Chapter Thirteen: Contents

(Mode Feedback – 10 December 2002– LA-UR 01-5716 – Portland Study Reports)

1. CONFIGURATION FILES 1

1.1 ALLSTR_ONDN_A-INIT.CFG 1
1.2 ALLSTR_ONDN_A-MF0.CFG 2
1.3 ALLSTR_ONDN_A-MF1.CFG 3
1.4 ALLSTR_ONDN_A-MF2.CFG 4
1.5 ALLSTR_ONDN_C-INIT.CFG 5
1.6 ALLSTR_ONDN_C-MF0.CFG 7
1.7 ALLSTR_ONDN_C-MF1.CFG 9
1.8 ALLSTR_ONDN_C-MF2.CFG 11
1.9 ALLSTR_ONDN-R-INIT.CFG 13
1.10 ALLSTR_ONDN_R-MF0.CFG 14
1.11 ALLSTR_ONDR_R-MF1.CFG 15
1.12 ALLSTR_ONDN_R-MF2.CFG 16

2. SCRIPTS 17

2.1 CALCMODECOSTS.PL 17
2.2 COMBINEDB.PL 23
2.3 FILTERONDN.PL 27
2.4 FIT.CSH 30
2.5 FIXMODES.PL 31
2.6 GUESSMODE.C 34
2.7 ITDBBYTOUR.PL 42
2.8 REMODEONDN.PL 54
2.9 REMODETRANS.PL 57
2.10 RUN.CSH 59
2.11 RUNCOLLATOR.CSH 61
2.12 RUNROUTER.PL 63
2.13 STRATIFYONDN.PL 65
2.14 TESTALPHA.PL 70

Chapter Thirteen—Mode Feedback

NOTE: Long code lines that do not fit completely on one line of this document are shown in italics and continued on to the next line.

1. CONFIGURATION FILES

1.1 allstr_ONDN_A-init.cfg

```
TRANSIMS_ROOT            /n/father/transims/CaseStudy3/scenarios/allstr

CONFIG_DEFAULT_FILE      $TRANSIMS_ROOT/allstr.cfg

############################################################
# recent corrections:

ACT_TRAVEL_TIMES_FILE              $TRANSIMS_ROOT/activity/traveltime_050101
ACT_TRAVEL_TIME_FUNCTION_MODES     1;2;3;4;5;6;7

ACT_SURVEY_ACTIVITY_FILE           $TRANSIMS_ROOT/data/survey_activities
ACT_SURVEY_WEIGHTS_FILE            $TRANSIMS_ROOT/data/survey_weights_25
#ACT_TRAVEL_TIME_INTERVALS_FILE    $TRANSIMS_ROOT/activity/time_intervals

LOG_ACT                            1

############################################################
# keys for ONDN activity feedback run:

ACTIVITY_FILE       /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/A-orig
ACT_FEEDBACK_FILE   /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/fdbk/init.fdbk

ACT_PARTIAL_OUTPUT  /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/partial.A-init
ACT_PROBLEM_FILE    /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/problems/problems.A-init
ACT_LOG_FILE        /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/logs/ActivityRegenerator.init.log

############################################################
```

1.2 allstr_ONDN_A-MF0.cfg

```
TRANSIMS_ROOT            /n/father/transims/CaseStudy3/scenarios/allstr

CONFIG_DEFAULT_FILE      $TRANSIMS_ROOT/allstr.cfg

############################################################
# recent corrections:

ACT_TRAVEL_TIMES_FILE              $TRANSIMS_ROOT/activity/traveltime_050101
ACT_TRAVEL_TIME_FUNCTION_MODES     1;2;3;4;5;6;7

ACT_SURVEY_ACTIVITY_FILE           $TRANSIMS_ROOT/data/survey_activities
ACT_SURVEY_WEIGHTS_FILE            $TRANSIMS_ROOT/data/survey_weights_25
#ACT_TRAVEL_TIME_INTERVALS_FILE    $TRANSIMS_ROOT/activity/time_intervals

LOG_ACT                            1

###########################################################
# keys for ONDN activity feedback run:

ACTIVITY_FILE       /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/A-init
ACT_FEEDBACK_FILE   /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/fdbk/MF0.fdbk

ACT_PARTIAL_OUTPUT  /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/A-MF0
ACT_PROBLEM_FILE    /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/problems/problems.A-MF0
ACT_LOG_FILE        /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/logs/ActivityRegenerator.MF0.log

###########################################################
```

1.3 allstr_ONDN_A-MF1.cfg

```
TRANSIMS_ROOT            /n/father/transims/CaseStudy3/scenarios/allstr

CONFIG_DEFAULT_FILE      $TRANSIMS_ROOT/allstr.cfg

############################################################
# recent corrections:

ACT_TRAVEL_TIMES_FILE             $TRANSIMS_ROOT/activity/traveltime_050101
ACT_TRAVEL_TIME_FUNCTION_MODES    1;2;3;4;5;6;7

ACT_SURVEY_ACTIVITY_FILE          $TRANSIMS_ROOT/data/survey_activities
ACT_SURVEY_WEIGHTS_FILE           $TRANSIMS_ROOT/data/survey_weights_25
#ACT_TRAVEL_TIME_INTERVALS_FILE   $TRANSIMS_ROOT/activity/time_intervals

LOG_ACT                           1

############################################################
# keys for ONDN activity feedback run:

ACTIVITY_FILE       /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/A-init
ACT_FEEDBACK_FILE   /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/fdbk/MF1.fdbk

ACT_PARTIAL_OUTPUT  /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/partial.A-MF1
ACT_PROBLEM_FILE    /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/problems/problems.A-MF1
ACT_LOG_FILE        /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/logs/ActivityRegenerator.MF1.log

############################################################
```

1.4 allstr_ONDN_A-MF2.cfg

```
TRANSIMS_ROOT            /n/father/transims/CaseStudy3/scenarios/allstr

CONFIG_DEFAULT_FILE      $TRANSIMS_ROOT/allstr.cfg

############################################################
# recent corrections:

ACT_TRAVEL_TIMES_FILE              $TRANSIMS_ROOT/activity/traveltime_050101
ACT_TRAVEL_TIME_FUNCTION_MODES     1;2;3;4;5;6;7

ACT_SURVEY_ACTIVITY_FILE           $TRANSIMS_ROOT/data/survey_activities
ACT_SURVEY_WEIGHTS_FILE            $TRANSIMS_ROOT/data/survey_weights_25
#ACT_TRAVEL_TIME_INTERVALS_FILE    $TRANSIMS_ROOT/activity/time_intervals

LOG_ACT                            1

###########################################################
# keys for ONDN activity feedback run:

ACTIVITY_FILE       /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/A-MF1
ACT_FEEDBACK_FILE   /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/fdbk/MF2.fdbk

ACT_PARTIAL_OUTPUT  /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/partial.A-MF2
ACT_PROBLEM_FILE    /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/problems/problems.A-MF2
ACT_LOG_FILE        /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/logs/ActivityRegenerator.MF2.log

###########################################################
```

1.5 allstr_ONDN_C-init.cfg

```
#######################################################################

# This file has Collator keys for creating an iteration database to be
# converted by ItdbByTour.pl into a "by-tour" database.  -JPS

#######################################################################

TRANSIMS_ROOT                  /n/father/transims/CaseStudy3/scenarios/allstr
#CONFIG_DEFAULT_FILE           $TRANSIMS_ROOT/allstr.cfg

NET_DIRECTORY                  $TRANSIMS_ROOT/network
NET_NODE_TABLE                 Node.tbl
NET_LINK_TABLE                 Link.tbl
NET_POCKET_LANE_TABLE          Pocket_Lane.tbl
NET_PARKING_TABLE              Parking.tbl
NET_LANE_CONNECTIVITY_TABLE    Lane_Connectivity.tbl
NET_UNSIGNALIZED_NODE_TABLE    Unsignalized_Node.tbl
NET_SIGNALIZED_NODE_TABLE      Signalized_Node.tbl
NET_PHASING_PLAN_TABLE         Phasing_Plan.tbl
NET_TIMING_PLAN_TABLE          Timing_Plan.tbl
NET_SPEED_TABLE                Speed.tbl
NET_LANE_USE_TABLE             Lane_Use.tbl
NET_TRANSIT_STOP_TABLE         Transit_Stop.tbl
NET_SIGNAL_COORDINATOR_TABLE   Signal_Coordinator.tbl
NET_DETECTOR_TABLE             Detector.tbl
NET_TURN_PROHIBITION_TABLE     Turn_Prohibition.tbl
NET_BARRIER_TABLE              Barrier.tbl
NET_ACTIVITY_LOCATION_TABLE    Activity_Location.tbl
NET_PROCESS_LINK_TABLE         Process_Link.tbl

ACT_HOME_ACTIVITY_TYPE         0
ACT_WORK_ACTIVITY_TYPE         1
ACT_SCHOOL_ACTIVITY_TYPE       7

ACT_ANCHOR_ACTIVITY_TYPE_1     0 # Home
ACT_ANCHOR_ACTIVITY_TYPE_2     1 # Work
ACT_ANCHOR_ACTIVITY_TYPE_3     7 # School
ACT_ANCHOR_ACTIVITY_TYPE_4     8 # College

#########################################################
# keys for parallel runs:

SEL_HOUSEHOLD_FILE    /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/HH/HH.R-init

ACT_POPULATION_FILE   $TRANSIMS_ROOT/population/pop_converted # for MakeHHFile
```

```
TRANSIT_ROUTE_FILE    $TRANSIMS_ROOT/network/Transit_Route.tbl

############################################################

SEL_POPULATION_FILE   $TRANSIMS_ROOT/population/pop_located
VEHICLE_FILE          $TRANSIMS_ROOT/vehicle/vehicles.all
SEL_PLAN_FILE         /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/plans/R-init
SEL_ACTIVITY_FILE     /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/A-init
#SEL_EVENT_FILE       $TRANSIMS_ROOT/output/anomaly.offplan.trv

SEL_ITDB_FILE         /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/itdb/init

SEL_MESSAGE_LEVEL     0
SEL_NO_ITDB_INDEX     1

############################################################
# Data gathered by the Collator for feedback

# Pop data
SEL_USE_RHHINC                  1
SEL_USE_AGE                     1

# Act data
SEL_USE_START_ACT_LOCATION      1
SEL_USE_END_ACT_TYPE            1
SEL_USE_END_MODE_PREF           1
SEL_USE_END_ACT_LOCATION        1
SEL_USE_END_OTHER_PARTICIPANTS  1
SEL_USE_END_DUR_UB              1
SEL_USE_END_DUR_LB              1
SEL_USE_DRIVER_ID               1

# Activity Location Table data
SEL_USE_START_ACT_USER_DATA     Tran_Dist; PARKING_ZN; URBAN_TYPE; River_Zone; Zones_8
SEL_USE_END_ACT_USER_DATA       Tran_Dist; PARKING_ZN; URBAN_TYPE; River_Zone; Zones_8

# Route data
SEL_USE_EUCLID                  1
SEL_USE_DURATION                1
SEL_USE_MODE_STRING             1

# event data
#SEL_USE_OFF_PLAN               1

############################################################
```

1.6 allstr_ONDN_C-MF0.cfg

```
#######################################################################

# This file has Collator keys for creating an iteration database to be
# converted by ItdbByTour.pl into a "by-tour" database.  -JPS

#######################################################################

TRANSIMS_ROOT                  /n/father/transims/CaseStudy3/scenarios/allstr
#CONFIG_DEFAULT_FILE           $TRANSIMS_ROOT/allstr.cfg

NET_DIRECTORY                  $TRANSIMS_ROOT/network
NET_NODE_TABLE                 Node.tbl
NET_LINK_TABLE                 Link.tbl
NET_POCKET_LANE_TABLE          Pocket_Lane.tbl
NET_PARKING_TABLE              Parking.tbl
NET_LANE_CONNECTIVITY_TABLE    Lane_Connectivity.tbl
NET_UNSIGNALIZED_NODE_TABLE    Unsignalized_Node.tbl
NET_SIGNALIZED_NODE_TABLE      Signalized_Node.tbl
NET_PHASING_PLAN_TABLE         Phasing_Plan.tbl
NET_TIMING_PLAN_TABLE          Timing_Plan.tbl
NET_SPEED_TABLE                Speed.tbl
NET_LANE_USE_TABLE             Lane_Use.tbl
NET_TRANSIT_STOP_TABLE         Transit_Stop.tbl
NET_SIGNAL_COORDINATOR_TABLE   Signal_Coordinator.tbl
NET_DETECTOR_TABLE             Detector.tbl
NET_TURN_PROHIBITION_TABLE     Turn_Prohibition.tbl
NET_BARRIER_TABLE              Barrier.tbl
NET_ACTIVITY_LOCATION_TABLE    Activity_Location.tbl
NET_PROCESS_LINK_TABLE         Process_Link.tbl

ACT_HOME_ACTIVITY_TYPE         0
ACT_WORK_ACTIVITY_TYPE         1
ACT_SCHOOL_ACTIVITY_TYPE       7

ACT_ANCHOR_ACTIVITY_TYPE_1     0 # Home
ACT_ANCHOR_ACTIVITY_TYPE_2     1 # Work
ACT_ANCHOR_ACTIVITY_TYPE_3     7 # School
ACT_ANCHOR_ACTIVITY_TYPE_4     8 # College

#########################################################
# keys for parallel runs:

SEL_HOUSEHOLD_FILE    /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/HH/HH.R-MF0

ACT_POPULATION_FILE   $TRANSIMS_ROOT/population/pop_converted # for MakeHHFile
```

```
TRANSIT_ROUTE_FILE    $TRANSIMS_ROOT/network/Transit_Route.tbl

############################################################

SEL_POPULATION_FILE    $TRANSIMS_ROOT/population/pop_located
VEHICLE_FILE           $TRANSIMS_ROOT/vehicle/vehicles.all
SEL_PLAN_FILE          /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/plans/R-MF0
SEL_ACTIVITY_FILE      /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/A-MF0
#SEL_EVENT_FILE        $TRANSIMS_ROOT/output/anomaly.offplan.trv

SEL_ITDB_FILE          /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/itdb/MF0

SEL_MESSAGE_LEVEL      0
SEL_NO_ITDB_INDEX      1

############################################################
# Data gathered by the Collator for feedback

# Pop data
SEL_USE_RHHINC                  1
SEL_USE_AGE                     1

# Act data
SEL_USE_START_ACT_LOCATION      1
SEL_USE_END_ACT_TYPE            1
SEL_USE_END_MODE_PREF           1
SEL_USE_END_ACT_LOCATION        1
SEL_USE_END_OTHER_PARTICIPANTS  1
SEL_USE_END_DUR_UB              1
SEL_USE_END_DUR_LB              1
SEL_USE_DRIVER_ID               1

# Activity Location Table data
SEL_USE_START_ACT_USER_DATA     Tran_Dist; PARKING_ZN; URBAN_TYPE; River_Zone; Zones_8
SEL_USE_END_ACT_USER_DATA       Tran_Dist; PARKING_ZN; URBAN_TYPE; River_Zone; Zones_8

# Route data
SEL_USE_EUCLID                  1
SEL_USE_DURATION                1
SEL_USE_MODE_STRING             1

# event data
#SEL_USE_OFF_PLAN               1

############################################################
```

