Appendix B. Data Reduction Procedures for Traffic Signal Systems Performance Measures

Attached document:

Data Reduction Procedures for Traffic Signal Systems Performance Measures

by

Joseph M. Ernst
Purdue University

Christopher M. Day
Purdue University

Darcy M. Bullock
Purdue University

Working Paper
August 31, 2011
INTRODUCTION

The data reduction procedures developed for this project develop a framework to link the INDOT database to a Google Maps enabled webpage that display NCHRP 3-79a performance measure graphics\(^1\). This webpage uses queries that exist as text files in a folder structure. The website code is structured so that new queries added to a folder will automatically be added to the website with no alterations to the rest of the website code. This allows website development, database development, and performance measure query development to be executed independently. A screenshot of the website built on this framework is shown in Figure 1.

\[\text{Figure 1 Screenshot of performance measures website.}\]

\(^1\) This paper focuses on implementation of the performance measures in dashboard. Technical details regarding the development of the performance measures is extensively documented in prior publications by the research team (1,2,3,4,5,6,7,8,9).
This following sections discuss the INDOT performance measure database infrastructure, describes each of the queries that have been developed, and then shares some conclusions including the lessons learned when developing the queries.

**DATABASE STRUCTURE**

The Indiana Department of Transportation (INDOT) maintains a database that stores information about deployed devices and data. This database includes the data used to generate the performance measures described in this section, but also includes many other tables. This section begins by describing the database tables that involved in the performance measures and then documents the performance measures that have been developed.

![Figure 2 INDOT_Signal_Systems partial database diagram](image-url)

**Figure 2 INDOT_Signal_Systems partial database diagram**
The database diagram for the tables involved in the performance measure queries is shown in Figure 2. The “Controller_Event_Log” table is the central table for performance measures because it stores all of the collected data. The other tables linked to this table provide text based descriptions of the enumerated data items (Table 1). This structure allows the “Controller_Event_Log” to stay relatively small for the amount of data that it stores. It is important that each data point in the “Controller_Event_Log” table be as small as possible because it currently holds over 1.7 billion rows of data and is growing every day.

The “Controller_Event_Log” has four columns:

- COMMISSION_NBR
- Event_Timestamp
- Event_Code_ID
- Event_Param

The “COMMISSION_NBR” links to the “SIGNALS” table. This indicates from which intersection the data has been collected. As one would expect, the “Event_Timestamp” records the time for each data point. The “Event_Code_ID” column holds an integer that represents a type of event relating to an intersection phase or detector. These events are translated by linking to the “Event_Code_LK” table shown in Table 1. The “Event_Code_IDS” that are not used in the performance measures in this section have a gray background. The “Event_Param” column can mean one of two different things. For events that are associated with an intersection phase, the “Event_Param” is the phase number. For events that are associated with a detector, the “Event_Param” is an integer that links to the “DETECTOR_NBR” column of the “DETECTORS” table.
<table>
<thead>
<tr>
<th>Event Code ID</th>
<th>Event Name</th>
<th>Event Code ID</th>
<th>Event Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Phase Off</td>
<td>42</td>
<td>Offset Length Change</td>
</tr>
<tr>
<td>1</td>
<td>Phase Green</td>
<td>43</td>
<td>Split 1 Change</td>
</tr>
<tr>
<td>2</td>
<td>Phase Yellow</td>
<td>44</td>
<td>Split 2 Change</td>
</tr>
<tr>
<td>3</td>
<td>Phase Red Clear</td>
<td>45</td>
<td>Split 3 Change</td>
</tr>
<tr>
<td>4</td>
<td>Ped Off</td>
<td>46</td>
<td>Split 4 Change</td>
</tr>
<tr>
<td>5</td>
<td>Ped Walk</td>
<td>47</td>
<td>Split 5 Change</td>
</tr>
<tr>
<td>6</td>
<td>Ped Clear</td>
<td>48</td>
<td>Split 6 Change</td>
</tr>
<tr>
<td>8</td>
<td>Detector On</td>
<td>49</td>
<td>Split 7 Change</td>
</tr>
<tr>
<td>9</td>
<td>Detector Off</td>
<td>50</td>
<td>Split 8 Change</td>
</tr>
<tr>
<td>10</td>
<td>Detector Failed</td>
<td>51</td>
<td>Split 9 Change</td>
</tr>
<tr>
<td>11</td>
<td>Detector Restored</td>
<td>52</td>
<td>Split 10 Change</td>
</tr>
<tr>
<td>12</td>
<td>Overlap Off</td>
<td>53</td>
<td>Split 11 Change</td>
</tr>
<tr>
<td>13</td>
<td>Overlap Green</td>
<td>54</td>
<td>Split 12 Change</td>
</tr>
<tr>
<td>14</td>
<td>Overlap Green</td>
<td>55</td>
<td>Split 13 Change</td>
</tr>
<tr>
<td></td>
<td>Extension</td>
<td>56</td>
<td>Split 14 Change</td>
</tr>
<tr>
<td>15</td>
<td>Overlap Yellow</td>
<td>57</td>
<td>Split 15 Change</td>
</tr>
<tr>
<td>16</td>
<td>Overlap Red Clear</td>
<td>58</td>
<td>Split 16 Change</td>
</tr>
<tr>
<td>17</td>
<td>Preempt Active</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Preempt Off</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Preempt Active</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Preempt Off</td>
<td>62</td>
<td>Coord cycle state change</td>
</tr>
<tr>
<td>24</td>
<td>Phase Hold Active</td>
<td>63</td>
<td>Coord phase yield point</td>
</tr>
<tr>
<td>25</td>
<td>Phase Hold Released</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Ped Call on Phase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Ped Call Cleared</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Phase Min Complete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Phase Term Gap Out</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Phase Term Max Out</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Phase Term Force Off</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Coord Pattern Change</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>Cycle Length Change</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0  Grayed out cells are not used in performance measures described in this section.
The “DETECTOR_PROFILES” table is used to link a detector to an intersection for a given time range. Since detection configurations change, every detector configuration must be saved so that data collected from that configuration can still be interpreted. The time range in the “DETECTOR_PROFILES” table helps to know which set of detectors was active for the data being analyzed. The last table is the systems table. This groups intersections into systems that could be coordinated. When making changes to intersections based upon the performance measures, it is necessary to look at how those changes will affect the rest of the system.

