Chapter Nine: Contents

(AS-7 – 15 October 2001 – LA-UR 01-5716 – Portland Study Reports)

1. SCRIPTS 1

1.1 FIXBRIDGECROSS.NS 1
1.2 FIXBRIDGECROSS.SN 7
1.3 FIXBRIDGES-AGAIN.CSH 13
1.4 MODFINDCROSSINGS.PL 14
1.5 SPLITACTBYZONE.PL.ROCK 18

Chapter Nine—AS-7

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. SCRIPTS

1.1 FixBridgeCross.NS

```
#! /bin/csh

# The following 5 parameters have no default value
# $1 is tour type, e.g. NSW, NSS, NSO, SNW, SNS, SNO, EW, WE
# $2 is target number of crossings
# $3 is initial value of c
# $4 is activity file
# $5 is number of processors to use

# The following 4 parameters are for continuation of a previous non-converged run
# $6 is previous value of c for which result was above target, or "none" if none known
# $7 is previous value of c for which result was below target, or "none" if none known
# $8 is iteration number to begin with
# $9 is iteration number to end at

setenv SCENARIO $TRANSIMS_HOME/scenarios/allstr
setenv CONFIG_FILE  $SCENARIO/config_files/allstr_actgen_river_cross.cfg
setenv LOG_FILE $SCENARIO/log/ActRegenCross$1.log
setenv ACT_LOG_FILE $SCENARIO/log/ActRegen$1.log
setenv ACT_DIR $SCENARIO/activity
setenv SCRIPT_DIR $ACT_DIR
setenv ACTIVITY_FILE $ACT_DIR/$4
setenv PARTIAL_FILE $ACT_DIR/cpar$1
setenv PROBLEM_FILE $ACT_DIR/cprob$1
setenv ITDB_FILE $ACT_DIR/itcross$1
setenv FEEDBACK_COMMANDS $ACT_DIR/fbcross$1
setenv HOUSEHOLD_FILE $ACT_DIR/hhcross$1
#setenv TMP_CONFIG /tmp/allstr.cfg
setenv TMP_CONFIG $ACT_DIR/cross$1.cfg

setenv MERGE_INDEX $TRANSIMS_HOME/bin/MergeIndices
setenv INDEXDEFRAG  $TRANSIMS_HOME/bin/IndexDefrag
setenv COLLATOR       $TRANSIMS_HOME/bin/Collator
```

```
setenv ACTREGEN       $TRANSIMS_HOME/bin/ActivityRegenerator
setenv HOUSEHOLDS $TRANSIMS_HOME/bin/MakeHouseholdFile
setenv DTOA $TRANSIMS_HOME/bin/10to26
setenv INDEX_ACT $TRANSIMS_HOME/bin/IndexActivityFile
setenv PERL /usr/bin/perl
setenv GAWK /usr/bin/gawk


@ targetlb = ($2 * 95) / 100
@ targetub = ($2 * 105) / 100
@ c = 0
set above = 0
set below = 0
set istart = 1
set iend = 1
set mxavail = `tail -100 $ACT_DIR/machines.$1`
@ mstart = 0
set length = `wc -l $ACT_DIR/machines.$1`
@ mend = $length[1]
@ muse = $5

# Check for continuation run
if ($#argv > 5) then
        echo "Continuation of previous iteration" >>&! $LOG_FILE
        @ c = $3
        if ($6 != "none") then
                set above = 1
                @ cmin = $6
        endif
        if ($7 != "none") then
                set below = 1
                @ cmax = $7
        endif
        set istart = $8
        set iend = $9
        goto loop
endif

# Regenerate the activities that crossed the river
echo "Regenerating activities to remove river crossings..." >>&! $LOG_FILE
echo "Target range for crossings is" $targetlb "to" $targetub >>&! $LOG_FILE

/bin/cp $CONFIG_FILE $TMP_CONFIG
/bin/chmod u+w $TMP_CONFIG

echo "ROUTER_HOUSEHOLD_FILE  "  $HOUSEHOLD_FILE >> $TMP_CONFIG
echo "SEL_HOUSEHOLD_FILE  "  $HOUSEHOLD_FILE >> $TMP_CONFIG
echo "ACTIVITY_FILE   "  $ACTIVITY_FILE >> $TMP_CONFIG
echo "SEL_ITDB_FILE   "  $ITDB_FILE  >> $TMP_CONFIG
echo "ACT_FEEDBACK_FILE  " $FEEDBACK_COMMANDS >> $TMP_CONFIG
echo "ACT_LOG_FILE    "  $ACT_LOG_FILE >> $TMP_CONFIG
```

```
echo "Dividing households among $muse processors" >>&! $LOG_FILE
$HOUSEHOLDS $TMP_CONFIG $muse >>&! $LOG_FILE

echo "Indexing activity file" >>&! $LOG_FILE
$INDEX_ACT $ACTIVITY_FILE >>&! $LOG_FILE
sleep 10

# Construct wordlist of machines to be used
@ m = 0
set mxused = ()
@ mused = 0
foreach mx ($mxavail)
        set result = `ping -q -c 2 $mx | grep "0 packets received"`
        if ("X$result" == "X") then
                if ($m >= $mstart && $m <= $mend && $mused < $muse) then
                        set mxused = ($mxused $mx)
                        @ mused++
                endif
                @ m++
        else
                echo "No response from $mx"
        endif
end
# Spawn processes
@ m = 0
echo "Running Collator" >>&! $LOG_FILE
#$COLLATOR $TMP_CONFIG >>&! $LOG_FILE
foreach mx ($mxused)
        set suffix = `$DTOA $m`
        echo "Collator process $m on $mx with suffix $suffix"
        echo "Collator process $m on $mx with suffix $suffix" >>&! $LOG_FILE
        ssh -n -f $mx $COLLATOR $TMP_CONFIG $m >>&! $LOG_FILE.t$suffix
        @ m++
end
# Wait for processes to finish on all cpus
set proc = Collator
sleep 10
foreach mx ($mxused)
        set numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
        while ($numJobs > 0)
                sleep 10
                @ numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
        end
end
# Concatenate outputs
@ m = 0
foreach mx ($mxused)
        set suffix = `$DTOA $m`
        if ($m == 0) then
                head -2 $ITDB_FILE.t$suffix.000.it >! $ITDB_FILE.000.it
```

```
        endif
        $GAWK 'NR>2' $ITDB_FILE.t$suffix.000.it >> $ITDB_FILE.000.it
        @ m++
end

echo "Finding river crossings in iteration database 0 " >>&! $LOG_FILE
$PERL $SCRIPT_DIR/ModFindCrossings.pl $ITDB_FILE.000.it $FEEDBACK_COMMANDS $LOG_FILE $1 1
@ cross = `tail -1 $LOG_FILE`
if ($cross > $targetub) then
        echo $cross "is too many crossings" >>&! $LOG_FILE