1.7 allstr_ONDN_C-MF1.cfg

```
#####################################################################

# This file has Collator keys for creating an iteration database to be
# converted by ItdbByTour.pl into a "by-tour" database.  -JPS

#####################################################################

TRANSIMS_ROOT                   /n/father/transims/CaseStudy3/scenarios/allstr
#CONFIG_DEFAULT_FILE            $TRANSIMS_ROOT/allstr.cfg

NET_DIRECTORY                   $TRANSIMS_ROOT/network
NET_NODE_TABLE                  Node.tbl
NET_LINK_TABLE                  Link.tbl
NET_POCKET_LANE_TABLE           Pocket_Lane.tbl
NET_PARKING_TABLE               Parking.tbl
NET_LANE_CONNECTIVITY_TABLE     Lane_Connectivity.tbl
NET_UNSIGNALIZED_NODE_TABLE     Unsignalized_Node.tbl
NET_SIGNALIZED_NODE_TABLE       Signalized_Node.tbl
NET_PHASING_PLAN_TABLE          Phasing_Plan.tbl
NET_TIMING_PLAN_TABLE           Timing_Plan.tbl
NET_SPEED_TABLE                 Speed.tbl
NET_LANE_USE_TABLE              Lane_Use.tbl
NET_TRANSIT_STOP_TABLE          Transit_Stop.tbl
NET_SIGNAL_COORDINATOR_TABLE    Signal_Coordinator.tbl
NET_DETECTOR_TABLE              Detector.tbl
NET_TURN_PROHIBITION_TABLE      Turn_Prohibition.tbl
NET_BARRIER_TABLE               Barrier.tbl
NET_ACTIVITY_LOCATION_TABLE     Activity_Location.tbl
NET_PROCESS_LINK_TABLE          Process_Link.tbl

ACT_HOME_ACTIVITY_TYPE          0
ACT_WORK_ACTIVITY_TYPE          1
ACT_SCHOOL_ACTIVITY_TYPE        7

ACT_ANCHOR_ACTIVITY_TYPE_1      0 # Home
ACT_ANCHOR_ACTIVITY_TYPE_2      1 # Work
ACT_ANCHOR_ACTIVITY_TYPE_3      7 # School
ACT_ANCHOR_ACTIVITY_TYPE_4      8 # College

#############################################################
# keys for parallel runs:

SEL_HOUSEHOLD_FILE    /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/HH/HH.C-MF1

ACT_POPULATION_FILE   $TRANSIMS_ROOT/population/pop_converted # for MakeHHFile
```

```
TRANSIT_ROUTE_FILE    $TRANSIMS_ROOT/network/Transit_Route.tbl

############################################################

SEL_POPULATION_FILE    $TRANSIMS_ROOT/population/pop_located
VEHICLE_FILE           $TRANSIMS_ROOT/vehicle/vehicles.all
SEL_PLAN_FILE          /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/plans/R-MF1
SEL_ACTIVITY_FILE      /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/A-MF1
#SEL_EVENT_FILE        $TRANSIMS_ROOT/output/anomaly.offplan.trv

SEL_ITDB_FILE          /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/itdb/MF1

SEL_MESSAGE_LEVEL      0
SEL_NO_ITDB_INDEX      1

############################################################
# Data gathered by the Collator for feedback

# Pop data
SEL_USE_RHHINC                  1
SEL_USE_AGE                     1

# Act data
SEL_USE_START_ACT_LOCATION      1
SEL_USE_END_ACT_TYPE            1
SEL_USE_END_MODE_PREF           1
SEL_USE_END_ACT_LOCATION        1
SEL_USE_END_OTHER_PARTICIPANTS  1
SEL_USE_END_DUR_UB              1
SEL_USE_END_DUR_LB              1
SEL_USE_DRIVER_ID               1

# Activity Location Table data
SEL_USE_START_ACT_USER_DATA     Tran_Dist; PARKING_ZN; URBAN_TYPE; River_Zone; Zones_8
SEL_USE_END_ACT_USER_DATA       Tran_Dist; PARKING_ZN; URBAN_TYPE; River_Zone; Zones_8

# Route data
SEL_USE_EUCLID                  1
SEL_USE_DURATION                1
SEL_USE_MODE_STRING             1

# event data
#SEL_USE_OFF_PLAN               1

############################################################
```

1.8 allstr_ONDN_C-MF2.cfg

```
#####################################################################

# This file has Collator keys for creating an iteration database to be
# converted by ItdbByTour.pl into a "by-tour" database.  -JPS

#####################################################################

TRANSIMS_ROOT                 /n/father/transims/CaseStudy3/scenarios/allstr
#CONFIG_DEFAULT_FILE          $TRANSIMS_ROOT/allstr.cfg

NET_DIRECTORY                 $TRANSIMS_ROOT/network
NET_NODE_TABLE                Node.tbl
NET_LINK_TABLE                Link.tbl
NET_POCKET_LANE_TABLE         Pocket_Lane.tbl
NET_PARKING_TABLE             Parking.tbl
NET_LANE_CONNECTIVITY_TABLE   Lane_Connectivity.tbl
NET_UNSIGNALIZED_NODE_TABLE   Unsignalized_Node.tbl
NET_SIGNALIZED_NODE_TABLE     Signalized_Node.tbl
NET_PHASING_PLAN_TABLE        Phasing_Plan.tbl
NET_TIMING_PLAN_TABLE         Timing_Plan.tbl
NET_SPEED_TABLE               Speed.tbl
NET_LANE_USE_TABLE            Lane_Use.tbl
NET_TRANSIT_STOP_TABLE        Transit_Stop.tbl
NET_SIGNAL_COORDINATOR_TABLE  Signal_Coordinator.tbl
NET_DETECTOR_TABLE            Detector.tbl
NET_TURN_PROHIBITION_TABLE    Turn_Prohibition.tbl
NET_BARRIER_TABLE             Barrier.tbl
NET_ACTIVITY_LOCATION_TABLE   Activity_Location.tbl
NET_PROCESS_LINK_TABLE        Process_Link.tbl

ACT_HOME_ACTIVITY_TYPE        0
ACT_WORK_ACTIVITY_TYPE        1
ACT_SCHOOL_ACTIVITY_TYPE      7

ACT_ANCHOR_ACTIVITY_TYPE_1    0 # Home
ACT_ANCHOR_ACTIVITY_TYPE_2    1 # Work
ACT_ANCHOR_ACTIVITY_TYPE_3    7 # School
ACT_ANCHOR_ACTIVITY_TYPE_4    8 # College

#############################################################
# keys for parallel runs:

SEL_HOUSEHOLD_FILE    /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/HH/HH.R-MF2

ACT_POPULATION_FILE   $TRANSIMS_ROOT/population/pop_converted # for MakeHHFile
```

```
TRANSIT_ROUTE_FILE    $TRANSIMS_ROOT/network/Transit_Route.tbl

##############################################################

SEL_POPULATION_FILE   $TRANSIMS_ROOT/population/pop_located
VEHICLE_FILE          $TRANSIMS_ROOT/vehicle/vehicles.all
SEL_PLAN_FILE         /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/plans/R-MF2
SEL_ACTIVITY_FILE     /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/A-MF2
#SEL_EVENT_FILE       $TRANSIMS_ROOT/output/anomaly.offplan.trv

SEL_ITDB_FILE         /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/itdb/MF2

SEL_MESSAGE_LEVEL     0
SEL_NO_ITDB_INDEX     1

##############################################################
# Data gathered by the Collator for feedback

# Pop data
SEL_USE_RHHINC                  1
SEL_USE_AGE                     1

# Act data
SEL_USE_START_ACT_LOCATION      1
SEL_USE_END_ACT_TYPE            1
SEL_USE_END_MODE_PREF           1
SEL_USE_END_ACT_LOCATION        1
SEL_USE_END_OTHER_PARTICIPANTS  1
SEL_USE_END_DUR_UB              1
SEL_USE_END_DUR_LB              1
SEL_USE_DRIVER_ID               1

# Activity Location Table data
SEL_USE_START_ACT_USER_DATA    Tran_Dist; PARKING_ZN; URBAN_TYPE; River_Zone; Zones_8
SEL_USE_END_ACT_USER_DATA      Tran_Dist; PARKING_ZN; URBAN_TYPE; River_Zone; Zones_8

# Route data
SEL_USE_EUCLID                 1
SEL_USE_DURATION               1
SEL_USE_MODE_STRING            1

# event data
#SEL_USE_OFF_PLAN    1

##############################################################
```

1.9 allstr_ONDN-R-init.cfg

```
TRANSIMS_ROOT           /n/father/transims/CaseStudy3/scenarios/allstr

CONFIG_DEFAULT_FILE     $TRANSIMS_ROOT/allstr.cfg

##############################################################
# keys for parallel runs:

ROUTER_HOUSEHOLD_FILE   /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/HH/HH.R-init

##############################################################
# keys for ONDN re-route run:

ACTIVITY_FILE                     /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/A-init
ROUTER_LINK_DELAY_FILE            /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/summary.tim

ROUTER_PROBLEM_FILE               /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/problems/problems.R-init
PLAN_FILE                         /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/plans/R-init
ROUTER_COMPLETED_HOUSEHOLD_FILE   /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/HH/completed-HH.R-init

ROUTER_DELAY_NOISE                0.10
ROUTER_OVERDO                     0.0
ROUTER_SEED                       912345678

# This should limit route hopping without impacting route generation:
ROUTER_GET_ON_TRANSIT_DELAY 240
ROUTER_GET_OFF_TRANSIT_DELAY 240

##############################################################
```

1.10 allstr_ONDN_R-MF0.cfg

```
TRANSIMS_ROOT           /n/father/transims/CaseStudy3/scenarios/allstr

CONFIG_DEFAULT_FILE     $TRANSIMS_ROOT/allstr.cfg

##############################################################
# keys for parallel runs:

ROUTER_HOUSEHOLD_FILE   /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/HH/HH.R-MF0

##############################################################
# keys for ONDN re-route run:

ACTIVITY_FILE                     /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/A-MF0
ROUTER_LINK_DELAY_FILE            /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/summary.tim

ROUTER_PROBLEM_FILE               /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/problems/problems.R-MF0
PLAN_FILE                         /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/plans/R-MF0
ROUTER_COMPLETED_HOUSEHOLD_FILE   /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/HH/completed-HH.R-MF0

ROUTER_DELAY_NOISE                0.10
ROUTER_OVERDO                     0.0
ROUTER_SEED                       912345678

# This should limit route hopping without impacting route generation:
ROUTER_GET_ON_TRANSIT_DELAY 240
ROUTER_GET_OFF_TRANSIT_DELAY 240

##############################################################
```

1.11 allstr_ONDR_R-MF1.cfg

```
TRANSIMS_ROOT           /n/father/transims/CaseStudy3/scenarios/allstr

CONFIG_DEFAULT_FILE     $TRANSIMS_ROOT/allstr.cfg

############################################################
# keys for parallel runs:

ROUTER_HOUSEHOLD_FILE   /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/HH/HH.R-MF1

############################################################
# keys for ONDN re-route run:

ACTIVITY_FILE                       /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/A-MF1
ROUTER_LINK_DELAY_FILE              /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/summary.tim

ROUTER_PROBLEM_FILE                 /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/problems/problems.R-MF1
PLAN_FILE                           /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/plans/R-MF1
ROUTER_COMPLETED_HOUSEHOLD_FILE     /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/HH/completed-HH.R-MF1

ROUTER_DELAY_NOISE                  0.10
ROUTER_OVERDO                       0.0
ROUTER_SEED                         912345678

# This should limit route hopping without impacting route generation:
ROUTER_GET_ON_TRANSIT_DELAY 240
ROUTER_GET_OFF_TRANSIT_DELAY 240

############################################################
```

1.12 allstr_ONDN_R-MF2.cfg

```
TRANSIMS_ROOT             /n/father/transims/CaseStudy3/scenarios/allstr

CONFIG_DEFAULT_FILE      $TRANSIMS_ROOT/allstr.cfg

############################################################
# keys for parallel runs:

ROUTER_HOUSEHOLD_FILE    /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/HH/HH.R-MF2

############################################################
# keys for ONDN re-route run:

ACTIVITY_FILE                      /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/acts/A-MF2
ROUTER_LINK_DELAY_FILE             /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/summary.tim

ROUTER_PROBLEM_FILE                /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/problems/problems.R-MF2
PLAN_FILE                          /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/plans/R-MF2
ROUTER_COMPLETED_HOUSEHOLD_FILE    /n/father/transims/CaseStudy3/scenarios/allstr/feedback/mode/ONDN/HH/completed-HH.R-MF2

ROUTER_DELAY_NOISE                 0.10
ROUTER_OVERDO                      0.0
ROUTER_SEED                        912345678

# This should limit route hopping without impacting route generation:
ROUTER_GET_ON_TRANSIT_DELAY 240
ROUTER_GET_OFF_TRANSIT_DELAY 240

############################################################
```

