupancy of the downstream detector station of each section based on the volume-occupancy diagram at the downstream loop detector station of each section. The typical threshold occupancy is the occupancy at capacity;

- (4) Defining the area of influence of each section (including several upstream on-ramps that are responsible for the volume reduction), according to how far it is to the downstream detector station and the historical demand level of the on-ramp. A typical definition is that entrance ramps in the area of influence should be within a maximum distance of two miles to the location of the downstream detector of the section;
- (5) Defining the weighting factor of each on-ramp in the area of influence of each section, based on historical demands pattern.

4 Technical supports

4.1 Release notes

Compare to the plugin used in PARAMICS version 3, this plugin has the following modification:

1. In the “bottleneck_control” file, the third row, “update cycle of metering rate”, is changed to “metering rate update interval”.

4.2 Contact information

Any comments and suggestions are welcome. Please contact us at the email address: lchu@translab.its.uci.edu