        set above = 1
        @ cmin = 0
        @ c = $3
else if ($cross < $targetlb) then
        echo $cross "is too few crossings" >>&! $LOG_FILE
        set below = 1
        @ cmax = 0
        @ c = $3
else
        echo $cross "is an acceptable number of crossings with c =" $c >>&! $LOG_FILE
        echo "Finished." >>& $LOG_FILE
        exit 0
endif


loop:
set done = 0
# Do iterations
set i = $istart
while ($i <= $iend)
        echo "Running Activity Regenerator for iteration" $i "with c =" $c >>&! $LOG_FILE
        echo "ACT_PARTIAL_OUTPUT " $PARTIAL_FILE.$i >> $TMP_CONFIG
        echo "ACT_PROBLEM_FILE   " $PROBLEM_FILE.$i >> $TMP_CONFIG
        echo "ACT_TRAVEL_TIME_FUNCTION_PARAMETERS  " $c >> $TMP_CONFIG
        $ACTREGEN $TMP_CONFIG >>&! $LOG_FILE

        # remove indices created by ActivityRegnerator because sometimes they are corrupted
        /bin/rm -f $PARTIAL_FILE.$i.*.idx
        echo "Indexing activity file" >>&! $LOG_FILE
        $INDEX_ACT $PARTIAL_FILE.$i >>&! $LOG_FILE
        sleep 10

        # Construct wordlist of machines to be used
        @ m = 0
        set mxused = ()
        @ mused = 0
        foreach mx ($mxavail)
                set result = `ping -q -c 2 $mx | grep "0 packets received"`
                if ("X$result" == "X") then
                        if ($m >= $mstart && $m <= $mend && $mused < $muse) then
                                set mxused = ($mxused $mx)
```

```
                                @ mused++
                        endif
                        @ m++
                else
                        echo "No response from $mx"
                endif
        end
        # Spawn processes
        # Wait for processes to finish on all cpus

        echo "ACTIVITY_FILE  " $PARTIAL_FILE.$i >> $TMP_CONFIG
        sleep 10
        echo "Running Collator" >>&! $LOG_FILE
#       $COLLATOR $TMP_CONFIG >>&! $LOG_FILE
        # Spawn processes
        @ m = 0
        foreach mx ($mxused)
                set suffix = `$DTOA $m`
                echo "Collator process $m on $mx with suffix $suffix"
                echo "Collator process $m on $mx with suffix $suffix" >>&! $LOG_FILE
                ssh -n -f $mx $COLLATOR $TMP_CONFIG $m >>&! $LOG_FILE.t$suffix
                @ m++
        end
        # Wait for processes to finish on all cpus
        set proc = Collator
        sleep 10
        foreach mx ($mxused)
                set numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
                while ($numJobs > 0)
                        sleep 10
                        @ numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
                end
        end
        # Concatenate outputs
        @ m = 0
        foreach mx ($mxused)
                set suffix = `$DTOA $m`
                if ($m == 0) then
                        head -2 $ITDB_FILE.t$suffix.00$i.it >! $ITDB_FILE.00$i.it
                endif
                $GAWK 'NR>2' $ITDB_FILE.t$suffix.00$i.it >> $ITDB_FILE.00$i.it
                @ m++
        end

        echo "Finding river crossings in iteration database" $i >>&! $LOG_FILE
        $PERL $SCRIPT_DIR/ModFindCrossings.pl $ITDB_FILE.00$i.it crossjunk$1 $LOG_FILE $1 0
        @ cross = `tail -1 $LOG_FILE`
        if ($cross > $targetub) then
                echo $cross "is too many crossings" >>&! $LOG_FILE
                set above = 1
                @ cmin = $c
```

```
                if ($below == 1) then
                        @ c = ($cmin + $cmax) / 2
                else
                        @ c = 2 * $c
                endif
        else if ($cross < $targetlb) then
                echo $cross "is too few crossings" >>&! $LOG_FILE
                set below = 1
                @ cmax = $c
                if ($above == 1) then
                        @ c = ($cmin + $cmax) / 2
                else
                        @ c = 2 * $c
                endif
        else
                echo $cross "is an acceptable number of crossings with c =" $c >>&! $LOG_FILE
                set done = $i
                break
        endif
        set i = `expr $i + 1`
end

if ($done == 0) then
        echo "Iterations completed without converging to target" >>&! $LOG_FILE
        echo "New activity file not created" >>&! $LOG_FILE
        echo " " >>&! $LOG_FILE
        exit 1
endif

# Merge the partial activity set indexes with the original activity set index.
echo "Merging partial activity indexes.. " >>& $LOG_FILE
$MERGE_INDEX $ACTIVITY_FILE.merged.hh.idx $ACTIVITY_FILE.hh.idx $PARTIAL_FILE.$done.hh.idx >>&! $LOG_FILE

# Create a merged activity file from the merged indexes
echo "Creating merged activity file.." >>& $LOG_FILE
$INDEXDEFRAG $ACTIVITY_FILE.merged.hh.idx $ACTIVITY_FILE.$1 >>&! $LOG_FILE

# Save the uniterated activity set and move the iterated
# one into the same filename which will be used by the Router.
#/bin/mv $ACTIVITY_FILE $ACTIVITY_FILE.before$1
#/bin/mv $ACTIVITY_FILE.$1 $ACTIVITY_FILE

# Clean up
#/bin/rm -f $TMP_CONFIG
#/bin/rm -f $PARTIAL_FILE.*
#/bin/rm -f $FEEDBACK_COMMANDS
#/bin/rm -f $PROBLEM_FILE.*
#/bin/rm -f $ACTIVITY_FILE.*idx
#/bin/rm -f crossjunk

echo "Finished." >>& $LOG_FILE
```