2. SCRIPTS

2.1 CalcModeCosts.pl

```
#! /usr/local/bin/perl -w

#######################################################################

# This script takes the output of CombineDB.pl (a table of 2-mode
# tours with demographics) and calculates a table of costs (to be used
# as input to GuessMode.pl), one for each of the 3 transit modes
# (w,l,b or t).  Other arguments are the values of the cost parameters
# (alphas) to be used in the calculation.

#######################################################################

$delim = "\t";
$heads = 1;

# Assume transit cost is $1.10 on average.

$TransitCost = 1.1; # dollars

# Parking Costs:
#
# Taz          Short   Long    Location
# -----        -----   ----    --------
# 1,2,10-16    2.47    4.94    CBD South of Burnside
# 3-6          1.59    3.18    CBD North of Burnside
# 43           1.38    2.76    Oregon Health Sciences University
# 510,934-936  0.80    1.59    Oregon City
# 846-847      0.00    3.03    Lloyd District
# 971-981      0.85    1.70    Vancouver WA

@ParkingCostShort = (0, 2.47, 1.59, 1.38, 0.80, 0.00, 0.85);
@ParkingCostLong  = (0, 4.94, 3.18, 2.76, 1.59, 3.03, 1.70);

# get command line arguments:
if($#ARGV+1 != 5) {
    print "\nusage: CalcModeCosts.pl TOURITDB COSTDB ALPHA1 ALPHA2 ALPHA3\n\n";
    print "   TOURITDB = input tour-based, tab delimited iteration database\n";
    print "   COSTDB = output file with cost fields\n";
    print "   ALPHA1 = parameter in walk-cost function\n\n";
    print "   ALPHA2 = parameter in rail-cost function\n\n";
    print "   ALPHA3 = parameter in transit-cost function\n\n";
    exit;
```

```
}
local($tourdb, $costdb, $tmpalpha1, $tmpalpha2, $tmpalpha3) = @ARGV;

# use base 10 log
$alpha1 = $tmpalpha1 / log(10);
$alpha2 = $tmpalpha2 / log(10);
$alpha3 = $tmpalpha3 / log(10);

###########################

# open input itdb and parse header
open(TOURDB, "$tourdb") || die "Failed to open $tourdb.";
# read header lines
for($i=0;$i<$heads;$i++) {
    $line = <TOURDB>;
}

# need: HH, AnchorAct, Income, Type, AnchorParking, Mode, Time; Mode2, Time2

# find fields automatically using header
chomp $line;
@data = split($delim, $line);
$i=0;
$HHField = $ActField = $IncomeField = $TypeField = $ParkingField = -1;
$ModeField = $TimeField = $Mode2Field = $Time2Field = -1;
while($data[$i]) {
    if($data[$i] eq "HH") {
        $HHField = $i;
    }
    elsif($data[$i] eq "AnchorAct") {
        $ActField = $i;
    }
    elsif($data[$i] eq "Income") {
        $IncomeField = $i;
    }
    elsif($data[$i] eq "Type") {
        $TypeField = $i;
    }
    elsif($data[$i] eq "AnchorParking") {
        $ParkingField = $i;
    }
    elsif($data[$i] eq "Mode") {
        $ModeField = $i;
    }
    elsif($data[$i] eq "Time") {
        $TimeField = $i;
    }
    elsif($data[$i] eq "Mode2") {
        $Mode2Field = $i;
    }
    elsif($data[$i] eq "Time2") {
```

```
        $Time2Field = $i;
    }
    $i++;
}
# check for mandatory fields
$Missing = 0;
$NumModes = 2;
if($HHField == -1) {
    print "Missing HH.\n";
    $Missing++;
}
if($ActField == -1) {
    print "Missing AnchorAct.\n";
    $Missing++;
}
if($IncomeField == -1) {
    print "Missing Income.\n";
    $Missing++;
}
if($TypeField == -1) {
    print "Missing Type.\n";
    $Missing++;
}
if($ParkingField == -1) {
    print "Missing AnchorParking.\n";
    $Missing++;
}
if($ModeField == -1) {
    print "Missing Mode.\n";
    $Missing++;
}
if($TimeField == -1) {
    print "Missing Time.\n";
    $Missing++;
}
if($Mode2Field == -1 || $Time2Field == -1) {
    $NumModes = 1;
}
if($Missing > 0) {exit;}

###########################

# open output file
open(COSTDB, ">$costdb") || die "Failed to open $costdb.";

# print itdb header
if( $NumModes == 1 ) {
    print COSTDB "HH\tAnchorAct\tMode\tCostW\tCostL\tCostT\n";
}
else { # 2-mode DB
    print COSTDB "HH\tAnchorAct\tMode1\tCost1\tMode2\tCost2\n";
```

```
}

#####################################################

# cost = alpha * log(Income) * Time + AnchorParkingCost + ModeCost

if( $NumModes == 1 ) {
    # loop over lines
    while ( <TOURDB> ) {
        $line = $_; chomp $line;
        @data = split($delim, $line);

        # correct for bad (negative) income
        $Income2 = ($data[$IncomeField] < 0) ? 0 : $data[$IncomeField];

        # calc costs
        if($data[$ModeField] == 3 || $data[$ModeField] == 4) {
            $dollars = $TransitCost;
        }
        elsif($data[$ModeField] == 2) {
            if($data[$TypeField] eq "work") {
                $dollars = $ParkingCostLong[$data[$ParkingField]];
            }
            else { # non-work
                $dollars = $ParkingCostShort[$data[$ParkingField]];
            }
        }
        else {
            $dollars = 0;
        }

        if($data[$TimeField] eq "NA") {
            $Cost1 = $alpha1 * log(1.01+$Income2) * 86400 + $dollars;
            $Cost2 = $alpha2 * log(1.01+$Income2) * 86400 + $dollars;
            $Cost3 = $alpha3 * log(1.01+$Income2) * 86400 + $dollars;
        }
        else {
            $Cost1 = $alpha1 * log(1.01+$Income2) * $data[$TimeField] + $dollars;
            $Cost2 = $alpha2 * log(1.01+$Income2) * $data[$TimeField] + $dollars;
            $Cost3 = $alpha3 * log(1.01+$Income2) * $data[$TimeField] + $dollars;
        }

        # print cost to new db
        print COSTDB $data[$HHField],"\t",$data[$ActField],"\t",$data[$ModeField],"\t";
        print COSTDB $Cost1,"\t",$Cost2,"\t",$Cost3,"\n";
    }
}
else { # 2-mode DB
    # loop over lines
    while ( <TOURDB> ) {
        $line = $_; chomp $line;
```

```
@data = split($delim, $line);

# choose an alpha
if(($data[$ModeField] == 1 && $data[$Mode2Field] == 2)
   || ($data[$ModeField] == 2 && $data[$Mode2Field] == 1)) {
    $alpha = $alpha1;
}
elsif(($data[$ModeField] == 4 && $data[$Mode2Field] == 2)
   || ($data[$ModeField] == 2 && $data[$Mode2Field] == 4)) {
    $alpha = $alpha2;
}
elsif(($data[$ModeField] == 3 && $data[$Mode2Field] == 2)
   || ($data[$ModeField] == 2 && $data[$Mode2Field] == 3)
   || ($data[$ModeField] == 5 && $data[$Mode2Field] == 2)
   || ($data[$ModeField] == 2 && $data[$Mode2Field] == 5)) {
    $alpha = $alpha3;
}
else {
    # bad mode pair ignored
    next;
}

# correct for bad (negative) income
$Income2 = ($data[$IncomeField] < 0) ? 0 : $data[$IncomeField];

# calc costs for Mode1
if($data[$ModeField] == 3 || $data[$ModeField] == 4) {
    $dollars = $TransitCost;
}
elsif($data[$ModeField] == 2) {
    if($data[$TypeField] eq "work") {
        $dollars = $ParkingCostLong[$data[$ParkingField]];
    }
    else { # non-work
        $dollars = $ParkingCostShort[$data[$ParkingField]];
    }
}
else {
    $dollars = 0;
}

if($data[$TimeField] eq "NA") {
    $Cost1 = $alpha * log(1.01+$Income2) * 86400 + $dollars;
}
else {
    $Cost1 = $alpha * log(1.01+$Income2) * $data[$TimeField] + $dollars;
}

# calc costs for Mode2
if($data[$Mode2Field] == 3 || $data[$Mode2Field] == 4) {
    $dollars = $TransitCost;
```

```
        }
        elsif($data[$Mode2Field] == 2) {
            if($data[$TypeField] eq "work") {
                $dollars = $ParkingCostLong[$data[$ParkingField]];
            }
            else { # non-work
                $dollars = $ParkingCostShort[$data[$ParkingField]];
            }
        }
        else {
            $dollars = 0;
        }

        if($data[$Time2Field] eq "NA") {
            $Cost2 = $alpha * log(1.01+$Income2) * 86400 + $dollars;
        }
        else {
            $Cost2 = $alpha * log(1.01+$Income2) * $data[$Time2Field] + $dollars;
        }

        # print cost to new db
        print COSTDB $data[$HHField],"\t",$data[$ActField],"\t",$data[$ModeField],"\t",$Cost1;
        print COSTDB "\t",$data[$Mode2Field],"\t",$Cost2,"\n";
    }
}

#####################################################

close TOURDB;
close COSTDB;

###########################

exit;
```

2.2 CombineDB.pl

```
#! /usr/local/bin/perl -w

#######################################################################

# This script takes a database of tested tours and second tour-based
# iteration database with all tours in the re-routed HH, and outputs a
# single 2-mode database to be used in calibrating alpha (the mode
# choice parameter).

# It is keyed on Trav & Tour, and saves only Mode and Time from second
# file.

# Entries in file2 but not file1 are ignored.

#######################################################################

$delim = "\t";
$heads = 1;

# get command line arguments: original and new iteration database filenames
if($#ARGV+1 != 3) {
    print "\nusage: CombineDB.pl TDB1 TDB2 COMBINEDDB\n\n";
    print "   TDB1 = main input tour-database of tested tours\n";
    print "   TDB2 = second input tour-database of rerouted HH\n";
    print "   COMBINEDDB = output, combined-mode tour database\n";
    exit;
}
local($tdb1, $tdb2, $combinedb) = @ARGV;

# open input files
open(TDB1, "$tdb1") || die "Failed to open $tdb1.";
open(TDB2, "$tdb2") || die "Failed to open $tdb2.";

# First get file 2 headers:

# read header lines
for($i=0;$i<$heads;$i++) {
    $line = <TDB2>;
}
# find fields automatically using header
chomp $line;
@data = split($delim, $line);
$i=0;
$TravField1 = $TourField1 = $ModeField1 = $TimeField1 = -1;
while($data[$i]) {
    if($data[$i] eq "Trav") {
```

```
        $TravField1 = $i;
    }
    elsif($data[$i] eq "Tour") {
        $TourField1 = $i;
    }
    elsif($data[$i] eq "Mode") {
        $ModeField1 = $i;
    }
    elsif($data[$i] eq "Time") {
        $TimeField1 = $i;
    }
    $i++;
}
# check for mandatory fields
$Missing = 0;
if($TravField1 == -1) {
    print "Missing Trav in $tdb2.\n";
    $Missing++;
}
if($TourField1 == -1) {
    print "Missing Tour in $tdb2.\n";
    $Missing++;
}
if($ModeField1 == -1) {
    print "Missing Mode in $tdb2.\n";
    $Missing++;
}
if($TimeField1 == -1) {
    print "Missing Time in $tdb2.\n";
    $Missing++;
}

# Then get file 1 header (so we can copy it to output):

# read header lines
for($i=0;$i<$heads;$i++) {
    $line = <TDB1>;
}
# find fields automatically using header
chomp $line;
@data = split($delim, $line);
$i=0;
$TravField2 = $TourField2 = $ModeField2 = $TimeField2 = -1;
while($data[$i]) {
    if($data[$i] eq "Trav") {
        $TravField2 = $i;
    }
    elsif($data[$i] eq "Tour") {
        $TourField2 = $i;
    }
    elsif($data[$i] eq "Mode") {
```

```
        $ModeField2 = $i;
    }
    elsif($data[$i] eq "Time") {
        $TimeField2 = $i;
    }
    $i++;
}
# check for mandatory fields
if($TravField2 == -1) {
    print "Missing Trav in $tdb1.\n";
    $Missing++;
}
if($TourField2 == -1) {
    print "Missing Tour in $tdb1.\n";
    $Missing++;
}
if($ModeField2 == -1) {
    print "Missing Mode in $tdb1.\n";
    $Missing++;
}
if($TimeField2 == -1) {
    print "Missing Time in $tdb1.\n";
    $Missing++;
}
if($Missing > 0) {exit;}

# open output files, print new header
open(COMBTDB, ">$combinedb") || die "Failed to open $combinedb.";

print COMBTDB $line,"\tMode2\tTime2\n";

#######################################################

# record data from database-2
# this has all tours in the tested HHs
while ( <TDB2> ) {
    $line = $_; chomp $line;
    @data = split($delim, $line);

    # save in a hash - should be unique
    $db2{$data[$TravField2]}{$data[$TourField2]}
      = join("\t",$data[$ModeField2],$data[$TimeField2]);
}

close TDB2;

###########################

# this is for the original DB, with only tested tours
while ( <TDB1> ) {
    $line = $_; chomp $line;
```

```
    @data = split($delim, $line);

    $Trav = $data[$TravField1];
    $Tour = $data[$TourField1];

    if( $db2{$Trav}{$Tour} ) {
        # save in file if match exists
        print COMBTDB $line,"\t",$db2{$Trav}{$Tour},"\n";
    }
    else {
        print "ERROR: no match in 2nd file for Trav=",$Trav,", Tour=",$Tour,"\n";
    }
}

close TDB1;
close COMBTDB;

#####################################################

exit;
```

2.3 FilterONDN.pl

```
#! /usr/local/bin/perl -w

#######################################################################

# This takes a tour-based itdb as input and filters to include only
# non-shared, ONDN tours with traveler's age>15.

#######################################################################

$delim = "\t";
$heads = 1;

$TranDist = 900; # maximum transit distance

# get command line arguments:
if($#ARGV+1 != 2) {
    print "\nusage: FilterONDN.pl TOURITDB NEWDB\n\n";
    print "   TOURITDB = input tour-based, tab delimited iteration database\n";
    print "   NEWDB = output as TOURITDB, but only selected tours\n\n";
    exit;
}
local($origitdb, $outitdb) = @ARGV;

# open input itdb and parse header
open(ORIGITDB, "$origitdb") || die "Failed to open $origitdb.";
# read header lines
for($i=0;$i<$heads;$i++) {
    $line = <ORIGITDB>;
}

# find fields automatically using header
chomp $line;
@data = split($delim, $line);
$i=0;
$AgeField = $ModeField = $SharField = $Tran1Field = $Tran2Field = -1;
while($data[$i]) {
    if($data[$i] eq "Age") {
       $AgeField = $i;
    }
    elsif($data[$i] eq "Mode") {
       $ModeField = $i;
    }
    elsif($data[$i] eq "Shared") {
       $SharField = $i;
    }
    elsif($data[$i] eq "HomeTranDist") {
```

```
        $Tran1Field = $i;
    }
    elsif($data[$i] eq "AnchorTranDist") {
        $Tran2Field = $i;
    }
    $i++;
}
# check for mandatory fields
$Missing = 0;
if($AgeField == -1) {
    print "Missing Age.\n";
    $Missing++;
}
if($ModeField == -1) {
    print "Missing Mode.\n";
    $Missing++;
}
if($SharField == -1) {
    print "Missing Shared.\n";
    $Missing++;
}
if($Tran1Field == -1) {
    print "Missing HomeTranDist.\n";
    $Missing++;
}
if($Tran2Field == -1) {
    print "Missing AnchorTranDist.\n";
    $Missing++;
}
if($Missing > 0) {exit;}


# open output files
open(NEWITDB, ">$outitdb") || die "Failed to open $outitdb.";

# print itdb header
print NEWITDB $line,"\n";

###########################

# loop over lines
while ( <ORIGITDB> ) {
    $line = $_; chomp $line;
    @data = split($delim, $line);

    # not-shared; ONDN; age>15;
    if( $data[$SharField] == 0
       && $data[$Tran1Field]<$TranDist && $data[$Tran2Field]<$TranDist
       && $data[$AgeField] > 15 ) {

        # print to new itdb
```

```
        print NEWITDB $line,"\n";
    }
}

##########################

close ORIGITDB;
close NEWITDB;

##########################
exit;
```

2.4 fit.csh

```
#! /bin/csh

#######################################################################

# This script simply runs the TestAlpha script for ten different
# values of alpha to count what fraction of travelers use the given
# mode over walk.  It is used in determining the alpha that will
# produce the correct mode split.

# Arguments are the substring that determines the combined tour
# database to test, the mode to compare to auto, and the top an bottom
# of the desired test range for alpha.

#######################################################################

set STRAT = $1
set MODE = $2
set BOT = $3
set TOP = $4

set TEST = `echo "scale=4;$TOP/$BOT" | bc -l`
if($TEST == "10.0000") then
    set INC = `echo "($TOP-$BOT)/9" | bc -l`
    set NUM =  10
else
    set INC = `echo "($TOP-$BOT)/10" | bc -l`
    set NUM =  11
endif

set a = $BOT
@ I = 1

while( $I <= $NUM )
    #echo $I $a
    ./scripts/TestAlpha.pl itdb/combined.$STRAT.tdb $MODE 2 $a
    set a = `echo "$a+$INC" | bc -l`
    @ I++
end
```