**DEFINITIONS OF PHASE INSTANCE AND CYCLE**

Many of the performance measures described in this section are calculated on a cycle by cycle basis. This requires the definition of a cycle boundary\(^2\). This section begins with an illustrative example to help describe why the definition of cycle length can be challenging. A discussion of the different definitions and the choice of cycle boundary definition follows.

The illustrative example is for a standard 8 phase intersection intersection with the phases defined in the ring diagram in Figure 3. The illustrative example focuses on the top ring (Phases: 1,2,3, and 4).

---

\(^2\) For an extended discussion of cycle definitions, refer to Day et al. (1)
Figure 4 shows the signal head states for the phases in the top ring for two different configurations of the intersection. The diagram is red when the signal head for the corresponding phase is red, green when it is green, and gray for the yellow/all red interval. The yellow and all red have been grouped to simplify the diagram. Figure 4a shows the phase diagram for a coordinated system. The signal head states are completely determined by the split timings. For this example a 15% split is used for the left turn movements (phases 1 and 3), a 20% split is used for the cross movement (phase 4) and a 50% split is used for the coordinated movement (phase 2). In this case there is no problem measuring the cycle length because any reasonable definition of cycle length will produce 100 seconds. The phase 2 beginning of red cycle length (CL\textsubscript{BOR2}) and the phase 2 beginning of green cycle length (CL\textsubscript{BOG2}) are shown as two examples.

Figure 4b shows the same diagram except that the intersection is actuated/coordinated, which means that it is coordinated with split timings, but will also respond to the information from the vehicle detectors. In this example, there are only two changes from the coordinated diagram: both phase 1 and phase 2 gap out early in cycle 1. In this case the cycle length cannot be determined by measuring the time between successive changes in signal head states. Two examples are shown where the CL\textsubscript{BOR2} is 110 seconds and the CL\textsubscript{BOG2} is 105 seconds.
Figure 4: Illustrative example of the complication of cycle length definitions.
With the illustrative example in mind, the options for defining cycle boundaries are now discussed. It is desirable to define performance measures using changes that are visible at the intersection. This is however in contradiction with the desire for the measured cycle length to be the same as the cycle length that is programmed in the controller.

Examples of cycle length calculations that use the changes that are visible at the intersection are limited to using the beginning or end of the red or green state of a given phase. The two examples shown in Figure 4 use the beginning of the green and red states of phase 2 (the coordinated phase in this example). There are also several choices for determining the phase from information reported in the “Controller_Event_Log”. One example is the “Cycle Length Change” event (Event Code ID 41). Every time that the controller changes the cycle length that is being used, it reports this event. Another option is to use the “Coord phase yield point” event (Event Code ID 63). This event is logged once per cycle for each coordinated phase and these events are spaced at exactly the programmed cycle length. Some controllers are also configured to report a “Cycle Length Change” event at midnight every night.

The decision of which cycle boundary to use is specific to each performance measure. For performance measures that make sense with either definition, the visible change of the signal head state is usually chosen. This choice is preferred because it available externally from the controller and because it is available for controllers that are not running coordinated patterns.
PERFORMANCE MEASURES

The performance measures developed in this project have been developed as a suite of database queries that generate plots on a website. Each of these queries and the code used to produce the images is presented in this section. A screenshot of the overall website is shown in Figure 1. The queries shown in this section use $\text{NBR}$ to represent the intersection’s “COMMISSION\_NBR” and $\text{DATE}$ to represent the date of the data selected in the following format: YYYY-MM-DD.