1.2 FixBridgeCross.SN

```
#! /bin/csh

# The following 5 parameters have no default value
# $1 is tour type, e.g. NSW, NSS, NSO, SNW, SNS, SNO, EW, WE
# $2 is target number of crossings
# $3 is initial value of c
# $4 is activity file
# $5 is number of processors to use

# The following 4 parameters are for continuation of a previous non-converged run
# $6 is previous value of c for which result was above target, or "none" if none known
# $7 is previous value of c for which result was below target, or "none" if none known
# $8 is iteration number to begin with
# $9 is iteration number to end at

setenv SCENARIO $TRANSIMS_HOME/scenarios/allstr
setenv CONFIG_FILE  $SCENARIO/config_files/allstr_actgen_river_cross.cfg
setenv LOG_FILE $SCENARIO/log/ActRegenCross$1.log
setenv ACT_LOG_FILE $SCENARIO/log/ActRegen$1.log
setenv ACT_DIR $SCENARIO/activity
setenv SCRIPT_DIR $ACT_DIR
setenv ACTIVITY_FILE $ACT_DIR/$4
setenv PARTIAL_FILE $ACT_DIR/cpar$1
setenv PROBLEM_FILE $ACT_DIR/cprob$1
setenv ITDB_FILE $ACT_DIR/itcross$1
setenv FEEDBACK_COMMANDS $ACT_DIR/fbcross$1
setenv HOUSEHOLD_FILE $ACT_DIR/hhcross$1
#setenv TMP_CONFIG /tmp/allstr.cfg
setenv TMP_CONFIG $ACT_DIR/cross$1.cfg

setenv MERGE_INDEX $TRANSIMS_HOME/bin/MergeIndices
setenv INDEXDEFRAG  $TRANSIMS_HOME/bin/IndexDefrag
setenv COLLATOR        $TRANSIMS_HOME/bin/Collator
setenv ACTREGEN        $TRANSIMS_HOME/bin/ActivityRegenerator
setenv HOUSEHOLDS $TRANSIMS_HOME/bin/MakeHouseholdFile
setenv DTOA $TRANSIMS_HOME/bin/10to26
setenv INDEX_ACT $TRANSIMS_HOME/bin/IndexActivityFile
setenv PERL /usr/bin/perl
setenv GAWK /usr/bin/gawk


@ targetlb = ($2 * 95) / 100
@ targetub = ($2 * 105) / 100
@ c = 0
set above = 0
```

```
set below = 0
set istart = 1
set iend = 1
set mxavail = `tail -100 $ACT_DIR/machines.$1`
@ mstart = 0
set length = `wc -l $ACT_DIR/machines.$1`
@ mend = $length[1]
@ muse = $5

# Check for continuation run
if ($#argv > 5) then
        echo "Continuation of previous iteration" >>&! $LOG_FILE
        @ c = $3
        if ($6 != "none") then
                set above = 1
                @ cmin = $6
        endif
        if ($7 != "none") then
                set below = 1
                @ cmax = $7
        endif
        set istart = $8
        set iend = $9
        goto loop
endif

# Regenerate the activities that crossed the river
echo "Regenerating activities to remove river crossings..." >>&! $LOG_FILE
echo "Target range for crossings is" $targetlb "to" $targetub >>&! $LOG_FILE

/bin/cp $CONFIG_FILE $TMP_CONFIG
/bin/chmod u+w $TMP_CONFIG

echo "ROUTER_HOUSEHOLD_FILE  "  $HOUSEHOLD_FILE >> $TMP_CONFIG
echo "SEL_HOUSEHOLD_FILE  "  $HOUSEHOLD_FILE >> $TMP_CONFIG
echo "ACTIVITY_FILE   "  $ACTIVITY_FILE >> $TMP_CONFIG
echo "SEL_ITDB_FILE   "  $ITDB_FILE  >> $TMP_CONFIG
echo "ACT_FEEDBACK_FILE  " $FEEDBACK_COMMANDS >> $TMP_CONFIG
echo "ACT_LOG_FILE    "  $ACT_LOG_FILE >> $TMP_CONFIG

echo "Dividing households among $muse processors" >>&! $LOG_FILE
$HOUSEHOLDS $TMP_CONFIG $muse >>&! $LOG_FILE

echo "Indexing activity file" >>&! $LOG_FILE
$INDEX_ACT $ACTIVITY_FILE >>&! $LOG_FILE
sleep 10

# Construct wordlist of machines to be used
@ m = 0
set mxused = ()
@ mused = 0
```

```
foreach mx ($mxavail)
        set result = `ping -q -c 2 $mx | grep "0 packets received"`
        if ("X$result" == "X") then
                if ($m >= $mstart && $m <= $mend && $mused < $muse) then
                        set mxused = ($mxused $mx)
                        @ mused++
                endif
                @ m++
        else
                echo "No response from $mx"
        endif
end
# Spawn processes
@ m = 0
echo "Running Collator" >>&! $LOG_FILE
#$COLLATOR $TMP_CONFIG >>&! $LOG_FILE
foreach mx ($mxused)
        set suffix = `$DTOA $m`
        echo "Collator process $m on $mx with suffix $suffix"
        echo "Collator process $m on $mx with suffix $suffix" >>&! $LOG_FILE
        ssh -n -f $mx $COLLATOR $TMP_CONFIG $m >>&! $LOG_FILE.t$suffix
        @ m++
end
# Wait for processes to finish on all cpus
set proc = Collator
sleep 10
foreach mx ($mxused)
        set numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
        while ($numJobs > 0)
                sleep 10
                @ numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
        end
end
# Concatenate outputs
@ m = 0
foreach mx ($mxused)
        set suffix = `$DTOA $m`
        if ($m == 0) then
                head -2 $ITDB_FILE.t$suffix.000.it >! $ITDB_FILE.000.it
        endif
        $GAWK 'NR>2' $ITDB_FILE.t$suffix.000.it >> $ITDB_FILE.000.it
        @ m++
end

echo "Finding river crossings in iteration database 0 " >>&! $LOG_FILE
$PERL $SCRIPT_DIR/ModFindCrossings.pl $ITDB_FILE.000.it $FEEDBACK_COMMANDS $LOG_FILE $1 1
@ cross = `tail -1 $LOG_FILE`
if ($cross > $targetub) then
        echo $cross "is too many crossings" >>&! $LOG_FILE
        set above = 1
        @ cmin = 0
```

```
        @ c = $3
else if ($cross < $targetlb) then
        echo $cross "is too few crossings" >>&! $LOG_FILE
        set below = 1