2.5 FixModes.pl

```
#! /usr/local/bin/perl -w

#######################################################################

# This will remode a specified fraction of tours selected uniformaly
# at random.  Each of walk, rail-only, general transit, and auto tours
# have their own remoding fraction.  Other tour modes are ignored.

# This only makes mode conversions in the acivity file between modes 2
# or 3.

#######################################################################

$delim = "\t";
$heads = 1;

# get command line arguments:
if($#ARGV+1 != 7) {
    print "\nusage: ReMode.pl SEED FRAC1 FRAC2 FRAC3 FRAC4 TOURDB ACTIVFB\n\n";
    print "   SEED = seed to the random number generator that selects traveler-tours\n";
    print "   FRAC1 = fraction of walk tours needed from total by re-mode\n";
    print "   FRAC2 = fraction of rail tours needed from total by re-mode\n";
    print "   FRAC3 = fraction of transit tours needed from total by re-mode\n";
    print "   FRAC4 = fraction of auto tours needed from total by re-mode\n";
    print "   TOURDB = input tour-based, tab delimited iteration database\n";
    print "   ACTIVFB = main activity regenerator feedback command file\n";
    exit;
}
local($seed, $frac1, $frac2, $frac3, $frac4, $origitdb, $activfile) = @ARGV;

# set random number seed
srand($seed);

# open input itdb and parse header
open(ORIGITDB, "$origitdb") || die "Failed to open $origitdb.";
# read header lines
for($i=0;$i<$heads;$i++) {
    $line = <ORIGITDB>;
}

# find fields automatically using header
chomp $line;
@data = split($delim, $line);
$i=0;
$HHField = $ModeField = $ModePrefField = $ActField = -1;
while($data[$i]) {
```

```
    if($data[$i] eq "HH") {
        $HHField = $i;
    }
    elsif($data[$i] eq "ModePref") {
        $ModePrefField = $i;
    }
    elsif($data[$i] eq "Mode") {
        $ModeField = $i;
    }
    elsif($data[$i] eq "AnchorAct") {
        $ActField = $i;
    }
    $i++;
}
# check for mandatory fields
$Missing = 0;
if($HHField == -1) {
    print "Missing HH.\n";
    $Missing++;
}
if($ModePrefField == -1) {
    print "Missing ModePref.\n";
    $Missing++;
}
if($ModeField == -1) {
    print "Missing Mode.\n";
    $Missing++;
}
if($ActField == -1) {
    print "Missing AnchorAct.\n";
    $Missing++;
}
if($Missing > 0) {exit;}


# open output files
open(ACTIVFB, ">$activfile") || die "Failed to open $activfile.";


###########################

# loop over lines
while ( <ORIGITDB> ) {
    $line = $_; chomp $line;
    @data = split($delim, $line);

    # only selected trav :
    if( $data[$ModePrefField]==2 && ($data[$ModeField]==2 || $data[$ModeField]==1) && rand() <= $frac4 ) {
        # send mode-change feedback command to activity feedback file
        print ACTIVFB $data[$HHField]," ",$data[$ActField]," M 3 2\n";
    }
```

```
    elsif( $data[$ModePrefField]==3 && ( ($data[$ModeField]==1 && rand() <= $frac1)
           || ($data[$ModeField]==4 && rand() <= $frac2)
           || (($data[$ModeField]==3 || $data[$ModeField]==5) && rand() <= $frac3) ) ) {
        print ACTIVFB $data[$HHField]," ",$data[$ActField]," M 2 3\n";
    }
}

##########################

close ORIGITDB;
close ACTIVFB;

##########################

exit;
```

2.6 GuessMode.C

```
// This crude program will read in a 2-mode cost file and construct a
// 2D distibution of cost1 v. cost2.  The second input file will have
// only one mode-cost pair.  From the distribution, a second mode-cost
// pair will be guessed and a mode choice made.  Output will be a
// household file and an activity geenrator feedback command file for
// any necessary new mode choices.

// Created 20010720 - jps
// Major modifications 20010831 - jps

#define MAX_CHARS 10000

#define MODE1  1  // walk
#define MODE2  2  // auto
#define MODE3  3  // bus or "transit"
#define MODE4  4  // rail-only
#define MODE5  5  // mixed transit

#include <iostream>
#include <fstream>
#include <string>
#include <stdlib.h>
#include <map>

int  main(int argc, char **argv) {

  ifstream CalibCostFile, InputCostFile;
  ofstream HHFile, FdbkFile, LogFile;
  string   CalibCostFileName, InputCostFileName, HHFileName, FdbkFileName, LogFileName;
  long     MinCount, SeedVal;
  float    SampleFrac;
  char     TmpStr[MAX_CHARS+1];

  multimap<float,float>  Cost12;  // Cost of mode 2 given cost of mode 1
  multimap<float,float>  Cost21;  // Cost of mode 1 given cost of mode 2
  multimap<float,float>  Cost42;  // Cost of mode 2 given cost of mode 4
  multimap<float,float>  Cost24;  // Cost of mode 4 given cost of mode 2
  multimap<float,float>  Cost32;  // Cost of mode 2 given cost of mode 3
  multimap<float,float>  Cost23;  // Cost of mode 3 given cost of mode 2


  //=========================================================================
  // get command line arguments

  if(argc != 8 && argc != 9 ) {
    cout << "usage: GuessMode FRAC MIN SEED 2COSTDB COSTDB HHFILE FDBKFILE [LOG]"
```

```
      << endl;
  exit(1);
}
else {
  SampleFrac        = atof(argv[1]);  // fraction of total records to use in sample
  MinCount          = atoi(argv[2]);  // minimum number of records to use in each sample
  SeedVal           = atoi(argv[3]);  // for mode selection
  CalibCostFileName = argv[4];  // database of 2-mode costs
  InputCostFileName = argv[5];  // database of 1-mode, 3-costs
  HHFileName        = argv[6];  // output router feedback file listing all HH in FDBKFILE
  FdbkFileName      = argv[7];  // output mode feedback command file
}
if(argc == 9) {
  LogFileName       = argv[8]; // output of prob. of mode change by entry in 1COSTDB
}

//==========================================================================
// open all files, set random number seed

CalibCostFile.open(CalibCostFileName.c_str(), ios::in);
if(!CalibCostFile) {
  cerr << "2-Mode cost file could not be opened." << endl;
  exit(1);
}
InputCostFile.open(InputCostFileName.c_str(), ios::in);
if(!InputCostFile) {
  cerr << "1-Mode, 3-cost file could not be opened." << endl;
  exit(1);
}
HHFile.open(HHFileName.c_str(), ios::out);
if(!HHFile) {
  cerr << "HH file could not be opened." << endl;
  exit(1);
}
FdbkFile.open(FdbkFileName.c_str(), ios::out);
if(!FdbkFile) {
  cerr << "Fdbk file could not be opened." << endl;
  exit(1);
}
if(argc == 9) {
  LogFile.open(LogFileName.c_str(), ios::out);
  if(!LogFile) {
    cerr << "Log file could not be opened." << endl;
    exit(1);
  }
}

srand48(SeedVal);

//==========================================================================
// read in 2-mode file (HH, ActID, Mode1, Cost1, Mode2, Cost2)
```

```
int   HH, Act;
int   TmpMode1, TmpMode2;
float TmpCost1, TmpCost2, TmpCost3;

// Discard the header line
CalibCostFile.getline(TmpStr,MAX_CHARS);
// Read lines if any.
CalibCostFile >> HH >> Act >> TmpMode1 >> TmpCost1 >> TmpMode2 >> TmpCost2;
CalibCostFile.getline(TmpStr,MAX_CHARS);
// NOTE: This method allows for blank line at end of file, but last
// line will be ignored if it doesn't have a line-feed
while(!CalibCostFile.eof()) {
  // only accept auto in one mode and w,l,b,t,m in the other:
  if(TmpMode1 == MODE2) {
    if(TmpMode2 == MODE1) {
      Cost12.insert(pair<float,float>(TmpCost2,TmpCost1)); // Cost12[TmpCost2] = TmpCost1;
      Cost21.insert(pair<float,float>(TmpCost1,TmpCost2)); // Cost21[TmpCost1] = TmpCost2;
    }
    else if(TmpMode2 == MODE4) {
      Cost42.insert(pair<float,float>(TmpCost2,TmpCost1)); // Cost42[TmpCost2] = TmpCost1;
      Cost24.insert(pair<float,float>(TmpCost1,TmpCost2)); // Cost24[TmpCost1] = TmpCost2;
    }
    else if(TmpMode2 == MODE3 || TmpMode2 == MODE5) {
      Cost32.insert(pair<float,float>(TmpCost2,TmpCost1)); // Cost32[TmpCost2] = TmpCost1;
      Cost23.insert(pair<float,float>(TmpCost1,TmpCost2)); // Cost23[TmpCost1] = TmpCost2;
    }
  }
  else if(TmpMode2 == MODE2) {
    if(TmpMode1 == MODE1) {
      Cost12.insert(pair<float,float>(TmpCost1,TmpCost2)); // Cost12[TmpCost1] = TmpCost2;
      Cost21.insert(pair<float,float>(TmpCost2,TmpCost1)); // Cost12[TmpCost2] = TmpCost1;
    }
    else if(TmpMode1 == MODE4) {
      Cost42.insert(pair<float,float>(TmpCost1,TmpCost2)); // Cost42[TmpCost1] = TmpCost2;
      Cost24.insert(pair<float,float>(TmpCost2,TmpCost1)); // Cost24[TmpCost2] = TmpCost1;
    }
    else if(TmpMode1 == MODE3 || TmpMode1 == MODE5) {
      Cost32.insert(pair<float,float>(TmpCost1,TmpCost2)); // Cost32[TmpCost1] = TmpCost2;
      Cost23.insert(pair<float,float>(TmpCost2,TmpCost1)); // Cost23[TmpCost2] = TmpCost1;
    }
  }
  // read next line, if any
  CalibCostFile >> HH >> Act >> TmpMode1 >> TmpCost1 >> TmpMode2 >> TmpCost2;
  CalibCostFile.getline(TmpStr,MAX_CHARS);
}

// won't be needing this anymore
CalibCostFile.close();

// simple check
```

```
  if(Cost21.size() < MinCount) {
    cerr<<"MIN = "<<MinCount<<": Too few entries ("<<Cost21.size()<<") with mode1."<<endl;
    exit(2);
  }
  else if(Cost23.size() < MinCount) {
    cerr<<"MIN = "<<MinCount<<": Too few entries ("<<Cost23.size()<<") with mode3."<<endl;
    exit(2);
  }
  else if(Cost24.size() < MinCount) {
    cerr<<"MIN = "<<MinCount<<": Too few entries ("<<Cost24.size()<<") with mode4."<<endl;
    exit(2);
  }

  //==========================================================================
  // foreach line of 1-mode file (HH, ActID, Mode1, Cost1, Cost2, Cost3),
  // output feedback commands if new mode is needed

  multimap<float,float>::iterator  CostIterUp, CostIterDown;
  multimap<float,float>::iterator  CostIterEnd;
  multimap<float,float>::iterator  Cost1IterUp, Cost1IterDown;
  multimap<float,float>::iterator  Cost1IterEnd;
  multimap<float,float>::iterator  Cost4IterUp, Cost4IterDown;
  multimap<float,float>::iterator  Cost4IterEnd;
  multimap<float,float>::iterator  Cost3IterUp, Cost3IterDown;
  multimap<float,float>::iterator  Cost3IterEnd;
  bool  Stop;
  long  Sum;  // Number in sample that have lower cost in new mode
  long  Count; // Total number in sample
  float CostDiffMin;
  float CostDiffUp, CostDiffDown;
  float Cost1DiffUp, Cost1DiffDown;
  float Cost4DiffUp, Cost4DiffDown;
  float Cost3DiffUp, Cost3DiffDown;
  long  MaxCount;
  long  LastHH = -999;
  int   NewMode, OldMode;

  // Discard the header line
  InputCostFile.getline(TmpStr,MAX_CHARS);
  // Read lines if any.
  InputCostFile >> HH >> Act >> TmpMode1 >> TmpCost1 >> TmpCost2 >> TmpCost3;
  InputCostFile.getline(TmpStr,MAX_CHARS);

  while(!InputCostFile.eof()) {

    Stop = false;
    Sum  = 0;

    // Choose database(s) of costs ==================================

    if(TmpMode1 == MODE1) { // guess on mode12 db only
```

```
    CostIterUp = Cost12.upper_bound(TmpCost1); // first element whose key is greater than arg
    CostIterDown = CostIterUp;  CostIterDown--; // one less
    CostIterEnd = Cost12.end();
    NewMode = MODE2;
    OldMode = MODE3;
    MaxCount = (long) ( (float) SampleFrac * Cost12.size() );
  }
  else if(TmpMode1 == MODE4) { // guess on mode4 db only
    CostIterUp = Cost42.upper_bound(TmpCost1); // first element whose key is greater than arg
    CostIterDown = CostIterUp;  CostIterDown--; // one less
    CostIterEnd = Cost42.end();
    NewMode = MODE2;
    OldMode = MODE3;
    MaxCount = (long) ( (float) SampleFrac * Cost42.size() );
  }
  else if(TmpMode1 == MODE3 || TmpMode1 == MODE5) { // guess on mode3 db only
    CostIterUp = Cost32.upper_bound(TmpCost1); // first element whose key is greater than arg
    CostIterDown = CostIterUp;  CostIterDown--; // one less
    CostIterEnd = Cost32.end();
    NewMode = MODE2;
    OldMode = MODE3;
    MaxCount = (long) ( (float) SampleFrac * Cost32.size() );
  }
  else if(TmpMode1 == MODE2) { // guess on all mode databases
    // for mode1 database
    Cost1IterUp = Cost21.upper_bound(TmpCost1);
    Cost1IterDown = Cost1IterUp;  Cost1IterDown--;
    Cost1IterEnd = Cost21.end();
    // for mode4 database
    Cost4IterUp = Cost24.upper_bound(TmpCost1);
    Cost4IterDown = Cost4IterUp;  Cost4IterDown--;
    Cost4IterEnd = Cost24.end();
    // for mode3 database
    Cost3IterUp = Cost23.upper_bound(TmpCost1);
    Cost3IterDown = Cost3IterUp;  Cost3IterDown--;
    Cost3IterEnd = Cost23.end();

    NewMode = MODE3;
    OldMode = MODE2;
    MaxCount = (long) ( (float) SampleFrac * (Cost12.size() + Cost42.size() + Cost32.size()) );
  }
  else { // skip invalid modes: read next line before continue
    InputCostFile >> HH >> Act >> TmpMode1 >> TmpCost1;
    InputCostFile.getline(TmpStr,MAX_CHARS);
    continue;
  }

  if(MaxCount < MinCount) MaxCount = MinCount;

  // Different behaviour for mode2 than for others in sampling data
```

```
if(TmpMode1 == MODE2) {

  // collect sample ==============================================

  // set iterators
  Cost1DiffUp   = Cost1IterUp->first - TmpCost1;
  Cost1DiffDown = TmpCost1 - Cost1IterDown->first;
  Cost4DiffUp   = Cost4IterUp->first - TmpCost1;
  Cost4DiffDown = TmpCost1 - Cost4IterDown->first;
  Cost3DiffUp   = Cost3IterUp->first - TmpCost1;
  Cost3DiffDown = TmpCost1 - Cost3IterDown->first;

  // invalidate any direction already at boundary
  if(Cost1IterUp   == Cost1IterEnd) Cost1DiffUp = -1;
  if(Cost1IterDown == Cost1IterEnd) Cost1DiffDown = -1;
  if(Cost4IterUp   == Cost4IterEnd) Cost4DiffUp = -1;
  if(Cost4IterDown == Cost4IterEnd) Cost4DiffDown = -1;
  if(Cost3IterUp   == Cost3IterEnd) Cost3DiffUp = -1;
  if(Cost3IterDown == Cost3IterEnd) Cost3DiffDown = -1;

  // accumulate in direction whose NEXT key is closest to query
  for(Count=0; Count < MaxCount; Count++) {

    // find closest data point (might be more efficient with a sorted list)
    if(Cost1DiffDown != -1)  CostDiffMin = Cost1DiffDown;
    else if(Cost1DiffUp != -1   && Cost1DiffUp < CostDiffMin)   CostDiffMin = Cost1DiffUp;
    else if(Cost4DiffDown != -1 && Cost4DiffDown < CostDiffMin) CostDiffMin = Cost4DiffDown;
    else if(Cost4DiffUp != -1   && Cost4DiffUp < CostDiffMin)   CostDiffMin = Cost4DiffUp;
    else if(Cost3DiffDown != -1 && Cost3DiffDown < CostDiffMin) CostDiffMin = Cost3DiffDown;
    else if(Cost3DiffUp != -1   && Cost3DiffUp < CostDiffMin)   CostDiffMin = Cost3DiffUp;

    // take data from there
    if(Cost1DiffDown == CostDiffMin) {
      if(Cost1IterDown->first > Cost1IterDown->second) Sum++;
      Cost1IterDown--;
      Cost1DiffDown = TmpCost1 - Cost1IterDown->first;
      if(Cost1IterDown == Cost1IterEnd)  Stop = true;
    }
    else if(Cost1DiffUp == CostDiffMin) {
      if(Cost1IterUp->first > Cost1IterUp->second) Sum++;
      Cost1IterUp++;
      Cost1DiffUp = Cost1IterUp->first - TmpCost1;
      if(Cost1IterUp == Cost1IterEnd)  Stop = true;
    }
    else if(Cost4DiffDown == CostDiffMin) {
      if(Cost4IterDown->first > Cost4IterDown->second) Sum++;
      Cost4IterDown--;
      Cost4DiffDown = TmpCost1 - Cost4IterDown->first;
      if(Cost4IterDown == Cost4IterEnd)  Stop = true;
    }
    else if(Cost4DiffUp == CostDiffMin) {
```

```
        if(Cost4IterUp->first > Cost4IterUp->second) Sum++;
        Cost4IterUp++;
        Cost4DiffUp = Cost4IterUp->first - TmpCost1;
        if(Cost4IterUp == Cost4IterEnd)  Stop = true;
      }
      else if(Cost3DiffDown == CostDiffMin) {
        if(Cost3IterDown->first > Cost3IterDown->second) Sum++;
        Cost3IterDown--;
        Cost3DiffDown = TmpCost1 - Cost3IterDown->first;
        if(Cost3IterDown == Cost3IterEnd)  Stop = true;
      }
      else if(Cost3DiffUp == CostDiffMin) {
        if(Cost3IterUp->first > Cost3IterUp->second) Sum++;
        Cost3IterUp++;
        Cost3DiffUp = Cost3IterUp->first - TmpCost1;
        if(Cost3IterUp == Cost3IterEnd)  Stop = true;
      }
    }

  }
  else { // not mode 2: use simple probability calculation

    // collect sample ==============================================

    // move in direction whose NEXT key is closest to query (not PRESENT key)
    CostDiffUp   = CostIterUp->first - TmpCost1;
    CostDiffDown = TmpCost1 - CostIterDown->first;

    if(CostIterUp == CostIterEnd)  Stop = true;
    if(CostIterDown == CostIterEnd)  Stop = true;

    for(Count=0; Count < MaxCount && Stop == false; Count++) {
      if(CostDiffUp > CostDiffDown) {
        if(CostIterDown->first > CostIterDown->second) Sum++;
        CostIterDown--;
        CostDiffDown = TmpCost1 - CostIterDown->first;
        if(CostIterDown == CostIterEnd)  Stop = true;
      }
      else {
        if(CostIterUp->first > CostIterUp->second) Sum++;
        CostIterUp++;
        CostDiffUp = CostIterUp->first - TmpCost1;
        if(CostIterUp == CostIterEnd)  Stop = true;
      }
    }

    // if one side exceeded bounds, accumulate from other until min-count is reached
    if(Stop == true && Count < MinCount) {
      if(CostIterUp == CostIterEnd) { // count down
        while(Count < MinCount) {
          if(CostIterDown->first > CostIterDown->second) Sum++;
```

```
            Count++;
            CostIterDown--;
          }
        }
        else { // count up
          while(Count < MinCount) {
            if(CostIterUp->first > CostIterUp->second) Sum++;
            Count++;
            CostIterUp++;
          }
        }
      }

    } // end: not mode2

    // log
    if(argc == 9) LogFile << HH <<" "<< Act <<" "<< ((float) Sum/Count) <<endl;

    // make mode choice decision, output if necessary ==============

    if(drand48() < ((float) Sum/Count)) {
      // send new mode to feedback command file
      FdbkFile<< HH <<" "<< Act <<" M "<< NewMode <<" "<< OldMode <<endl;
      // only save HH if haven't already
      if(LastHH != HH) {
        HHFile<< HH <<endl;;
        LastHH = HH;
      }
    }

    //=============================================

    // next line if there is one
    InputCostFile >> HH >> Act >> TmpMode1 >> TmpCost1 >> TmpCost2 >> TmpCost3;
    InputCostFile.getline(TmpStr,MAX_CHARS);

  } // end: read all lines of file2

  //=========================================================================
  // clean-up

  InputCostFile.close();
  HHFile.close();
  FdbkFile.close();
  if(argc == 9) LogFile.close();

} // end: main
```