Communication and Detector Health

The first performance measures that must be evaluated is the health of the sensor network. If there is missing data from the system it will be difficult to determine the validity of any of the other performance measures. A first order test of system health is if the database server can communicate with the controllers. This can be accomplished through a simple “ping” test. The script that runs this test queries the database for a list of ip addresses and then attempts to ping each controller. Successful results of ping tests are stored in the database. The script that executes the ping test is shown here:

```bash
#!/bin/bash

psql=/usr/bin/psql
db=INDOT\_Signal\_Systems
nmap=/usr/bin/nmap
ping=/bin/ping
folder=/home/jernst/datamaps/scripts
query='select "IP" from "SIGNALS" where "IP" is not null';

tstamp=`date +%Y-%m-%d %H:%M:%S`

echo $query | $psql -At $db | while read ip; do
    echo $ip
    pnum=`ping -c 10 $ip | sed -n "s/.*transmitted, *\([0-9]\)*/\1/p`
    nbr=`echo "select "COMMISSION\_NBR" from "SIGNALS" where "IP"='$ip'" | $psql -At $db`
    if [ "x$pnum" != "x0" ]; then
        echo "insert into health\_ping\_test (signal\_nbr,ip,ping\_time) values ('$nbr','$ip','$tstamp')" | $psql $db
    fi
done
```
A query to see when the most recent ping test was completed is shown here, where $$\text{maxtime}$$ is the most recent attempt at a ping test and $$\text{COMMISSION_NBR}$$ is the identifier for the controller being queried:

```sql
select
    max(h1.ping_time) as ping_time,
    date_part('epoch','$$\text{maxtime}$$'-max(h1.ping_time)) as diff
from
    health_ping_test as h1
where
    h1.signal_nbr='$$\text{COMMISSION_NBR}$$'
group by
    h1.ping_time,h1.signal_nbr
order by diff"
```

A second health test finds the most recent data that has been inserted from each controller. This makes sure that the controller is not only communicating, but that it is producing data that is getting inserted correctly into the database. In this case a query is executed to find the most recent data from each controller. This is compared to the most recent system wide data. The following query finds the most recent data inserted from a given controller.

```sql
select
    max("Event_Timestamp") as tstamp
from
    "Controller_Event_Log"
where
    "COMMISSION_NBR" = ".".".$\text{siginfo}'nbr'.""
Cycle Length

The previous section discussed the complication of calculating the cycle length. In order to make sure that all of the cycle length data is being reported and is consistent, the cycle length is calculated in several different ways. One way is to take the difference in time between consecutive “Coord Phase Yeild Point” events. These are spaced at the programmed cycle length and are reported for each coordinated phase. An example query is shown here:

```sql
select
    mod(date_part('epoch',log1."Event_Timestamp")::integer-5*3600,3600*24)::float/3600,
    date_part('epoch',min(log2."Event_Timestamp")-log1."Event_Timestamp")
as cycle_length
from
    "Controller_Event_Log" as log1,
    "Controller_Event_Log" as log2
where
    log1."COMMISSION_NBR" ='$NBR$' and
    log1."Event_Code_ID" =63 and
    log1."Event_Timestamp" > '$DATE$' and
    log1."Event_Timestamp" < '$DATE$::timestamp + '1 day'::interval
    and
    log2."COMMISSION_NBR" ='$NBR$' and
    log2."Event_Code_ID" =63 and
    log2."Event_Timestamp" > '$DATE$' and
    log1."Event_Param"=log2."Event_Param" and
    log2."Event_Timestamp">log1."Event_Timestamp"
group by
    log1."Event_Timestamp"
order by
    log1."Event_Timestamp"
```

Another way to find the cycle length is to query for the “Cycle Length Change” event in the “Controller_Event_Log”. This query is shown here:

```sql
select
    mod(date_part('epoch',log1."Event_Timestamp")::integer-5*3600,3600*24)::float/3600,
    log1."Event_Param"
from
    "Controller_Event_Log" as log1
where
    log1."COMMISSION_NBR" ='$NBR$' and
    log1."Event_Code_ID" =41 and
    log1."Event_Timestamp" > '$DATE$' and
    log1."Event_Timestamp" < '$DATE$::timestamp + '1 day'::interval
order by
    log1."Event_Timestamp"
```
Another way to interpret the cycle length is to take the difference between consecutive ending times for the green state of a given phase. This will not always give the programmed cycle length, but can be useful in gaining information about the variability of the effective cycle length. An example query is shown here for phase 2:

```
select 
  mod(date_part('epoch',log1."Event_Timestamp")::integer-5*3600,3600*24)::float/3600,
  date_part('epoch',min(log2."Event_Timestamp")-log1."Event_Timestamp")
  as cycle_length
from 
  "Controller_Event_Log" as log1,
  "Controller_Event_Log" as log2
where 
  log1."COMMISSION_NBR"='$$NBR$$' and 
  log1."Event_Code_ID"=2 and 
  log1."Event_Param"=2 and 
  log1."Event_Timestamp" > '$$DATE$$' and 
  log1."Event_Timestamp" < '$$DATE$$':timestamp + '1 day':interval 
  and 
  log2."COMMISSION_NBR"='$$NBR$$' and 
  log2."Event_Code_ID"=2 and 
  log2."Event_Param"=2 and 
  log1."Event_Param"=log2."Event_Param" and 
  log2."Event_Timestamp">log1."Event_Timestamp" and 
  log2."Event_Timestamp"<log1."Event_Timestamp"+'1 second':interval*300
  group by 
  log1."Event_Timestamp"
order by 
  log1."Event_Timestamp"
```

A more complicated query can be used to calculate the amount of time between ring barriers. This query first creates a temporary table and then uses this temporary table to calculate the final result. This query is shown here:

```
create temp view p1256 as 
(
  select 
    max(log1."Event_Timestamp") as t 
  from 
    "Controller_Event_Log" as log4, 
    "Controller_Event_Log" as log1
  where 
    log4."COMMISSION_NBR"='$$NBR$$' and 
    log1."COMMISSION_NBR"='$$NBR$$' and 
    log4."Event_Code_ID"=2 and 
    log1."Event_Code_ID"=2 and 
    log4."Event_Param"=4 and 
    log1."Event_Param" in (1,2,5,6) and 
```

select
  mod(date_part('epoch',peog.t)::integer-5*3600,3600*24)::float/3600,
  date_part('epoch',min(eog.t)-peog.t) as cycle_length
from
  p1256 as peog,
  p1256 as eog
where
  eog.t>peog.t and
  eog.t<peog.t + '1 second'::interval*300
group by
  peog.t
order by
  peog.t

Volumes

One useful value is the cycle-by-cycle volume of a given phase at an intersection. This volume is helpful for understanding the demand on an intersection. These numbers can also be used in a V/C calculation. This query is somewhat complicated because all of the detectors for a given phase must be indentified and included in the query. The query for the cycle-by-cycle volumes is shown here:

```sql
select
  (t1-4*3600-floor(((t1-4*3600)/3600/24)*3600*24)/3600,
  count(*)
from
  (select
    date_part('epoch',max(t1."Event_Timestamp") as t1,
    date_part('epoch',t2."Event_Timestamp") + det."Seconds_To_StopBar" as t2
  from
    "Controller_Event_Log" as t1,
    "Controller_Event_Log" as t2,
  ( select case
        when"Seconds_To_StopBar" is NULL then '5'
        else "Seconds_To_StopBar"
        end as det
  ) )
```
end as "Seconds_To_StopBar",
"DETECTOR_ID",
"DETECTOR_PROFILE_ID",
"PHASE",
"DIRECTION",
"DETECTOR_NBR"
from
"DETECTORS"
) as det,

(
select
"DETECTOR_PROFILE_ID" as id,
"COMMISSION_NBR" as commission_nbr,
"EFFECTIVE_DATE" as ed,
case
when "REMOVED_DATE" IS NULL then 'now':text::timestamp without
time zone
else "REMOVED_DATE"
end as rd
from
"DETECTOR_PROFILES" as dp1
where
"COMMISSION_NBR"='$$NBR$$'
) as dp
where
'$$DATE$$':timestamp+'1 hour':interval*12 between dp.ed and dp.rd
and
t1."COMMISSION_NBR"='$$NBR$$' and
t1."Event_Timestamp" > '$$DATE$$' and
t1."Event_Timestamp" < '$$DATE$$':timestamp+'1 day':interval and
t2."COMMISSION_NBR"='$$NBR$$' and
t2."Event_Timestamp" > '$$DATE$$' and
t2."Event_Timestamp" < '$$DATE$$':timestamp+'1 day':interval and
dp.id=det."DETECTOR_PROFILE_ID" and -- use correct detectors
det."PHASE"='1' and -- filter by intersection phase
t2."Event_Code_ID"='9' and -- detector_on
t2."Event_Param"=det."DETECTOR_NBR" and -- filter log by
detector_nbr
t1."Event_Code_ID"='2' and -- end of green
t1."Event_Param" = '1' and
t2."Event_Timestamp"> t1."Event_Timestamp" and -- t2>t1
t2."Event_Timestamp"< t1."Event_Timestamp"+'1 second':interval*300

group by
date_part('epoch',t2."Event_Timestamp") + det."Seconds_To_StopBar"
) as t

group by
t.t1
order by
t.t1
Green Time and Capacity

To determine the amount of green time in each cycle, each time that the light turns green is subtracted from the next time that the light turns red. An example of the green time query is shown here for phase 2.