        @ cmax = 0
        @ c = $3
else
        echo $cross "is an acceptable number of crossings with c =" $c >>&! $LOG_FILE
        echo "Finished." >>& $LOG_FILE
        exit 0
endif


loop:
set done = 0
# Do iterations
set i = $istart
while ($i <= $iend)
        echo "Running Activity Regenerator for iteration" $i "with c =" $c >>&! $LOG_FILE
        echo "ACT_PARTIAL_OUTPUT " $PARTIAL_FILE.$i >> $TMP_CONFIG
        echo "ACT_PROBLEM_FILE   " $PROBLEM_FILE.$i >> $TMP_CONFIG
        echo "ACT_TRAVEL_TIME_FUNCTION_PARAMETERS  " $c >> $TMP_CONFIG
        $ACTREGEN $TMP_CONFIG >>&! $LOG_FILE

        # remove indices created by ActivityRegnerator because sometimes they are corrupted
        /bin/rm -f $PARTIAL_FILE.$i.*.idx
        echo "Indexing activity file" >>&! $LOG_FILE
        $INDEX_ACT $PARTIAL_FILE.$i >>&! $LOG_FILE
        sleep 10

        # Construct wordlist of machines to be used
        @ m = 0
        set mxused = ()
        @ mused = 0
        foreach mx ($mxavail)
                set result = `ping -q -c 2 $mx | grep "0 packets received"`
                if ("X$result" == "X") then
                        if ($m >= $mstart && $m <= $mend && $mused < $muse) then
                                set mxused = ($mxused $mx)
                                @ mused++
                        endif
                        @ m++
                else
                        echo "No response from $mx"
                endif
        end
        # Spawn processes
        # Wait for processes to finish on all cpus

        echo "ACTIVITY_FILE  " $PARTIAL_FILE.$i >> $TMP_CONFIG
        sleep 10
```

```
        echo "Running Collator" >>&! $LOG_FILE
#       $COLLATOR $TMP_CONFIG >>&! $LOG_FILE
        # Spawn processes
        @ m = 0
        foreach mx ($mxused)
                set suffix = `$DTOA $m`
                echo "Collator process $m on $mx with suffix $suffix"
                echo "Collator process $m on $mx with suffix $suffix" >>&! $LOG_FILE
                ssh -n -f $mx $COLLATOR $TMP_CONFIG $m >>&! $LOG_FILE.t$suffix
                @ m++
        end
        # Wait for processes to finish on all cpus
        set proc = Collator
        sleep 10
        foreach mx ($mxused)
                set numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
                while ($numJobs > 0)
                        sleep 10
                        @ numJobs = `ssh -n $mx ps -C $proc | grep $proc | wc -l`
                end
        end
        # Concatenate outputs
        @ m = 0
        foreach mx ($mxused)
                set suffix = `$DTOA $m`
                if ($m == 0) then
                        head -2 $ITDB_FILE.t$suffix.00$i.it >! $ITDB_FILE.00$i.it
                endif
                $GAWK 'NR>2' $ITDB_FILE.t$suffix.00$i.it >> $ITDB_FILE.00$i.it
                @ m++
        end

        echo "Finding river crossings in iteration database" $i >>&! $LOG_FILE
        $PERL $SCRIPT_DIR/ModFindCrossings.pl $ITDB_FILE.00$i.it crossjunk$1 $LOG_FILE $1 0
        @ cross = `tail -1 $LOG_FILE`
        if ($cross > $targetub) then
                echo $cross "is too many crossings" >>&! $LOG_FILE
                set above = 1
                @ cmin = $c
                if ($below == 1) then
                        @ c = ($cmin + $cmax) / 2
                else
                        @ c = 2 * $c
                endif
        else if ($cross < $targetlb) then
                echo $cross "is too few crossings" >>&! $LOG_FILE
                set below = 1
                @ cmax = $c
                if ($above == 1) then
                        @ c = ($cmin + $cmax) / 2
                else
```

```
                        @ c = 2 * $c
                endif
        else
                echo $cross "is an acceptable number of crossings with c =" $c >>&! $LOG_FILE
                set done = $i
                break
        endif
        set i = `expr $i + 1`
end

if ($done == 0) then
        echo "Iterations completed without converging to target" >>&! $LOG_FILE
        echo "New activity file not created" >>&! $LOG_FILE
        echo " " >>&! $LOG_FILE
        exit 1
endif

# Merge the partial activity set indexes with the original activity set index.
#echo "Merging partial activity indexes.. " >>& $LOG_FILE
#$MERGE_INDEX $ACTIVITY_FILE.merged.hh.idx $ACTIVITY_FILE.hh.idx $PARTIAL_FILE.$done.hh.idx >>&! $LOG_FILE

# Create a merged activity file from the merged indexes
#echo "Creating merged activity file.." >>& $LOG_FILE
#$INDEXDEFRAG $ACTIVITY_FILE.merged.hh.idx $ACTIVITY_FILE.$1 >>&! $LOG_FILE

# Save the uniterated activity set and move the iterated
# one into the same filename which will be used by the Router.
#/bin/mv $ACTIVITY_FILE $ACTIVITY_FILE.before$1
#/bin/mv $ACTIVITY_FILE.$1 $ACTIVITY_FILE

# Clean up
#/bin/rm -f $TMP_CONFIG
#/bin/rm -f $PARTIAL_FILE.*
#/bin/rm -f $FEEDBACK_COMMANDS
#/bin/rm -f $PROBLEM_FILE.*
#/bin/rm -f $ACTIVITY_FILE.*idx
#/bin/rm -f crossjunk

echo "Finished." >>& $LOG_FILE
```

