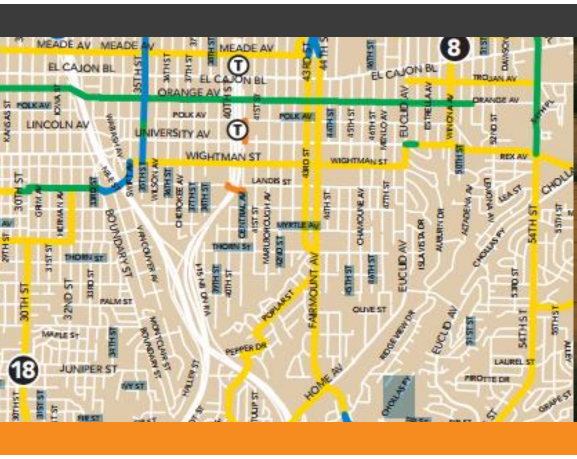
ABM AT ENHANCEMENTS

1.30.2015



PREPARED FOR:
SANDAG

SUBMITTED BY:
RSG

IN COOPERATION WITH:
HOOD TRANSPORTATION CONSULTING
PARSONS BRINCKERHOFF

600 B Street, Suite 2202
San Diego, CA 92101
619.375.0716
www.rsginc.com

ck here to enter a date.

PREPARED FOR:
SANDAG

CONTENTS

1.0 INTRODUCTION

The purpose of this document is to describe the active transportation enhancements made to SANDAG's activity-based model system. These enhancements were necessary in order to make the model system sensitive to bicycle infrastructure improvements such as the implementation of bicycle paths, lanes and routes, as well as "cycle tracks" and "bicycle boulevards." In addition, the model system was also enhanced to provide more accurate "all streets" based measures of pedestrian accessibilities.

The document first provides an overview of SANDAG's activity-based model system, and identifies where active transportation-related changes were required. A key focus of this discussion is on the process for generating multiple bicycle path alternatives. Software development and revision were key components of this effort, and the document provides detailed descriptions of the overall software architecture framework and processing flow, new software components, and modifications to existing activity-based model components.

Establishing a new active transportation sensitive model input networks was essential to making the models sensitive to active transportation investments, and a significant amount of effort was devoted to building and testing more detailed input AT networks. The document describes the inputs to this process, as well as the methods used to build and update active transportation networks.

Once the new active transportation model components and input data were integrated into SANDAG's activity-based model, it was necessary to re-calibrate and re-validate the model system. A key focus of the effort was on calibrating and modifying the tour mode choice models, although calibration, validation and testing of the bicycle route choice model was also performed. The AT-enhanced ABM was also subjected to a set of sensitivity tests to evaluate how it responded to active transportation improvements.

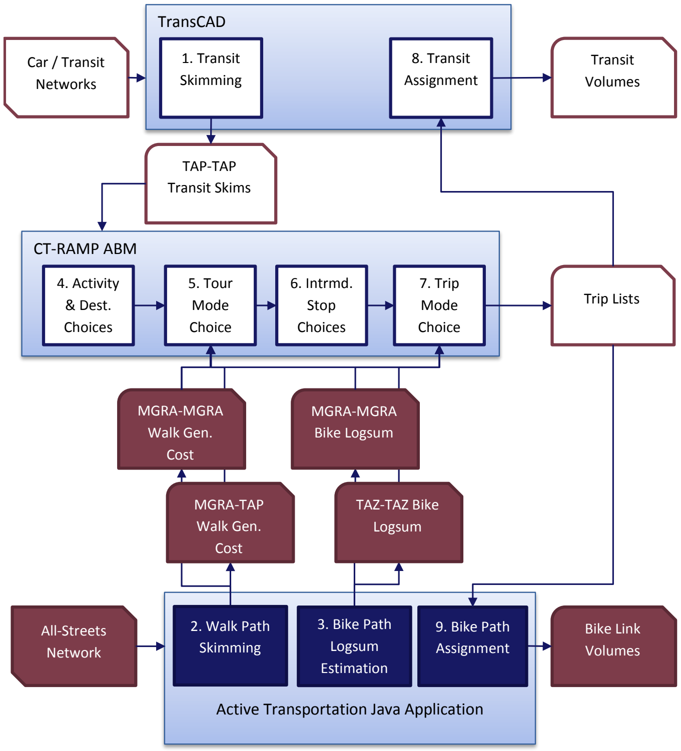
Finally, this document provides detailed information on the configuration and running of the new active transportation enhanced activity-based model