2.7 ItdbByTour.pl

```
#! /usr/local/bin/perl -w

#######################################################################

# This script takes an iteration database with the mandatory fields as
# specified below, and creates a new tour-based iteration database.

# The tour database will have four types of fields: traveler/household
# demographics, properties of the tour, properties of the home
# location, and properties of the primary anchor.

#######################################################################

# To get fields from itdb using gawk:
# head -2 itdb.000.it | tail -1 | gawk '{for(i=1;i<=NF;i++)print i-1,$i}' FS=","

$delim = ",";
$heads = 2;

# get command line arguments: original and new iteration database filenames
if($#ARGV+1 != 2) {
    print "\nusage: ItdbByTour.pl ORIGITDB TOURITDB\n\n";
    exit;
}
local($origitdb, $touritdb) = @ARGV;

# open input file
open(ORIGITDB, "$origitdb") || die "Failed to open $origitdb.";
# read header lines
for($i=0;$i<$heads;$i++) {
    $line = <ORIGITDB>;
}
# find fields automatically using header
chomp $line;
@data = split($delim, $line);
$i=0;
$HHField = $TravField = $TourField = $SubTField = $Act1Field = $Act2Field = $IncomeField = $AgeField = $ActLoc1Field = $ActLoc2Field =
$TypeField = $ModeField = $SharField = $DurLBField = $DurUBField = $DriverField = $ParkingField = $Urban1Field = $Urban2Field =
$River1Field = $River2Field = $Tran1Field = $Tran2Field = $Zone1Field = $Zone2Field = $TimeField = $DistField = $ModeStringField = -1;
while($data[$i]) {
    if($data[$i] eq "HH") {
        $HHField = $i;
    }
    elsif($data[$i] eq "TRAV") {
        $TravField = $i;
    }
    elsif($data[$i] eq "TOUR") {
        $TourField = $i;
```

```
    }
    elsif($data[$i] eq "SUBTOUR") {
        $SubTField = $i;
    }
    elsif($data[$i] eq "START_ACT_ID") {
        $Act1Field = $i;
    }
    elsif($data[$i] eq "END_ACT_ID") {
        $Act2Field = $i;
    }
    elsif($data[$i] eq "RHHINC") {
        $IncomeField = $i;
    }
    elsif($data[$i] eq "AGE") {
        $AgeField = $i;
    }
    elsif($data[$i] eq "START_ACT_LOCATION") {
        $ActLoc1Field = $i;
    }
    elsif($data[$i] eq "END_ACT_LOCATION") {
        $ActLoc2Field = $i;
    }
    elsif($data[$i] eq "END_ACT_TYPE") {
        $TypeField = $i;
    }
    elsif($data[$i] eq "END_MODE_PREF") {
        $ModeField = $i;
    }
    elsif($data[$i] eq "END_OTHER_PARTICIPANTS") {
        $SharField = $i;
    }
    elsif($data[$i] eq "END_DUR_UB") {
        $DurUBField = $i;
    }
    elsif($data[$i] eq "END_DUR_LB") {
        $DurLBField = $i;
    }
    elsif($data[$i] eq "DRIVER_ID") {
        $DriverField = $i;
    }
    elsif($data[$i] eq "END_ACT_USER_DATA_PARKING_ZN") { # *
        $ParkingField = $i;
    }
    elsif($data[$i] eq "START_ACT_USER_DATA_URBAN_TYPE") { # *
        $Urban1Field = $i;
    }
    elsif($data[$i] eq "END_ACT_USER_DATA_URBAN_TYPE") { # *
        $Urban2Field = $i;
    }
    elsif($data[$i] eq "START_ACT_USER_DATA_River_Zone") { # *
        $River1Field = $i;
```

```
    }
    elsif($data[$i] eq "END_ACT_USER_DATA_River_Zone") { # *
        $River2Field = $i;
    }
    elsif($data[$i] eq "START_ACT_USER_DATA_Tran_Dist") {
        $Tran1Field = $i;
    }
    elsif($data[$i] eq "END_ACT_USER_DATA_Tran_Dist") {
        $Tran2Field = $i;
    }
    elsif($data[$i] eq "START_ACT_USER_DATA_Zones_8") {
        $Zone1Field = $i;
    }
    elsif($data[$i] eq "END_ACT_USER_DATA_Zones_8") {
        $Zone2Field = $i;
    }
    elsif($data[$i] eq "DURATION") {
    #elsif($data[$i] eq "TIME_SUM") {
        $TimeField = $i;
    }
    elsif($data[$i] eq "EUCLID") {
    #elsif($data[$i] eq "DISTANCE_SUM") {
        $DistField = $i;
    }
    elsif($data[$i] eq "MODE_STRING") {
        $ModeStringField = $i;
    }
    $i++;
}
# check for mandatory fields
$Missing = 0;
if($HHField == -1) {
    print "Missing HH.\n";
    $Missing++;
}
if($TravField == -1) {
    print "Missing TRAV.\n";
    $Missing++;
}
if($TourField == -1) {
    print "Missing TOUR.\n";
    $Missing++;
}
if($SubTField == -1) {
    print "Missing SUBTOUR.\n";
    $Missing++;
}
if($Act1Field == -1) {
    print "Missing START_ACT_ID.\n";
    $Missing++;
}
```

```
if($Act2Field == -1) {
    print "Missing END_ACT_ID.\n";
    $Missing++;
}
if($IncomeField == -1) {
    print "Missing RHHINC.\n";
    $Missing++;
}
if($AgeField == -1) {
    print "Missing AGE.\n";
    $Missing++;
}
if($ActLoc1Field == -1) {
    print "Missing START_ACT_LOCATION.\n";
    $Missing++;
}
if($ActLoc2Field == -1) {
    print "Missing END_ACT_LOCATION.\n";
    $Missing++;
}
if($TypeField == -1) {
    print "Missing END_ACT_TYPE.\n";
    $Missing++;
}
if($ModeField == -1) {
    print "Missing END_MODE_PREF.\n";
    $Missing++;
}
if($SharField == -1) {
    print "Missing END_OTHER_PARTICIPANTS.\n";
    $Missing++;
}
if($DurUBField == -1) {
    print "Missing END_DUR_UB.\n";
    $Missing++;
}
if($DurLBField == -1) {
    print "Missing END_DUR_LB.\n";
    $Missing++;
}
if($DriverField == -1) {
    print "Missing DRIVER_ID.\n";
    $Missing++;
}
if($ParkingField == -1) {
    print "Missing END_ACT_USER_DATA_PARKING_ZN.\n";
    $Missing++;
}
if($Urban1Field == -1) {
    print "Missing START_ACT_USER_DATA_URBAN_TYPE.\n";
    $Missing++;
```

```
}
if($Urban2Field == -1) {
    print "Missing END_ACT_USER_DATA_URBAN_TYPE.\n";
    $Missing++;
}
if($River1Field == -1) {
    print "Missing START_ACT_USER_DATA_River_Zone.\n";
    $Missing++;
}
if($River2Field == -1) {
    print "Missing END_ACT_USER_DATA_River_Zone.\n";
    $Missing++;
}
if($Tran1Field == -1) {
    print "Missing START_ACT_USER_DATA_Tran_Dist.\n";
    $Missing++;
}
if($Tran2Field == -1) {
    print "Missing END_ACT_USER_DATA_Tran_Dist.\n";
    $Missing++;
}
if($Zone1Field == -1) {
    print "Missing START_ACT_USER_DATA_Zones_8.\n";
    $Missing++;
}
if($Zone2Field == -1) {
    print "Missing END_ACT_USER_DATA_Zones_8.\n";
    $Missing++;
}
if($TimeField == -1) {
    print "Missing DURATION.\n";
    $Missing++;
}
if($DistField == -1) {
    print "Missing EUCLID.\n";
    $Missing++;
}
if($ModeStringField == -1) {
    print "Missing MODE_STRING.\n";
    $Missing++;
}
if($Missing > 0) {exit;}

# open output files, print new header
open(TOURITDB, ">$touritdb") || die "Failed to open $touritdb.";
print TOURITDB "HH\tTrav\tIncome\tAge\t";
print TOURITDB "Tour\tTourAct\tType\tMode\tModePref\tTime\tDistance\tMulitMode\tShared\tSubTours";
print TOURITDB "\tCrossColumbia\tCrossWillamette\tRailOnly\tMaxTranDist\tModeString\t";
print TOURITDB "HomeZone\tHomeRiver\tHomeUrban\tHomeTranDist\t";
print TOURITDB "AnchorAct\tAnchorZone\tAnchorRiver\tAnchorUrban\tAnchorParking\tAnchorTranDist\tAnchorMod\n";
```

```
#####################################################

$Tour = -1;
$NotFirstTrav = 0;

# loop over lines
while ( <ORIGITDB> ) {
    $line = $_; chomp $line;
    @data = split($delim, $line);
    if($data[$Act1Field] != $data[$Act2Field]) { # skip "activity" entries
        if($Tour == $data[$TourField] && $Trav == $data[$TravField]) { # same tour
            if($data[$ModeField] != $CurrentMode) { $MultiMode = 1; }
            if($data[$ModeField] == 2 && $data[$River1Field] != $data[$River2Field]) {
                if($data[$River1Field] == 1 || $data[$River2Field] == 1) { $River1++; }
                else { $River2++; }
            }
            $CurrentMode = $data[$ModeField];
            $LastLoc = $data[$ActLoc2Field];
            if($data[$ModeField] == 4) { $RailOnly++; }
            if($MaxTranDist < $data[$Tran2Field]) { $MaxTranDist = $data[$Tran2Field]; }
            if($data[$SubTField] == 0) { # ignore sub-tours
                $GoodTour = ($data[$TimeField] ne "NA") ? $GoodTour : 0; # check Router field
                if($GoodTour == 1) { $Time += $data[$TimeField]; } # can't accumulate "NA"
                $Dist += $data[$DistField];
                $ModeString .= $data[$ModeStringField];
                $ActMode .= $data[$ModeField];
                if($data[$SharField] != 0) {
                    if($data[$DriverField] == $Trav) { $Shared = 4; } # main-tour shared-ride driver
                    elsif($Shared < 2) {$Shared = 2;} # main-tour shared-ride passenger
                }
                if($LastLoc != $HomeLoc) { # final at-home activity cannot be anchor
                    $ActivDuration = ($data[$DurLBField] + $data[$DurUBField])/2;
                    $NewAnchor = 0;
                    # work-school--or-college anchor:
                    if($data[$TypeField] == 1 || $data[$TypeField] == 7 || $data[$TypeField] == 8) {
                        if($TourType ne "work" || $ActivDuration > $AnchorDuration) {
                            $NewAnchor = 1; # better anchor type, or better work anchor
                            $TourType = "work";
                        }
                    }
                    elsif($TourType ne "work" && $data[$TypeField] == 2) { # shop anchor
                        if($TourType ne "shop" || $ActivDuration > $AnchorDuration) {
                            $NewAnchor = 1; # better anchor type, or better shop anchor
                            $TourType = "shop";
                        }
                    }
                    elsif($ActivDuration > $AnchorDuration) { # "other" type anchor, default
                        $NewAnchor = 1; # better anchor
                    } # note precedence of w, s, then o anchor locations
                    if($NewAnchor == 1) {
                        # get new anchor data
```

```
                $AnchorAct = $data[$Act2Field];
                $AnchorZone = $data[$Zone2Field];
                $AnchorRiver = $data[$River2Field];
                $AnchorUrban = $data[$Urban2Field];
                $AnchorParking = $data[$ParkingField];
                $AnchorTranDist = $data[$Tran2Field];
                $AnchorDuration = $ActivDuration;
                $AnchorMode = $CurrentMode;
            }
        } # ingnore at-home activities
    } # not a sub-tour
    else { # is a subtour
        if($data[$SharField] != 0) {
            if($data[$DriverField] == $Trav && $Shared < 3) { $Shared = 3; } # sub-tour shared-ride driver
            elsif($Shared == 0) {$Shared = 1;} # sub-tour shared-ride passenger
        }
        # count
        $Nsubtours++;
    }
} # same tour
else { # new tour or HH
    if($NotFirstTrav == 1) {
        # first determine tour-mode

        $TmpModeString = $ModeString;
        # test for illegal modes
        if($TmpModeString =~ /U/ || $TmpModeString =~ /NA/ || $TmpModeString =~ /p/ || $TmpModeString =~ /t/) {
            $TourMode = -1; # "bad mode"
        }
        else {
            # then get rid of magic k & K:
            $TmpModeString =~ s/k/w/g;
            $TmpModeString =~ s/K/w/g;
            # ignore bike for now:
            $TmpModeString =~ s/i/w/g;
            # determine mode for tour
            if($TmpModeString =~ /c/) {
                if($TmpModeString =~ /l/ || $TmpModeString =~ /b/) {
                    $TourMode = 6; # "possible P&R mode"
                }
                else {
                    $TourMode = 2; # "car-only mode"
                }
            }
            elsif($TmpModeString =~ /b/) {
                if($TmpModeString =~ /l/) {
                    $TourMode = 5; # "mixed-transit mode"
                }
                else {
                    $TourMode = 3; # "bus-only mode"
                }
```

```
        }
        elsif($TmpModeString =~ /l/) {
            $TourMode = 4; # "rail-only mode"
        }
        elsif($TmpModeString =~ /w/) {
            $TourMode = 1; # "walk-only mode (includes k,K,i)"
        }
    }

    #determine tour mode preference (activity modes)
    $TmpModeString = $ActMode;
    # test for illegal modes
    if($TmpModeString =~ /NA/) {
        $TourModePref = -1; # "bad mode"
    }
    else {
        # then get rid of magic k & K:
        $TmpModeString =~ s/8/1/g;
        $TmpModeString =~ s/9/1/g;
        # ignore bike for now:
        $TmpModeString =~ s/7/1/g;
        # determine mode for tour
        if($TmpModeString =~ /2/) {
            if($TmpModeString =~ /6/ || $TmpModeString =~ /5/
               || $TmpModeString =~ /4/ || $TmpModeString =~ /3/) {
                $TourModePref = 6; # "possible P&R mode"
            }
            else {
                $TourModePref = 2; # "car-only mode"
            }
        }
        elsif($TmpModeString =~ /3/) {
            if($TmpModeString =~ /4/) {
                $TourModePref = 5; # "mixed-transit mode"
            }
            else {
                $TourModePref = 3; # "bus-only mode"
            }
        }
        elsif($TmpModeString =~ /4/) {
            $TourModePref = 4; # "rail-only mode"
        }
        elsif($TmpModeString =~ /1/) {
            $TourModePref = 1; # "walk-only mode (includes k,K,i)"
        }
    }


    # print info from last tour
    print TOURITDB $HH,"\t",$Trav,"\t",$Income,"\t",$Age,"\t";
    print TOURITDB $Tour,"\t",$TourAct,"\t",$TourType,"\t",$TourMode,"\t",$TourModePref,"\t";
```

```
        if($GoodTour == 1 && $LastLoc == $HomeLoc) { # only OK if they had times & got back home
            print TOURITDB $Time;
        }
        else {
            print TOURITDB "NA";
        }
        print TOURITDB "\t",$Dist,"\t",$MultiMode,"\t",$Shared,"\t",$Nsubtours,"\t";
        print TOURITDB $River1,"\t",$River2,"\t",$RailOnly,"\t",$MaxTranDist,"\t",$ModeString,"\t";
        print TOURITDB $HomeZone,"\t",$HomeRiver,"\t",$HomeUrban,"\t",$HomeTranDist,"\t";
        print TOURITDB $AnchorAct,"\t",$AnchorZone,"\t",$AnchorRiver,"\t",$AnchorUrban,"\t";
        print TOURITDB $AnchorParking,"\t",$AnchorTranDist,"\t",$AnchorMode,"\n";
    }
    $NotFirstTrav = 1;

    # Traveler data
    $HH     = $data[$HHField];
    $Trav   = $data[$TravField];
    $Income = $data[$IncomeField];
    $Age    = $data[$AgeField];
    # Tour data
    $Tour    = $data[$TourField];
    $TourAct = $data[$Act2Field];
    $GoodTour = ($data[$TimeField] ne "NA") ? 1 : 0;
    if($GoodTour == 1) { $Time = $data[$TimeField]; }
    $Dist = $data[$DistField];
    $ModeString = $data[$ModeStringField];
    $ActMode = $data[$ModeField];
    if($data[$SharField] != 0) {
        if($data[$DriverField] == $Trav) { $Shared = 4; } # main-tour shared-ride driver
        else {$Shared = 2;} # main-tour shared-ride passenger
    }
    else {$Shared = 0;}
    $CurrentMode = $data[$ModeField];
    $MultiMode = 0;
    $AnchorDuration = 0;
    $Nsubtours = 0;
    if($data[$ModeField] == 2 && $data[$River1Field] != $data[$River2Field]) {
        if($data[$River1Field] == 1 || $data[$River2Field] == 1) { $River1 = 1; $River2 = 0; }
        else { $River1 = 0; $River2 = 1; }
    }
    else { $River1 = 0; $River2 = 0; }
    if($data[$ModeField] == 4) { $RailOnly = 1; }
    else { $RailOnly = 0; }
    $MaxTranDist = ($data[$Tran1Field] > $data[$Tran2Field]) ? $data[$Tran1Field] : $data[$Tran2Field];
    # Home data
    $HomeLoc = $data[$ActLoc1Field];
    $HomeZone = $data[$Zone1Field];
    $HomeRiver = $data[$River1Field];
    $HomeUrban = $data[$Urban1Field];
    $HomeTranDist = $data[$Tran1Field];
    # Anchor data
```

```
            $AnchorAct = $data[$Act2Field];
            $AnchorZone = $data[$Zone2Field];
            $AnchorRiver = $data[$River2Field];
            $AnchorUrban = $data[$Urban2Field];
            $AnchorParking = $data[$ParkingField];
            $AnchorTranDist = $data[$Tran2Field];
            $AnchorDuration = ($data[$DurLBField] + $data[$DurUBField])/2;
            $AnchorMode = $CurrentMode;
            # work-school-or-college anchor:
            if($data[$TypeField] == 1 || $data[$TypeField] == 7 || $data[$TypeField] == 8) {
                $TourType = "work";
            }
            elsif($data[$TypeField] == 2) { # shop anchor
                $TourType = "shop";
            }
            else { # "other" type anchor, default
                $TourType = "other";
            } # note precedence of w, s, then o anchor locations
        } # new tour
    } # skip "activity" entries
}

#######################################################

# first determine tour-mode

$TmpModeString = $ModeString;
# test for illegal modes
if($TmpModeString =~ /U/ || $TmpModeString =~ /NA/ || $TmpModeString =~ /p/ || $TmpModeString =~ /t/) {
    $TourMode = -1; # "bad mode"
}
else {
    # then get rid of magic k & K:
    $TmpModeString =~ s/k/w/g;
    $TmpModeString =~ s/K/w/g;
    # ignore bike for now:
    $TmpModeString =~ s/i/w/g;
    # determine mode for tour
    if($TmpModeString =~ /c/) {
        if($TmpModeString =~ /l/ || $TmpModeString =~ /b/) {
            $TourMode = 6; # "possible P&R mode"
        }
        else {
            $TourMode = 2; # "car-only mode"
        }
    }
    elsif($TmpModeString =~ /b/) {
        if($TmpModeString =~ /l/) {
            $TourMode = 5; # "mixed-transit mode"
        }
        else {
```

```
            $TourMode = 3; # "bus-only mode"
        }
    }
    elsif($TmpModeString =~ /l/) {
        $TourMode = 4; # "rail-only mode"
    }
    elsif($TmpModeString =~ /w/) {
        $TourMode = 1; # "walk-only mode (includes k,K,i)"
    }
}

#determine tour mode preference (activity modes)
$TmpModeString = $ActMode;
# test for illegal modes
if($TmpModeString =~ /NA/) {
    $TourModePref = -1; # "bad mode"
}
else {
    # then get rid of magic k & K:
    $TmpModeString =~ s/8/1/g;
    $TmpModeString =~ s/9/1/g;
    # ignore bike for now:
    $TmpModeString =~ s/7/1/g;
    # determine mode for tour
    if($TmpModeString =~ /2/) {
        if($TmpModeString =~ /6/ || $TmpModeString =~ /5/
           || $TmpModeString =~ /4/ || $TmpModeString =~ /3/) {
            $TourModePref = 6; # "possible P&R mode"
        }
        else {
            $TourModePref = 2; # "car-only mode"
        }
    }
    elsif($TmpModeString =~ /3/) {
        if($TmpModeString =~ /4/) {
            $TourModePref = 5; # "mixed-transit mode"
        }
        else {
            $TourModePref = 3; # "bus-only mode"
        }
    }
    elsif($TmpModeString =~ /4/) {
        $TourModePref = 4; # "rail-only mode"
    }
    elsif($TmpModeString =~ /1/) {
        $TourModePref = 1; # "walk-only mode (includes k,K,i)"
    }
}

# print info from last tour
print TOURITDB $HH,"\t",$Trav,"\t",$Income,"\t",$Age,"\t";
```

```
print TOURITDB $Tour,"\t",$TourAct,"\t",$TourType,"\t",$TourMode,"\t",$TourModePref,"\t";
if($GoodTour == 1 && $LastLoc == $HomeLoc) { # only OK if they had times & got back home
    print TOURITDB $Time;
}
else {
    print TOURITDB "NA";
}
print TOURITDB "\t",$Dist,"\t",$MultiMode,"\t",$Shared,"\t",$Nsubtours,"\t";
print TOURITDB $River1,"\t",$River2,"\t",$RailOnly,"\t",$MaxTranDist,"\t",$ModeString,"\t";
print TOURITDB $HomeZone,"\t",$HomeRiver,"\t",$HomeUrban,"\t",$HomeTranDist,"\t";
print TOURITDB $AnchorAct,"\t",$AnchorZone,"\t",$AnchorRiver,"\t",$AnchorUrban,"\t";
print TOURITDB $AnchorParking,"\t",$AnchorTranDist,"\t",$AnchorMode,"\n";

close ORIGITDB;
close TOURITDB;

exit
```