1.3 fixbridges-again.csh

```
#! /usr/local/bin/tcsh -f

cd /home/jpsmith/TRANSIMS/Feedback/allstr/AS3.25/bridges

# split activities by river zone
perl /home/Gershwinoutput1/kpb/allstr-act/bcfb/SplitActByZone.pl.rock
/home/projects/transims/config/integration/scenarios/allstr/network/Activity_Location.tbl
/home/jpsmith/TRANSIMS/Feedback/allstr/AS3.25/fixwalks/AS-6 activ.N_share activ.N_no-sh activ.W_share activ.W_no-sh activ.E_share
activ.E_no-sh

# merge to form NS and SN activities
cat activ.N_share activ.N_no-sh >! activ.NS
cat activ.W_share activ.W_no-sh activ.E_share activ.E_no-sh >! activ.SN

# count bridge crossings
FixBridgeCross.NS NS $NUM 0 actfile numprocs
FixBridgeCross.NS NS $NUM 0 actfile numprocs none none $ITER $NEWNUM
```

```
#uses: /home/Gershwinoutput1/kpb/allstr-act/bcfb/ModFindCrossings.pl

It should be used by the FixBridgeCross script described below.
(Note that FixBridgeCross refers to FindCrossings.pl, so you should
rename the Find script or modify the Fix script to use the modified one.)
```