2.0 MODEL DESIGN AND IMPLEMENTATION

2.1 | OVERVIEW

A diagram of SANDAG's activity-based model (ABM) with active transportation enhancements appears in Figure 2.1. The model is split into three primary components, depending on the software running the calculations. Transit skimming and network calculations are performed in TransCAD. These components remain unchanged from the original ABM. The estimation of demand for tours and trips by persons and households is performed by the CT-RAMP activity-based modeling software in Java. Modifications to this model have been made in the Tour Mode Choice and Trip Mode Choice models. The estimation of walk and bike levels-of-service and assignment of bike trips to network links is performed by the new active transportation java application. The components of these instructions, inputs, and outputs of these calculations appear shaded with a solid color in the diagram to highlight what components are new to the version of the ABM enhanced with active transportation capabilities. The following sections describe the numbered components in the diagram, and modifications that were made to them, in the order in which the calculations are performed.

FIGURE 2.1 ABM SYSTEM ACTIVE TRANSPORTATION ENHANCEMENTS



TRANSIT SKIMMING

This component calculates matrices describing the level-of-service such as in-vehicle time and wait time—or “skims”—at and on transit vehicles between transit access points (TAPs), which are geographic points representing bus and rail stops, or clusters of stops, in the region. It outputs a TAP-TAP matrix of these skims in TransCAD format that are used by several components of CT-RAMP. No changes were made to transit skimming for the active transportation enhancements.

WALK PATH SKIMMING

This component calculates matrices describing the level-of-service by walking between MGRAs and between MGRAs and TAPs. The matrices contain a perceived time—or generalized cost—and actual time, are output in .csv format, and are used by CT-RAMP both for estimating demand and measuring performance metrics such as actual time spent walking. These skims, as well as the new bike skims, are computed using a new “all-streets” network that contains more geographic detail than the existing highway network, and additional attributes such as the type of facility, and the elevation gain. The generalized cost of walking between nodes is calculated using Dijkstra’s algorithm, where the minimum path cost is found by summing the costs of individual links using the cost function in Table 2.1. The MGRA-TAP perceived time is combined in CT-RAMP with the TAP-TAP skims from the transit skims to determine the cost of entire origin-destination (OD) movements by the transit modes.

TABLE 2.1 WALK GENERALIZED COST FUNCTION PARAMETERS

Variable	Coef.
Distance (arc length), miles	20.000
Elevation gain, ignoring declines, feet	0.067

BIKE PATH LOGSUM ESTIMATION

This component calculates matrices describing the level-of-services by bicycling between MGRAs and TAZs. The matrices contain two primary measures: 1) a logsum corresponding to the expected maximum utility from a multinomial logit choice over multiple route alternatives, and 2) the actual time spent cycling. The algorithm used to estimate the logsums is described in 1.2.3. The logums are used by CT-RAMP in the tour mode choice and trip mode choice models to represent the impedance of cycling between origins and destinations. To save computation time, the logsums are estimated between MGRAs for tours of two miles or less, and between TAZs for tours of up to twenty miles. For a given OD pair, CT-RAMP first checks the MGRA matrix for a value. If no value is found, it then checks the TAZ matrix. If no value is found in either matrix, cycling is set to unavailable in the mode choice models.

ACTIVITY & DESTINATION CHOICES

This component predicts the long term choices of residents, such as work location and auto ownership, and the departure time and destinations of travel tours—sequences of trips beginning and ending at home or work. No changes were made to these models, although their results will be slightly different with the active transportation enhancements because the values of the tour mode choice logsum, which represents the impedance of travel by all modes in these models, will be different with the new walk and bike impedances.

TOUR MODE CHOICE

This component predicts the primary mode used to travel on tours. The utility function of the model was changed with the active transportation enhancements to account for differences between the original walk and bike impedances and the enhanced impedances. First, the walk and bike impedance coefficients were adjusted. Second, variables with asserted coefficients describing the density and mixture of land uses were removed from the bike utility. Third, a variable increasing the likelihood of bike for tours near the coast was introduced. Finally, the alternative-specific constants (ASCs) of all modes were adjusted by tour purpose to match adjusted modal distribution targets from the household survey and bicycle traffic counts after applying the bicycle assignment model. The details of this calibration are described in Section 5 of this document.