```sql
select
  mod(date_part('epoch',log1."Event_Timestamp")::integer-
  5*3600,3600*24)::float/3600,
  date_part('epoch',min(log2."Event_Timestamp")-log1."Event_Timestamp")
as cycle_length
from
  "Controller_Event_Log" as log1,
  "Controller_Event_Log" as log2
where
  log1."COMMISSION_NBR" ='$$NBR$$' and
  log1."Event_Code_ID" =1 and
  log1."Event_Param"=2 and
  log1."Event_Timestamp" > '$$DATE$$' and
  log1."Event_Timestamp" < '$$DATE$$::timestamp + '1 day'::interval
and
  log2."COMMISSION_NBR" ='$$NBR$$' and
  log2."Event_Code_ID" =2 and
  log2."Event_Param"=2 and
  log2."Event_Timestamp"=log1."Event_Param" and
  log2."Event_Timestamp">log1."Event_Timestamp" and
  log2."Event_Timestamp"<log1."Event_Timestamp"+1 second':::interval*300
group by
  log1."Event_Timestamp"
order by
  log1."Event_Timestamp"
```

The capacity query is similar to the greentime query except that the green time must be multiplied by the number of lanes and the saturation flow rate. The following query uses 1900 vehicles per hour for the saturation flow rate and queries the “DETECTOR_LIST_VW” view for the number of thru lanes. These lanes are scaled by the estimate of the percentage of “thru” vehicle in each lane. The following query finds the cycle by cycle capacity for phase 2.

```sql
select
  mod(date_part('epoch',log1."Event_Timestamp")::integer-
  5*3600,3600*24)::float/3600,
  date_part('epoch',min(log2."Event_Timestamp")-
  log1."Event_Timestamp")*1900/3600*num.num as cycle_length
from
  "Controller_Event_Log" as log1,
  "Controller_Event_Log" as log2,
  (
    select
      ...) as num
```
sum("LANES"**"THRU_PCT"/100) as num from "DETECTOR_LIST_VW" where "COMMISSION_NBR"='$$NBR$$' and '$$DATE$$':date="EFFECTIVE_DATE" and '$$DATE$$':date="LAST_DATE" and "PHASE"='2'
) as num

where

log1."COMMISSION_NBR" ='$$NBR$$' and
log1."Event_Code_ID" = 1 and
log1."Event_Param"=2 and
log1."Event_Timestamp" > '$$DATE$$' and
log1."Event_Timestamp" < '$$DATE$$':timestamp + '1 day':interval

and

log2."COMMISSION_NBR" ='$$NBR$$' and
log2."Event_Code_ID" = 2 and
log2."Event_Param"=2 and
log1."Event_Param"=log2."Event_Param" and
log2."Event_Timestamp">log1."Event_Timestamp" and
log2."Event_Timestamp"<log1."Event_Timestamp"+'1 second':interval*300
group by

log1."Event_Timestamp",
num.num

order by

log1."Event_Timestamp"

**Volume-to-Capacity Ratio**

The volume to capacity ratio query follows from the cycle-by-cycle volume and cycle-by-cycle capacity queries discussed above. An example of this query for phase 2 is shown here:

```sql
select
  t,
  volume/(green_time*1900/3600*num) as vc
from

(

select
  mod(date_part('epoch',t1)::integer-5*3600,3600*24)::float/3600 as t,
  date_part('epoch',t2-t1) as green_time,
  max(num.num) as num,
  count(*) as volume
from

  (select
   sum("LANES") as num
from

  "Detector_List_VW"
where

  "COMMISSION_NBR"='$$NBR$$' and
```
"PHASE"='2' and
"Seconds_To_StopBar"='-1'
) as num,

) select
log1."Event_Timestamp" as t1,
min(log2."Event_Timestamp") as t2
from
"Controller_Event_Log" as log1,
"Controller_Event_Log" as log2
where
log1."COMMISSION_NBR" = '$$NBR$$' and
log1."Event_Code_ID" = 1 and
log1."Event_Param"=2 and
log1."Event_Timestamp" > '$$DATE$$' and
log1."Event_Timestamp" < '$$DATE$$'::timestamp + '1 day'::interval
and
log2."COMMISSION_NBR" = '$$NBR$$' and
log2."Event_Code_ID" = 2 and
log2."Event_Param"=2 and
log1."Event_Param"=log2."Event_Param" and
log2."Event_Timestamp">log1."Event_Timestamp" and
log2."Event_Timestamp"<log1."Event_Timestamp"+'1 second'::interval*300

) as temp,
"Detector_List_VW" as det,
"Controller_Event_Log" as log3
where
det."PHASE"='2' and
det."COMMISSION_NBR"='$$NBR$$' and
det."Seconds_To_StopBar"='-1' and
log3."COMMISSION_NBR"='$$NBR$$' and
log3."Event_Code_ID"='9' and
log3."Event_Param"=det."DETECTOR_NBR" and
log3."Event_Timestamp">temp.t1 and
log3."Event_Timestamp"<temp.t2

) as temp
Purdue Coordination Diagram (PCD)

The Purdue Coordination Diagram (PCD) allows all vehicle arrivals to be viewed in one image that represents all of the progression data throughout the day. This query also shows the beginning of the green interval and the beginning of the red interval. This means that there are two separate queries: detection data points and signal head state. Because of the size of the database, temporary tables are also created in the process of generating the data for each query. The purpose of the temporary tables is to filter the enormous “Controller_Event_Log” and to create a table with just the time range and data types of interest.

There are also several minor metadata queries that must be run to figure out whether the primary movement for the intersection is North-South or East-West and to find which phase is associated with each. These queries are shown here.

North/East Direction:

```sql
select case
    when "DIRECTION"='N' then 'North'
    when "DIRECTION"='E' then 'East'
end
from
(select distinct det."DIRECTION"
from "Detector_List_VW" as det
where
'$$DATE$$':timestamp between det."EFFECTIVE_DATE" and det."LAST_DATE" and det."LANE" not in ('L','R') and det."DIRECTION" in ('N','E') and det."PHASE" in ('2','6') and det."COMMISSION_NBR"='$$NBR$$'
) as temp
```

North/East Phase:

```sql
select distinct det."PHASE"
from
"Detector_List_VW" as det
where
```
South/West Direction:

```sql
select
  case
    when "DIRECTION"='S'
    then 'South'
    when "DIRECTION"='W'
    then 'West'
  end
from
  (select distinct
    det."DIRECTION"
  from
    "Detector_List_VW" as det
  where
    '$$DATE$$'::timestamp between det."EFFECTIVE_DATE" and det."LAST_DATE" and det."LANE" not in ('L','R') and det."DIRECTION" in('S','W') and det."PHASE" in ('2','6') and det."COMMISSION_NBR"='$$NBR$$') as temp
```

South/West Phase:

```sql
select distinct
  det."PHASE"
from
  "Detector_List_VW" as det
where
  '$$DATE$$'::timestamp between det."EFFECTIVE_DATE" and det."LAST_DATE" and det."LANE" not in ('L','R') and det."DIRECTION" in('S','W') and det."PHASE" in ('2','6') and det."COMMISSION_NBR"='$$NBR$$'
```

The query for the South/West signal state information is shown here. The North/East signal state information is similar.

```sql
select
  date_part('epoch', log."Event_Timestamp") as t
from
  temp table temp1
```
"Controller_Event_Log" as log
where
log."COMMISSION_NBR"='$$NBR$$' and
log."Event_Timestamp" > '$$DATE$$' and
log."Event_Timestamp" < '$$DATE$$':timestamp+'1 day':interval and
log."Event_Code_ID"='2' and
log."Event_Param"='$$SWphase$$'
order by t
;
create index temp1_ind on temp1(t);

select
date_part('epoch',log."Event_Timestamp") as t
into
temp table temp2
from
"Controller_Event_Log" as log
where
log."COMMISSION_NBR"='$$NBR$$' and
log."Event_Timestamp" > '$$DATE$$' and
log."Event_Timestamp" < '$$DATE$$':timestamp+'1 day':interval and
log."Event_Code_ID"='1' and
log."Event_Param"='$$SWphase$$'
order by t
;
create index temp2_ind on temp2(t);

select
(t1-4*3600-floor((t1-4*3600)/3600/24)*3600*24)/3600,
t2-t1,
t3-t1
from
(
select
t1.t as t1,
min(t2.t) as t2,
min(t3.t) as t3
from
temp1 as t1,
temp2 as t2,
temp3 as t3
where
  t2.t> t1.t and
  t3.t> t1.t and
  t2.t< t1.t+300 and
  t3.t< t1.t+300
  group by
    t1.t
) as t
order by
t.t1
The query for the South/West detector data is shown here.

```sql
set constraint_exclusion=on;
set enable_bitmapscan=off;

select
date_part('epoch',log."Event_Timestamp") as t
into
temp table temp1
from
"Controller_Event_Log" as log
where
log."COMMISSION_NBR"='$$NBR$$' and
log."Event_Timestamp" > '$$DATE$$' and
log."Event_Timestamp" < '$$DATE$$'::timestamp+'1 day'::interval and
log."Event_Code_ID"='2' and
log."Event_Param"='$$SWphase$$'
order by t
;
create index temp1_ind on temp1(t);

select
date_part('epoch',log."Event_Timestamp") as t
into
temp table temp2
from
"Controller_Event_Log" as log,
"Detector_List_VW" as det
where
log."COMMISSION_NBR"='$$NBR$$' and
log."Event_Timestamp" > '$$DATE$$' and
log."Event_Timestamp" < '$$DATE$$'::timestamp+'1 day'::interval and
log."Event_Code_ID"='9' and
det."COMMISSION_NBR"=log."COMMISSION_NBR" and
det."PHASE"='$$SWphase$$' and
det."Seconds_To_StopBar" > 0 and
log."Event_Param"=det."DETECTOR_NBR"
order by t;
create index temp2_ind on temp2(t);

select
(t2-4*3600-floor((t2-4*3600)/3600/24)*3600*24)/3600,
t2-t1
from
{
select
 max(temp1.t) as t1,
temp2.t as t2
from
 temp1,
 temp2
where
 temp2.t>temp1.t and
}````
temp2.t<temp1.t+300
group by
temp2.t
) as t
order by
(t2-4*3600-floor((t2-4*3600)/3600/24)*3600*24)/3600;