1.4 ModFindCrossings.pl

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

# find river crossings in itdb file
# ARGV[3] is the tour type: NS, SN, EW, WE, ALL
# ARGV[4] indicates whether to save tours in feedback file

use English;

open IN, "<$ARGV[0]" or die "Couldn't open $ARGV[0] for input ($OS_ERROR)\n";
open OUT, ">$ARGV[1]" or die "Couldn't open $ARGV[1] for output ($OS_ERROR)\n";
open LOG, ">>$ARGV[2]" or die "Couldn't open $ARGV[2] for output ($OS_ERROR)\n";
print LOG "tour type is $ARGV[3]\n";

%counts = ();
$counts{"ALL"} = 0;
$counts{"NSW"} = 0;
$counts{"NSS"} = 0;
$counts{"NSO"} = 0;
$counts{"SNW"} = 0;
$counts{"SNS"} = 0;
$counts{"SNO"} = 0;
$counts{"EWW"} = 0;
$counts{"EWS"} = 0;
$counts{"EWO"} = 0;
$counts{"WEW"} = 0;
$counts{"WES"} = 0;
$counts{"WEO"} = 0;
$total = 0;
$tourid = -1;
$first = 1;
# skip header lines
$line = <IN>;
$line = <IN>;
while (<IN>)
{
        chomp;
#     0      1      2       3        4     5     6      7       8       9      10      11           12
#       ($hh, $trav, $tour, $subtour, $trip, $sid, $eid, $etype, $emode, $eloc, $eother, $drivespass, $ereg)
        @line = split /,/;
        if ($line[2] == $tourid)      # same tour
        {
                push @tour, [ @line ];
                next;
        }
```