2.8 ReModeONDN.pl

```
#! /usr/local/bin/perl -w

#######################################################################

# NOTE: This takes a TOUR-based ONDN itdb as input

# - randomly select specified fraction of tours
# - create a new itdb with only selected entries
# - create a new itdb with only un-selected entries
# - create activity feedback file with re-mode commands
# - create a router household file for router feedback

# NOTE: This assumes the only modes in the acivity file are 2 or 3.

#######################################################################

$delim = "\t";
$heads = 1;

# get command line arguments:
if($#ARGV+1 != 8) {
    print "\nusage: ReModeONDN.pl SEED FRAC MIN TOURDB NEWTDB1 NEWTDB2 ROUTEFB ACTIVFB\n\n";
    print "   SEED = seed to the random number generator that selects traveler-tours\n";
    print "   FRAC = re-mode fraction, e.g.: 0.5 selects 50% of ONDN traveler-tours\n";
    print "   MIN = minimum number of remoded traveler  -  NOT YET ENABLED\n";
    print "   TOURDB = input tour-based, tab delimited iteration database\n";
    print "   NEWTDB1 = as origitdb, but only selected tours\n";
    print "   NEWTDB2 = as origitdb, but only unselected tours\n";
    print "   ROUTEFB = main household file for re-routing selected traveler-tours\n";
    print "   ACTIVFB = main activity regenerator feedback command file\n";
    exit;
}
local($seed, $frac, $minnum, $origitdb, $outitdb1, $outitdb2, $routefile, $activfile) = @ARGV;

# set random number seed
srand($seed);
$minnum = 100;

# open input itdb and parse header
open(ORIGITDB, "$origitdb") || die "Failed to open $origitdb.";
# read header lines
for($i=0;$i<$heads;$i++) {
    $line = <ORIGITDB>;
}

# find fields automatically using header
```

```
chomp $line;
@data = split($delim, $line);
$i=0;
$HHField = $TravField = $ModeField = $ActField = -1;
while($data[$i]) {
    if($data[$i] eq "HH") {
        $HHField = $i;
    }
    elsif($data[$i] eq "Trav") {
        $TravField = $i;
    }
    elsif($data[$i] eq "Mode") {
        $ModeField = $i;
    }
    elsif($data[$i] eq "AnchorAct") {
        $ActField = $i;
    }
    $i++;
}
# check for mandatory fields
$Missing = 0;
if($HHField == -1) {
    print "Missing HH.\n";
    $Missing++;
}
if($TravField == -1) {
    print "Missing Trav.\n";
    $Missing++;
}
if($ModeField == -1) {
    print "Missing Mode.\n";
    $Missing++;
}
if($ActField == -1) {
    print "Missing AnchorAct.\n";
    $Missing++;
}
if($Missing > 0) {exit;}


# open output files
open(NEWITDB1, ">$outitdb1") || die "Failed to open $outitdb1.";
open(NEWITDB2, ">$outitdb2") || die "Failed to open $outitdb2.";
open(ROUTEFB, ">$routefile") || die "Failed to open $routefile.";
open(ACTIVFB, ">$activfile") || die "Failed to open $activfile.";

# print new tour db headers
print NEWITDB1 $line,"\n";
print NEWITDB2 $line,"\n";

###########################
```

```
$Trav = -991;
$HH = -999;

# loop over lines
while ( <ORIGITDB> ) {
    $line = $_; chomp $line;
    @data = split($delim, $line);

    # only one per trav; selected trav, :
    if( $Trav != $data[$TravField] && rand() <= $frac ) {

        $Trav = $data[$TravField];

        # print to new itdb of tested tours
        print NEWITDB1 $line,"\n";

        # send HH to router feedback file if not already sent
        if($HH != $data[$HHField]) {
            $HH = $data[$HHField];
            print ROUTEFB $HH,"\n";
        }

        # send mode-change feedback command to activity feedback file
        if($data[$ModeField]==2) {
            print ACTIVFB $HH," ",$data[$ActField]," M 3 2\n";
        }
        else {
            print ACTIVFB $HH," ",$data[$ActField]," M 2 3\n";
        }
    }
    else {
        # print to new itdb of untested tours
        print NEWITDB2 $line,"\n";
    }
}

###########################

close ORIGITDB;
close NEWITDB1;
close NEWITDB2;
close ROUTEFB;
close ACTIVFB;

###########################

exit;
```