Degree of Intersection Saturation

The degree of intersection saturation is different from most of the rest of the queries in that it is not calculated per phase. As indicated by the name, it reflects the saturation of the intersection as a whole. This query uses many of the concepts developed other queries. The cycle boundary for this query must be found in terms of a ring diagram. Therefore the boundary is at the end of phase 2 and phase 6.

The query is very complex and achieves reasonable speed by creating a series of temporary tables. This set of queries is shown here and the purpose of each temporary table is described.

The “temp_eog” temporary table selects the data for the correct day and intersection corresponding to the end of a green phase. This small chunk of the “Controller_Event_Log” is stored in a temporary table and indexed so that it can be searched quickly. The database option “constrain_exclusion” is turned on to take advantage of the “Controller_Event_Log” table partitioning and the “enable_bitmapscan” is disabled to encourage the queries to use the existing indexes on the “Controller_Event_Log” table partitions.

```
set constraint_exclusion=on;
set enable_bitmapscan=off;

select
  "Event_Param" as phase,
  date_part('epoch',log."Event_Timestamp") as t
into
temp table temp_eog
from
  "Controller_Event_Log" as log
where
  log."COMMISSION_NBR"='$$NBR$$' and
  log."Event_Timestamp" > '$$DATE$$' and
  log."Event_Timestamp" < '$$DATE$$'::timestamp+'1 day'::interval and
  log."Event_Code_ID" = '2';

create index temp_eog_ind on temp_eog(phase,t);
```
The “temp_bog” table is identical to the “temp_eog” table except that it stores the information for the beginning of green events instead of the end of green events. The query to create this table is shown here:

```sql
select
    "Event_Param" as phase,
    date_part('epoch',log."Event_Timestamp") as t
into
    temp table temp_bog
from
    "Controller_Event_Log" as log
where
    log."COMMISSION_NBR"='$$NBR$$' and
    log."Event_Timestamp" > '$$DATE$$' and
    log."Event_Timestamp" < '$$DATE$$'::timestamp+'1 day'::interval and
    log."Event_Code_ID" = '1'
;
create index temp_bog_ind on temp_bog(phase,t);
```

The next temporary table is the “temp_detections” table. This table includes the phase and time of all vehicle detections for the specified date and intersection. The query to create this temporary table is shown here:

```sql
select
    det."PHASE" as phase,
    date_part('epoch',log."Event_Timestamp") as t
into
    temp table temp_detections
from
    "Controller_Event_Log" as log,
    "Detector_List_VW" as det
where
    log."COMMISSION_NBR"='$$NBR$$' and
    log."Event_Timestamp" > '$$DATE$$' and
    log."Event_Timestamp" < '$$DATE$$'::timestamp+'1 day'::interval and
    log."Event_Code_ID" = '9' and
    det."COMMISSION_NBR"=log."COMMISSION_NBR" and
    log."Event_Param"=det."DETECTOR_NBR"
order by t
;
create index temp_detections_ind on temp_detections(phase,t);
```

The next temporary table splits time by the beginning of the green interval for phase 2. This is not the cycle boundary that we are interested in; however, there will be one such cycle boundary
between each pair of \( t_{\text{start}} \) and \( t_{\text{stop}} \) times in the temporary table created by this query. This temporary table is called temp\_ss2. The two is meant to indicate that it uses the phase two data.

```sql
select
    log1.t as t_start,
    min(log2.t) as t_stop
into
temp table temp\_ss2
from
temp\_bog as log1,
temp\_bog as log2
where
    log2.t > log1.t and
    log2.t < log1.t +500 and
    log2.phase=2 and
    log2.phase=2
order by
    log1.t;
```

Using this table the ring barrer can be found. This is accomplished by finding the maximum end of green time for phase two and phase six between each of the phase beginning of green times found in the previous query. The query to find the ring barrier is shown here:

```sql
select
    case
        when log2.t > log6.t then log2.t
        else log6.t
    end as t
into
temp table boundary
from
temp\_ss2 as ss,
temp\_eog as log2,
temp\_eog as log6
where
    log2.phase=2 and
    log6.phase=6 and
    log2.t > ss.t_start and log2.t < ss.t_stop and
    log6.t > ss.t_start and log6.t < ss.t_stop
order by
    ss.t_start
;
```
The list of ring barrier times is then reformatted into a beginning and end of cycle format. This query is shown below and creates the “boundary_ss” temporary table.

```sql
select
    log1.t as t_start,
    min(log2.t) as t_stop
into
temp table boundary_ss
from
    boundary as log1,
    boundary as log2
where
    log2.t > log1.t and
    log2.t < log1.t +500
group by
    log1.t
order by
    log1.t;
```