```
        if (!$first)   # analyze complete tour
        {
                AnalyzeTour();
                $first = 1;
        }

        # begin new tour
        if ($line[2] == 0)            # home activity
        {
                $home = $line[9];
                $hreg = $line[12];
                $tourid = -1;
        }
        else                                  # next tour
        {
                $tourid = $line[2];
                @tour = ();
                push @tour, [ @line ];
                $first = 0;
        }
}
if (!$first) { AnalyzeTour(); }

$counts{"NS"} = $counts{"NSW"} + $counts{"NSS"} + $counts{"NSO"};
$counts{"SN"} = $counts{"SNW"} + $counts{"SNS"} + $counts{"SNO"};
$counts{"WE"} = $counts{"WEW"} + $counts{"WES"} + $counts{"WEO"};
$counts{"EW"} = $counts{"EWW"} + $counts{"EWS"} + $counts{"EWO"};

if ($ARGV[3] eq 'ALL' || $ARGV[4])
{
        print LOG qq!N-S total tours = $counts{"NS"}\twork tours = $counts{"NSW"}\tshop tours = $counts{"NSS"}\tother tours =
$counts{"NSO"}\n!;
        print LOG qq!S-N total tours = $counts{"SN"}\twork tours = $counts{"SNW"}\tshop tours = $counts{"SNS"}\tother tours =
$counts{"SNO"}\n!;
        print LOG qq!W-E total tours = $counts{"WE"}\twork tours = $counts{"WEW"}\tshop tours = $counts{"WES"}\tother tours =
$counts{"WEO"}\n!;
        print LOG qq!E-W total tours = $counts{"EW"}\twork tours = $counts{"EWW"}\tshop tours = $counts{"EWS"}\tother tours =
$counts{"EWO"}\n!;
        print LOG "Total tours = $total\n";
}
elsif ($ARGV[3] eq 'NS') { print LOG qq!N-S tours = $counts{"NS"}\n!; }
elsif ($ARGV[3] eq 'SN') { print LOG qq!S-N tours = $counts{"SN"}\n!; }
elsif ($ARGV[3] eq 'WE') { print LOG qq!W-E tours = $counts{"WE"}\n!; }
elsif ($ARGV[3] eq 'EW') { print LOG qq!E-W tours = $counts{"EW"}\n!; }
for $key (keys %counts)
{
        if ($key eq $ARGV[3]) { $tmp = $counts{$key}; }
}
if ($ARGV[3] ne 'ALL') { print LOG "$tmp\n"; }
```

```
sub AnalyzeTour
{
	$total++;
	$hhid = $tour[0][0];
	$sactid = $tour[0][5];
	$work = $shop = $shared = $car = $drivespass = 0;
	$crossns = $crosssn = $crossew = $crosswe = 0;
	for ($i=0; $i < $#tour; $i++)
	{
		if ($tour[$i][9] eq $home) { next; }
		if ($tour[$i][10] > 0) { $shared = 1; }
		if ($tour[$i][11] eq "true") { $drivespass = 1; }
		if ($tour[$i][7] eq 1) { $work = 1; }
		if ($tour[$i][7] eq 8) { $work = 1; }
		if ($tour[$i][7] eq 2) { $shop = 1; }
		if ($tour[$i][8] eq 2) { $car = 1; }
		if ($hreg eq 1 && $tour[$i][12] ne 1) { $crossns = 1; }
		if ($hreg ne 1 && $tour[$i][12] eq 1) { $crosssn = 1; }
		if ($hreg eq 2 && $tour[$i][12] eq 3) { $crosswe = 1; }
		if ($hreg eq 3 && $tour[$i][12] eq 2) { $crossew = 1; }
	}
	if ($work && $shop) { $shop = 0; }
	if ($crossns && $car)
	{
		if ($work) { $counts{"NSW"}++; }
		elsif ($shop) { $counts{"NSS"}++; }
		elsif (!$work && !$shop) { $counts{"NSO"}++; }
	}
	elsif ($crosssn && $car)
	{
		if ($work) { $counts{"SNW"}++; }
		elsif ($shop) { $counts{"SNS"}++; }
		elsif (!$work && !$shop) { $counts{"SNO"}++; }
	}
	elsif ($crosswe && $car)
	{
		if ($work) { $counts{"WEW"}++; }
		elsif ($shop) { $counts{"WES"}++; }
		elsif (!$work && !$shop) { $counts{"WEO"}++; }
	}
	elsif ($crossew && $car)
	{
		if ($work) { $counts{"EWW"}++; }
		elsif ($shop) { $counts{"EWS"}++; }
		elsif (!$work && !$shop) { $counts{"EWO"}++; }
	}
	if ($ARGV[4])
	{
		if ($ARGV[3] eq "NS" && $hreg eq 1 && (!$shared || $drivespass))
			{ print OUT "$hhid $sactid LTR $sactid\n"; }
		elsif ($ARGV[3] eq "SN" && $hreg ne 1 && (!$shared || $drivespass))
```