2.9 ReModeTrans.pl

```
#! /usr/local/bin/perl -w

#######################################################################

# NOTE: This takes a TOUR-based itdb as input

# This script creates feedback commands that will convert ALL rail or
# walk trips in the tours in a tour database into regular transit
# trips.

# - create activity feedback file with re-mode commands
# - create a router household file for router feedback

#######################################################################

$delim = "\t";
$heads = 1;

# get command line arguments:
if($#ARGV+1 != 3) {
    print "\nusage: ReModeTrans.pl TOURITDB ROUTEFB ACTIVFB\n\n";
    print "   TOURITDB = input tour-based, tab delimited iteration database\n";
    print "   ROUTEFB  = household file changing w,l-trips into t-trips\n";
    print "   ACTIVFB  = activity regenerator feedback command file\n\n";
    exit;
}
local($origitdb, $routefile, $activfile) = @ARGV;

# open input itdb and parse header
open(ORIGITDB, "$origitdb") || die "Failed to open $origitdb.";
# read header lines
for($i=0;$i<$heads;$i++) {
    $line = <ORIGITDB>;
}

# find fields automatically using header
chomp $line;
@data = split($delim, $line);
$i=0;
$HHField = $ActField = -1;
while($data[$i]) {
    if($data[$i] eq "HH") {
        $HHField = $i;
    }
    elsif($data[$i] eq "AnchorAct") {
        $ActField = $i;
    }
    $i++;
```

```
}
# check for mandatory fields
$Missing = 0;
if($HHField == -1) {
    print "Missing HH.\n";
    $Missing++;
}
if($ActField == -1) {
    print "Missing AnchorAct.\n";
    $Missing++;
}
if($Missing > 0) {exit;}


# open output files
open(ROUTEFB, ">$routefile") || die "Failed to open $routefile.";
open(ACTIVFB, ">$activfile") || die "Failed to open $activfile.";

###########################

$HH2 = -999;

# loop over lines
while ( <ORIGITDB> ) {
    $line = $_; chomp $line;
    @data = split($delim, $line);

    $HH = $data[$HHField];

    # send mode-change feedback command to activity feedback file
    print ACTIVFB $HH," ",$data[$ActField]," M 3 1 4\n";
    # send HH to router feedback file if not already sent
    if($HH2 != $HH) {
        $HH2 = $HH;
        print ROUTEFB $HH,"\n";
    }
}

###########################

close ORIGITDB;
close ROUTEFB;
close ACTIVFB;

###########################
exit
```

2.10 run.csh

```
#!/bin/csh -f

#####################################################################

# This script runs the Router in distributed fashion on rockhopper.

# This version uses CPUs between $2 and $3 in the (hard-wired)
# machines file as the allowed range of nodes.  If $4 is present, that
# is the starting suffix for the jobs (e.g. $4==6 means first job uses
# *.tAF).  Any CPU that doesn't respond to ping is skipped, but that
# would mean that one less job is started.  In that case re-run using
# $2==$3, and $4!="".  This version also keeps the jobs in order on
# the CPUs.  That is, if num==6 is skipped, then there will be no
# num==6 job started.

#####################################################################

if( $3 == "" ) then
    echo ""
    echo "usage:  $0 CONFIG FIRSTCPU LASTCPU [STARTJOBID]"
    echo ""
    exit
endif

#######################################################

set TRANSIMS_HOME = /n/father/transims/CaseStudy3
set ROUTERDIR = $PWD
set TENTO26 = $TRANSIMS_HOME/bin/10to26
set MACHFILE = /n/father/transims/CaseStudy3/scenarios/allstr/feedback/config/machines
set ROUTER = /n/father/transims/CaseStudy3/bin/Router

set SSH = "ssh -f" # "rsh" #
set DATE = `date +%y%m%d.%H:%M`

#######################################################

set CONFIG = $1
# check that config file exists, and make sure it has a full pathname
if ( $CONFIG == `echo $CONFIG | awk '{print $NF}' FS='/'` ) then
    echo \# redirecting CONFIG variable to \$PWD.
    set CONFIG = `pwd`/$CONFIG
endif
if ( ! -e $CONFIG ) then
    echo ""; echo Cannot open config file \"$CONFIG\" for input.;
    echo ""; exit
endif
```

```
# set start node, end node, and job counter starting point
@ start = $2 - 1
@ end = $3 - 1
if($4 != "") then
    set num = $4
else
    set num = 0
endif
set count = 0
set total = `echo $end-$start+1 | bc`
set fix = `echo $total-1+$num | bc`

#######################################################

cd $ROUTERDIR
date >! $ROUTERDIR/router.log.$DATE

echo "# Starting $total jobs beginning at CPU line $start in machinefile"
foreach f (`tail +3 $MACHFILE`)
    if ($count >= $start && $count <= $end) then
        set result = `ping -q -c 2 -i 5 $f | grep "0 packets received"`
        if ( "X$result" == "X" ) then
            echo $f $num `$TENTO26 $num`
            echo $f $num `$TENTO26 $num` >>& $ROUTERDIR/router.log.$DATE

            touch $ROUTERDIR/logs/router.log.t`$TENTO26 $num`.$DATE
            $SSH -n $f time $ROUTER $CONFIG $num \
                >>& $ROUTERDIR/logs/router.log.t`$TENTO26 $num`.$DATE
        else
            echo "# ERROR: Skipping $f"
            echo "# * Fix $f, then run: RunRouter.csh $count $count $num"
            set fix = `echo $fix-1 | bc`
        endif
        @ num++
    endif
    @ count++
end

if($num<$total) echo "# ERROR: Ran out of CPUs in file."

#######################################################
# jps 20010322; modified: 20010508
```

2.11 RunCollator.csh

```
#!/bin/csh -f

#######################################################################

# This script runs the Collator in distributed fashion on rockhopper.

# This version uses CPUs between $2 and $3 in the (hard-wired)
# machines file as the allowed range of nodes.  If $4 is present, that
# is the starting suffix for the jobs (e.g. $4==6 means first job uses
# *.tAF).  Any CPU that doesn't respond to ping is skipped, but that
# would mean that one less job is started.  In that case re-run using
# $2==$3, and $4!="".  This version also keeps the jobs in order on
# the CPUs.  That is, if num==6 is skipped, then there will be no
# num==6 job started.

#######################################################################

if( $3 == "" ) then
    echo ""
    echo "usage:  $0 CONFIG FIRSTCPU LASTCPU [STARTJOBID]"
    echo ""
    exit
endif

#######################################################

set TRANSIMS_HOME = /n/father/transims/CaseStudy3
set RUNDIR = $PWD
set TENTO26 = $TRANSIMS_HOME/bin/10to26
set MACHFILE = /n/father/transims/CaseStudy3/scenarios/allstr/feedback/config/machines
set COLLATOR = /n/father/transims/CaseStudy3/bin/Collator

set SSH = "ssh -f" # "rsh" #
set DATE = `date +%y%m%d.%H:%M`

#######################################################

set CONFIG = $1
# check that config file exists, and make sure it has a full pathname
if ( $CONFIG == `echo $CONFIG | awk '{print $NF}' FS='/'` ) then
    echo \# redirecting CONFIG variable to \$PWD.
    set CONFIG = `pwd`/$CONFIG
endif
if ( ! -e $CONFIG ) then
    echo ""; echo Cannot open config file \"$CONFIG\" for input.;
    echo ""; exit
```

```
endif

# set start node, end node, and job counter starting point
@ start = $2 - 1
@ end = $3 - 1
if($4 != "") then
    set num = $4
else
    set num = 0
endif
set count = 0
set total = `echo $end-$start+1 | bc`
set fix = `echo $total-1+$num | bc`

#######################################################

cd $RUNDIR
date >! $RUNDIR/collator.log.$DATE

echo "# Starting $total jobs beginning at CPU line $start in machinefile"
foreach f (`tail +3 $MACHFILE`)
    if ($count >= $start && $count <= $end) then
        set result = `ping -q -c 2 -i 5 $f | grep "0 packets received"`
        if ( "X$result" == "X" ) then
            echo $f $num `$TENTO26 $num`
            echo $f $num `$TENTO26 $num` >>& $RUNDIR/collator.log.$DATE

            touch $RUNDIR/logs/collator.log.t`$TENTO26 $num`.$DATE
            $SSH -n $f time $COLLATOR $CONFIG $num \
                >>& $RUNDIR/logs/collator.log.t`$TENTO26 $num`.$DATE
        else
            echo "# ERROR: Skipping $f"
            echo "# * Fix $f, then run: RunCollator.csh $count $count $num"
            set fix = `echo $fix-1 | bc`
        endif
        @ num++
    endif
    @ count++
end

if($num<$total) echo "# ERROR: Ran out of CPUs in file."

#######################################################
# jps: 20010713
```

2.12 RunRouter.pl

```
#!/bin/csh -f

#####################################################################

# This script runs the Router in distributed fashion on rockhopper.

# This version uses CPUs between $2 and $3 in the (hard-wired)
# machines file as the allowed range of nodes.  If $4 is present, that
# is the starting suffix for the jobs (e.g. $4==6 means first job uses
# *.tAF).  Any CPU that doesn't respond to ping is skipped, but that
# would mean that one less job is started.  In that case re-run using
# $2==$3, and $4!="".  This version also keeps the jobs in order on
# the CPUs.  That is, if num==6 is skipped, then there will be no
# num==6 job started.

#####################################################################

if( $3 == "" ) then
    echo ""
    echo "usage:  $0 CONFIG FIRSTCPU LASTCPU [STARTJOBID]"
    echo ""
    exit
endif

#######################################################

set TRANSIMS_HOME = /n/father/transims/CaseStudy3
set ROUTERDIR = $PWD
set TENTO26 = $TRANSIMS_HOME/bin/10to26
set MACHFILE = /n/father/transims/CaseStudy3/scenarios/allstr/feedback/config/machines
set ROUTER = /n/father/transims/CaseStudy3/bin/Router

set SSH = "ssh -f" # "rsh" #
set DATE = `date +%y%m%d.%H:%M`

#######################################################

set CONFIG = $1
# check that config file exists, and make sure it has a full pathname
if ( $CONFIG == `echo $CONFIG | awk '{print $NF}' FS='/'` ) then
    echo \# redirecting CONFIG variable to \$PWD.
    set CONFIG = `pwd`/$CONFIG
endif
if ( ! -e $CONFIG ) then
    echo ""; echo Cannot open config file \"$CONFIG\" for input.;
    echo ""; exit
```

```
endif

# set start node, end node, and job counter starting point
@ start = $2 - 1
@ end = $3 - 1
if($4 != "") then
    set num = $4
else
    set num = 0
endif
set count = 0
set total = `echo $end-$start+1 | bc`
set fix = `echo $total-1+$num | bc`

#######################################################

cd $ROUTERDIR
date >! $ROUTERDIR/router.log.$DATE

echo "# Starting $total jobs beginning at CPU line $start in machinefile"
foreach f (`tail +3 $MACHFILE`)
    if ($count >= $start && $count <= $end) then
        set result = `ping -q -c 2 -i 5 $f | grep "0 packets received"`
        if ( "X$result" == "X" ) then
            echo $f $num `$TENTO26 $num`
            echo $f $num `$TENTO26 $num` >>& $ROUTERDIR/router.log.$DATE

            touch $ROUTERDIR/logs/router.log.t`$TENTO26 $num`.$DATE
            $SSH -n $f time $ROUTER $CONFIG $num \
                >>& $ROUTERDIR/logs/router.log.t`$TENTO26 $num`.$DATE
        else
            echo "# ERROR: Skipping $f"
            echo "# * Fix $f, then run: RunRouter.csh $count $count $num"
            set fix = `echo $fix-1 | bc`
        endif
        @ num++
    endif
    @ count++
end

if($num<$total) echo "# ERROR: Ran out of CPUs in file."

#######################################################
# jps 20010322; modified: 20010508
```