To calculate the capacity for the v/c calculation, the number of lanes associated with the detectors in each phase must be calculated. This association between the phase and the number of effective lanes is shown below. Note that if two sensors are in the same lane, this effectively doubles the apparent capacity of the lane if both sensors are being used in the volume calculation. The temporary table that associates the phase with an apparent number of lanes is called the “temp_lanes” temporary table.

```sql
select
    "PHASE" as phase,
    sum("LANES") as lanes
into temp table temp_lanes
from
    "Detector_List_VW" as det
where
    "EFFECTIVE_DATE"< '$$DATE$$' and
    "LAST_DATE">'$$DATE$$' and
    "COMMISSION_NBR"='$$NBR$$'
```
The “temp_cycle” temporary table defined below associates the beginning of each ring barrier cycle boundary with the start and end green time for a given phase. This query is shown here:

```sql
select
    ss.t_start,
    bog.phase as phase,
    bog.t as tbog,
    eog.t as teog
into temp table temp_cycle
from
    boundary_ss as ss,
    temp_bog as bog,
    temp_eog as eog
where
    bog.phase=eog.phase and
    bog.t > ss.t_start and bog.t <= ss.t_stop and
    eog.t > ss.t_start and eog.t <= ss.t_stop and
    eog.t > bog.t
order by
    ss.t_start;
```

This next query uses the previously created temporary tables to calculate the intersection saturation. It is a set of nested queries. The innermost query calculates the v/c for each phase in each cycle. The v/c values are then summed into v/c values for each ring barrier pair (e.g., the phase 1 v/c is added to the phase 2 v/c). The max v/c for each side of the ring barrier is then found. The the sum of the vc before and after the ring barrier is calculated. The time reported is the time of day in hours. This nested query is shown here:

```sql
select
    (t_start-4*3600-floor((t_start-4*3600)/3600/24)*3600*24)/3600,
    sum(maxrb_vc) as xc
from
    (select
        t_start,
        mod(rb::integer,2),
        max(rb_vc) as maxrb_vc
        from
            (select
                cycle.t_start,
                cycle.phase,
                count(*)/(1900*lanes.lanes*(teog-tbog)/3600) as vc
```
from
    temp_cycle as cycle,
    temp_lanes as lanes,
    temp_detections as det
where
cycle.phase::text=lanes.phase::text and
cycle.phase::text=det.phase::text and
det.t > cycle.tbog and det.t < cycle.teog

group by
cycle.t_start,
cycle.phase,
lanes.lanes,
teog,
tbog
order by
cycle.t_start,cycle.phase
) as t

group by t_start, floor((phase+1)/2)
order by t_start, floor((phase+1)/2)
) as t2

group by t_start, mod(rb::integer,2)
order by t_start
) as t3

group by t_start
order by t_start;
CONCLUSIONS

The framework developed to display performance measures on a website allow for rapid progression from the development of a query to making the resulting plots available. Many queries have been written for use with the INDOT database in order to create plots for the website. These queries have been detailed in this section.

During the development of this toolset, the size of the database size has increased and the rate of growth has also increased. Due to the size and projected size of this database, special considerations are required to make sure that the queries can be run quickly. One important step in this process is partitioning the “Controller_Event_Log” into one partition per month. This allows queries to quickly eliminate all other partitions from its query. In order to make sure that the database uses these partitions to increase the speed of the queries, the following line should be placed at the beginning of each query:

```
set constraint_exclusion=on;
```

Each of the partitions are also indexed. While developing the queries it has been noticed that the query planner is less likely to use an index search in combination with the constraint exclusion. Encouraging the query builder to use the index increases the speed of most queries to about five times the original speed. In order to do this, the following line of code should be placed at the beginning of each query:

```
set enable_bitmapscan=off;
```

Also queries that must reference the “Controller_Event_Log” more than once (which includes most of the queries documented in this section) should create a temporary table with only the relevant data from the relevant intersections and time range.

The queries developed in this section allow for quick analysis of the transportation network. The framework developed for this project will allow new queries to be quickly linked into the existing infrastructure. New queries should also run quickly, even as the database increases in size by following the query building guidelines in this section.
ACKNOWLEDGMENTS

This work was supported by the Joint Transportation Research Program administered by the Indiana Department of Transportation and Purdue University. The contents of this paper reflect the views of the authors, who are responsible for the facts and the accuracy of the data presented herein, and do not necessarily reflect the official views or policies of the sponsoring organizations. These contents do not constitute a standard, specification, or regulation.

REFERENCES