```
                        { print OUT "$hhid $sactid LTR $sactid\n"; }
                elsif ($ARGV[3] eq "WE" && $hreg eq 2 && (!$shared || $drivespass))
                        { print OUT "$hhid $sactid LTR $sactid\n"; }
                elsif ($ARGV[3] eq "EW" && $hreg eq 3 && (!$shared || $drivespass))
                        { print OUT "$hhid $sactid LTR $sactid\n"; }
        }
}
```

1.5 SplitActByZone.pl.rock

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

# split activities according to river zone household is located in and by shared rides
# $ARGV[0] is activity location table
# $ARGV[1] is activity file
# $ARGV[2] is activities for households with home in north having shared rides
# $ARGV[3] is activities for households with home in north having no shared rides
# $ARGV[4] is activities for households with home in west having shared rides
# $ARGV[5] is activities for households with home in west having no shared rides
# $ARGV[6] is activities for households with home in east having shared rides
# $ARGV[7] is activities for households with home in east having no shared rides

use English;


# read activity location file and store river zone for each location
open IN, "<$ARGV[0]" or die "Couldn't open $ARGV[0] for input ($OS_ERROR)\n";
$line = <IN>;
while (<IN>)
{
	chomp;
	@line = split;
	$id = $line[0];
	$rzones{$id} = $line[24];
}
close IN;
$nrz = keys %rzones;
print "size of activity location array is $nrz\n";

$hhid = -1;
$first = 1;
open IN, "<$ARGV[1]" or die "Couldn't open $ARGV[1] for input ($OS_ERROR)\n";
open NS, ">$ARGV[2]" or die "Couldn't open $ARGV[2] for output ($OS_ERROR)\n";
open NN, ">$ARGV[3]" or die "Couldn't open $ARGV[3] for output ($OS_ERROR)\n";
open WS, ">$ARGV[4]" or die "Couldn't open $ARGV[4] for output ($OS_ERROR)\n";
open WN, ">$ARGV[5]" or die "Couldn't open $ARGV[5] for output ($OS_ERROR)\n";
open ES, ">$ARGV[6]" or die "Couldn't open $ARGV[6] for output ($OS_ERROR)\n";
open EN, ">$ARGV[7]" or die "Couldn't open $ARGV[7] for output ($OS_ERROR)\n";
open JUNK, ">junksplit" or die "Couldn't open junksplit for output ($OS_ERROR)\n";
while (<IN>)
{
	chomp;
	@line = split;
	if ($line[0] == $hhid)	# same household
	{
```

```
                push @hh, [ @line ];
                next;
        }

        if (!$first)   # analyze complete household
        {
                AnalyzeHousehold();
        }

        # begin new household
        $hhid = $line[0];
        @hh = ();
        push @hh, [ @line ];
        $first = 0;
}
if (!$first) { AnalyzeHousehold(); }


sub AnalyzeHousehold
{
        $hreg = $rzones{$hh[0][20]};
        $shared = 0;
        for ($i=0; $i < $#hh; $i++)
        {
                if ($hh[$i][21] > 0) { $shared = 1; }
        }
        if ($hreg == 1 && $shared) { $FH = *NS; }
        elsif ($hreg == 1 && !$shared) { $FH = *NN; }
        elsif ($hreg == 2 && $shared) { $FH = *WS; }
        elsif ($hreg == 2 && !$shared) { $FH = *WN; }
        elsif ($hreg == 3 && $shared) { $FH = *ES; }
        elsif ($hreg == 3 && !$shared) { $FH = *EN; }
        else
        {
                print "household $hhid in region $hreg with $shared shared rides\n";
#               return;
                $FH = *JUNK;
        }
        for $i (0 .. $#hh)
        {
                $aref = $hh[$i];
                for $j (0 .. $#{$aref})
                {
                        print $FH "$aref->[$j]\t";
                }
                print $FH "\n" or die "Can't write $hhid to $FH ($OS_ERROR)\n";
        }
}
```