2.13 StratifyONDN.pl

```
#! /usr/local/bin/perl -w

#######################################################################

# This script takes any tour-based database and creates the 18
# separate files corresponding to the 18 strata decided upon by METRO:
#   1) work / non-work
#   2) home urbanization: (very urban, urban, sub-urban) = (1,2,3)
#   3) primary anchor urbanization: very urban, urban, sub-urban

# If no 2nd argument is given, then no output is made to files, but
# the counts-by-bin are sent to standard output.

# NOTE: This script will work independent of optional fields.

#######################################################################

$heads = 1;
$delim = "\t";
$Output = 0;

# get command line arguments:
if($#ARGV+1 == 1) {
    local($origitdb) = @ARGV;
}
elsif($#ARGV+1 == 2) {
    local($origitdb, $outitdb) = @ARGV;
    $Output = 1;
}
else {
    print "\nusage: StratifyDB.pl TOURITDB [ NEWDBPREFIX ]\n\n";
    print "   TOURITDB = input tour-based, tab delimited iteration database\n";
    print "   NEWDBPREFIX = output as TOURITDB, but only selected tours\n";
    print "                 in each of 18 files - one for each stratum.\n\n";
    exit;
}

local($origitdb, $outitdb) = @ARGV;

# open input itdb and parse header
open(ORIGITDB, "$origitdb") || die "Failed to open $origitdb.";
# read header lines
for($i=0;$i<$heads;$i++) {
    $line = <ORIGITDB>;
}

# find fields automatically using header
chomp $line;
```

```
@data = split($delim, $line);
$i=0;
$TypeField = $Urban1Field = $Urban2Field = -1;
while($data[$i]) {
    if($data[$i] eq "Type") {
        $TypeField = $i;
    }
    elsif($data[$i] eq "HomeUrban") {
        $Urban1Field = $i;
    }
    elsif($data[$i] eq "AnchorUrban") {
        $Urban2Field = $i;
    }
    $i++;
}
# check for mandatory fields
$Missing = 0;
if($TypeField == -1) {
    print "Missing Type.\n";
    $Missing++;
}
if($Urban1Field == -1) {
    print "Missing HomeUrban.\n";
    $Missing++;
}
if($Urban2Field == -1) {
    print "Missing AnchorUrban.\n";
    $Missing++;
}
if($Missing > 0) {exit;}


#######################################################

if($Output == 1) { # stratify DB into files

    # open output files - brute force for now
    open(NEWITDBw11, ">$outitdb.w.1.1") || die "Failed to open $outitdb.w.1.1.";
    open(NEWITDBw12, ">$outitdb.w.1.2") || die "Failed to open $outitdb.w.1.2.";
    open(NEWITDBw13, ">$outitdb.w.1.3") || die "Failed to open $outitdb.w.1.3.";
    open(NEWITDBw21, ">$outitdb.w.2.1") || die "Failed to open $outitdb.w.2.1.";
    open(NEWITDBw22, ">$outitdb.w.2.2") || die "Failed to open $outitdb.w.2.2.";
    open(NEWITDBw23, ">$outitdb.w.2.3") || die "Failed to open $outitdb.w.2.3.";
    open(NEWITDBw31, ">$outitdb.w.3.1") || die "Failed to open $outitdb.w.3.1.";
    open(NEWITDBw32, ">$outitdb.w.3.2") || die "Failed to open $outitdb.w.3.2.";
    open(NEWITDBw33, ">$outitdb.w.3.3") || die "Failed to open $outitdb.w.3.3.";
    open(NEWITDBn11, ">$outitdb.n.1.1") || die "Failed to open $outitdb.n.1.1.";
    open(NEWITDBn12, ">$outitdb.n.1.2") || die "Failed to open $outitdb.n.1.2.";
    open(NEWITDBn13, ">$outitdb.n.1.3") || die "Failed to open $outitdb.n.1.3.";
    open(NEWITDBn21, ">$outitdb.n.2.1") || die "Failed to open $outitdb.n.2.1.";
    open(NEWITDBn22, ">$outitdb.n.2.2") || die "Failed to open $outitdb.n.2.2.";
```

```
open(NEWITDBn23, ">$outitdb.n.2.3") || die "Failed to open $outitdb.n.2.3.";
open(NEWITDBn31, ">$outitdb.n.3.1") || die "Failed to open $outitdb.n.3.1.";
open(NEWITDBn32, ">$outitdb.n.3.2") || die "Failed to open $outitdb.n.3.2.";
open(NEWITDBn33, ">$outitdb.n.3.3") || die "Failed to open $outitdb.n.3.3.";

# print itdb header
print NEWITDBw11 $line,"\n";
print NEWITDBw12 $line,"\n";
print NEWITDBw13 $line,"\n";
print NEWITDBw21 $line,"\n";
print NEWITDBw22 $line,"\n";
print NEWITDBw23 $line,"\n";
print NEWITDBw31 $line,"\n";
print NEWITDBw32 $line,"\n";
print NEWITDBw33 $line,"\n";
print NEWITDBn11 $line,"\n";
print NEWITDBn12 $line,"\n";
print NEWITDBn13 $line,"\n";
print NEWITDBn21 $line,"\n";
print NEWITDBn22 $line,"\n";
print NEWITDBn23 $line,"\n";
print NEWITDBn31 $line,"\n";
print NEWITDBn32 $line,"\n";
print NEWITDBn33 $line,"\n";

# loop over lines
while ( <ORIGITDB> ) {
    $line = $_; chomp $line;
    @data = split($delim, $line);

    # send to appropriate file
    if($data[$TypeField] eq "work") {
        if($data[$Urban1Field] == 1) {
            if($data[$Urban2Field] == 1) {
                print NEWITDBw11 $line,"\n";
            }
            elsif($data[$Urban2Field] == 2) {
                print NEWITDBw12 $line,"\n";
            }
            else {
                print NEWITDBw13 $line,"\n";
            }
        }
        elsif($data[$Urban1Field] == 2) {
            if($data[$Urban2Field] == 1) {
                print NEWITDBw21 $line,"\n";
            }
            elsif($data[$Urban2Field] == 2) {
                print NEWITDBw22 $line,"\n";
            }
            else {
```

```
                print NEWITDBw23 $line,"\n";
            }
        }
        else { # home urban = 3
            if($data[$Urban2Field] == 1) {
                print NEWITDBw31 $line,"\n";
            }
            elsif($data[$Urban2Field] == 2) {
                print NEWITDBw32 $line,"\n";
            }
            else {
                print NEWITDBw33 $line,"\n";
            }
        }
    }
    else { # non-work
        if($data[$Urban1Field] == 1) {
            if($data[$Urban2Field] == 1) {
                print NEWITDBn11 $line,"\n";
            }
            elsif($data[$Urban2Field] == 2) {
                print NEWITDBn12 $line,"\n";
            }
            else {  # work urban = 3
                print NEWITDBn13 $line,"\n";
            }
        }
        elsif($data[$Urban1Field] == 2) {
            if($data[$Urban2Field] == 1) {
                print NEWITDBn21 $line,"\n";
            }
            elsif($data[$Urban2Field] == 2) {
                print NEWITDBn22 $line,"\n";
            }
            else {
                print NEWITDBn23 $line,"\n";
            }
        }
        else {  # home urban = 3
            if($data[$Urban2Field] == 1) {
                print NEWITDBn31 $line,"\n";
            }
            elsif($data[$Urban2Field] == 2) {
                print NEWITDBn32 $line,"\n";
            }
            else {  # work urban = 3
                print NEWITDBn33 $line,"\n";
            }
        }
    }
}
```

```
    close NEWITDBw11;
    close NEWITDBw12;
    close NEWITDBw13;
    close NEWITDBw21;
    close NEWITDBw22;
    close NEWITDBw23;
    close NEWITDBw31;
    close NEWITDBw32;
    close NEWITDBw33;
    close NEWITDBn11;
    close NEWITDBn12;
    close NEWITDBn13;
    close NEWITDBn21;
    close NEWITDBn22;
    close NEWITDBn23;
    close NEWITDBn31;
    close NEWITDBn32;
    close NEWITDBn33;
}

#####################################################

if($Output == 0) { # no output files

    # loop over lines
    while ( <ORIGITDB> ) {
        $line = $_; chomp $line;
        @data = split($delim, $line);

        # count by strata
        if($data[$TypeField] eq "work") { $work = "w"; }
        else { $work = "n"; }
        $Counts{$work}{$data[$Urban1Field]}{$data[$Urban2Field]}++;
    }

    # print results
    print "Work/N\tHomUrb\tAnchUrb\tCount\n";
    foreach $key1 (sort keys %Counts) {
        foreach $key2 (sort keys %{ $Counts{$key1} } ) {
            foreach $key3 (sort keys %{ $Counts{$key1}{$key2} } ) {
                print $key1,"\t",$key2,"\t",$key3,"\t",$Counts{$key1}{$key2}{$key3},"\n";
            }
        }
    }
}

#####################################################
close ORIGITDB;

exit;
```

2.14 TestAlpha.pl

```
#! /usr/local/bin/perl -w

#######################################################################

# This script takes the output of CombineDB.pl (a table of 2-mode
# tours with demographics) and calculates a table of costs (which are
# not output), one for each of the modes listed in the database.  Also
# as input is the value of the cost parameter alpha to be used in the
# calculation.  The table of costs is used to detemine the partial
# mode split: given the two modes in the arguments, what fraction of
# travelers tested on both those modes choose one over the other.
# This will be used for fitting alpha to a desired mode split.

# Try order alpha = 0.0001

#######################################################################

$delim = "\t";
$heads = 1;

# Assume transit cost is $1.10 on average
$TransitCost = 1.1; # dollars

# Parking Costs:
#
# Taz          Short   Long    Location
# -----        -----   ----    --------
# 1,2,10-16    2.47    4.94    CBD South of Burnside
# 3-6          1.59    3.18    CBD North of Burnside
# 43           1.38    2.76    Oregon Health Sciences University
# 510,934-936  0.80    1.59    Oregon City
# 846-847      0.00    3.03    Lloyd District
# 971-981      0.85    1.70    Vancouver WA

@ParkingCostShort = (0, 2.47, 1.59, 1.38, 0.80, 0.00, 0.85);
@ParkingCostLong  = (0, 4.94, 3.18, 2.76, 1.59, 3.03, 1.70);

# get command line arguments:
if($#ARGV+1 != 4) {
    print "\nusage: TestAlpha.pl COMBTDB MODE1 MODE2 ALPHA\n\n";
    print "   COMBTDB = input combined (2-mode) tour database\n";
    print "   MODE1 = test this mode in binary comparison\n";
    print "   MODE2 = test this mode in binary comparison\n";
    print "   ALPHA = guess for alpha parameter\n";
    exit;
}
```

```
local($combtdb, $mode1, $mode2, $alpha) = @ARGV;

###########################

# open input itdb and parse header
open(COMBTDB, "$combtdb") || die "Failed to open $combtdb.";
# read header lines
for($i=0;$i<$heads;$i++) {
    $line = <COMBTDB>;
}

# need: Income, Type, AnchorParking, Mode, Time; Mode2, Time2

# find fields automatically using header
chomp $line;
@data = split($delim, $line);
$i=0;
$IncomeField = $TypeField = $ParkingField = $ModeField = $TimeField = $Mode2Field = $Time2Field = -1;
while($data[$i]) {
    if($data[$i] eq "Income") {
        $IncomeField = $i;
    }
    elsif($data[$i] eq "Type") {
        $TypeField = $i;
    }
    elsif($data[$i] eq "AnchorParking") {
        $ParkingField = $i;
    }
    elsif($data[$i] eq "Mode") {
        $ModeField = $i;
    }
    elsif($data[$i] eq "Time") {
        $TimeField = $i;
    }
    elsif($data[$i] eq "Mode2") {
        $Mode2Field = $i;
    }
    elsif($data[$i] eq "Time2") {
        $Time2Field = $i;
    }
    $i++;
}
# check for mandatory fields
$Missing = 0;
if($IncomeField == -1) {
    print "Missing Income.\n";
    $Missing++;
}
if($TypeField == -1) {
    print "Missing Type.\n";
    $Missing++;
```

```
}
if($ParkingField == -1) {
    print "Missing AnchorParking.\n";
    $Missing++;
}
if($ModeField == -1) {
    print "Missing Mode.\n";
    $Missing++;
}
if($TimeField == -1) {
    print "Missing Time.\n";
    $Missing++;
}
if($Mode2Field == -1) {
    print "Missing Mode2.\n";
    $Missing++;
}
if($Time2Field == -1) {
    print "Missing Time2.\n";
    $Missing++;
}
if($Missing > 0) {exit;}

#####################################################
# record all viable data

# cost = alpha * log(Income) * Time + AnchorParkingCost + TransitCost

$NumEntries=0;
while ( <COMBTDB> ) {
    $line = $_; chomp $line;
    @data = split($delim, $line);

    # deal with "NA" from Router by making it 24 hour tour
    if($data[$TimeField] ne "NA") {
        $T1 = $data[$TimeField];
    }
    else {
        $T1 = 86400;
    }
    if($data[$Time2Field] ne "NA") {
        $T2 = $data[$Time2Field];
    }
    else {
        $T2 = 86400;
    }

    # check for valid modes
    if($data[$ModeField] == $mode1 && $data[$Mode2Field] == $mode2) {
        $order = 1;
    }
```

```
elsif($data[$Mode2Field] == $mode1 && $data[$ModeField] == $mode2) {
    $order = -1;
}
else {
    next; # should skip to next line of data
}

# calc partial costs
if($data[$ModeField] == 3 || $data[$ModeField] == 4) {
    $dollars1 = $TransitCost;
}
elsif($data[$ModeField] == 2) { # auto
    if($data[$TypeField] eq "work") {
        $dollars1 = $ParkingCostLong[$data[$ParkingField]];
    }
    else { # non-work
        $dollars1 = $ParkingCostShort[$data[$ParkingField]];
    }
}
else {
    $dollars1 = 0;
}

if($data[$Mode2Field] == 3 || $data[$Mode2Field] == 4) {
    $dollars2 = $TransitCost;
}
elsif($data[$Mode2Field] == 2) { # auto
    if($data[$TypeField] eq "work") {
        $dollars2 = $ParkingCostLong[$data[$ParkingField]];
    }
    else { # non-work
        $dollars2 = $ParkingCostShort[$data[$ParkingField]];
    }
}
else {
    $dollars2 = 0;
}

# correct for bad (negative) income
$Income2 = ($data[$IncomeField] < 0) ? 0 : $data[$IncomeField];

# record entry so that first one is for mode1
if($order == 1) {
    $CostData1{$NumEntries} = log(1.01+$Income2) * $T1;
    $CostData2{$NumEntries} = $dollars1;
    $CostData3{$NumEntries} = log(1.01+$Income2) * $T2;
    $CostData4{$NumEntries} = $dollars2;
}
else { # reverse entries
    $CostData3{$NumEntries} = log(1.01+$Income2) * $T1;
    $CostData4{$NumEntries} = $dollars1;
```

```
        $CostData1{$NumEntries} = log(1.01+$Income2) * $T2;
        $CostData2{$NumEntries} = $dollars2;
    }
    $NumEntries++;
}

close COMBTDB;

#######################################################
# calculate mode split

# absord constant into alpha: use log10() not ln() for income
$alphatest = $alpha / log(10);

# count entries preferring mode 1
$ChooseMode1 = 0;
for($i=0; $i<$NumEntries; $i++) {
    if( $alphatest * $CostData1{$i} + $CostData2{$i}
       < $alphatest * $CostData3{$i} + $CostData4{$i} ) {
        # mode2 cost is greater
        $ChooseMode1++;
    }
}

print $ChooseMode1," (",$ChooseMode1/$NumEntries,") choose mode ",$mode1," over mode ",$mode2," for alpha = ",$alpha, " with
",$NumEntries," valid entries\n";

#######################################################

exit;
```