INTERMEDIATE STOP CHOICES

This component predicts the locations, times, and purposes of intermediate stops made on tours. Minor changes were made to the software running this model to accommodate the new walk and bike skims by correctly setting the availability of alternatives in the stop location choice model. The distribution of locations will also differ slightly for walk and bike tours between the original and new ABM because the value of the trip mode choice logsum will differ for these tour modes.

TRIP MODE CHOICES

This component predicts the mode of individual trips on a tour, given the primary mode of the tour. The utility function was modified to accommodate the active-transportation enhancements with coefficients that are consistent with changes made to the tour mode choice model.

TRANSIT ASSIGNMENT

After lists of trips are output by CT-RAMP in .csv format, this component assigns transit trips to bus and rail routes and links in TransCAD. No changes were made to this component with the active transportation enhancements.

BIKE PATH ASSIGNMENT

Similarly to the transit assignment , this component assigns bicycle trips to links in the all-streets network. The form of the model is similar to the choice model in the bike path logsum estimation (1.1.3), but with two differences. First, instead of outputting the logsum, the component uses the probabilities of individual route alternatives to proportion estimated numbers of trips to the network links that make up the routes. Second, these probabilities are calculated for individual disaggregate bike trips rather than for all OD pairs in the region.

2.2 | DETAILS OF BIKE PATH CHOICE CALCULATIONS

The bike path choice model is a multinomial logit discrete choice model that estimates the probabilities of an individual's choosing among multiple alternative routes between a given origin and destination. This discrete choice model forms the basis of both the estimation of the level of service between OD pairs used by the demand models and the estimation of the number of cyclists assigned to network links. The level of service is the expected maximum utility, or logsum, from the model, and network link assignments are made with individual route probabilities.

UTILITY FUNCTION

The utility function variables and parameters appear in Table 2.2. The utility accounts for the distance on different types of facilities, the gain in elevation turns, signal delay, and navigating un-signalized intersections with high-volume facilities. To account for the correlation in the random utilities of overlapping routes, the utility function in the model includes a "path size" measure, described in the following section. Coefficients were transferred from route choice model estimated from GPS data in Monterey, California; Portland, Oregon; and San Francisco, California by scaling multiples of the total distance coefficient. The path assignment utility function also segments the coefficient for the gain in elevation by the tour purpose and gender of the cyclist.

PATH SIZE

The size of path i in alternative list n is calculated using the formula:

$$PS_{in} = \sum_{a \in \Gamma_i} \frac{l_a}{L_i} \frac{1}{M_{an}}$$

where Γi is the set of links in path i, la and Li are the length of link a and path i, and Man is the number of paths in the path alternative set n that contain link a. Its use derives from the theory of aggregate and elemental alternatives where a link is an aggregation of all paths that use the link. If multiple routes overlap, their "size" is less than one. If two routes overlap completely, their size will be one-half.

PATH ALTERNATIVE LIST GENERATION

Route choice modeling requires, for each origin-destination pair, the identification of a set of alternative routes. In large networks, the universal choice set is typically of unknown size, and candidates must be extracted from the network. The path alternative lists in the active transportation enhancements are generated in two phases: path sampling, and path resampling.

TABLE 2.2 BICYCLE PATH CHOICE UTILITY FUNCTION PARAMETERS

Variable	Coef.	Source
Distance, total (mi.)	–0.858	Monterey[1]
Distance on class I bike paths	0.610	Portland[2]
Distance on class II bike lanes	0.314	Monterey
Distance on class III bike routes	0.085	Monterey
Distance on arterials without bike lanes	–1.050	Monterey

[1] Hood, J., Erhardt, G., Frazier, C., Schenk, A. (2014). "Developing a Stand-Alone Bicycle Facility Emission Reduction Benefit Estimator: Incremental Nested Logit Analysis of Bicycle Trips in California's Monterey Bay Area," presented at the 93rd annual meeting of the Transporation Research Board, Washington, D.C. Jan. 12-16, 2014. http://docs.trb.org/prp/14-0965.pdf (4/13/14)

[2] Broach, J., Gliebe, J., Dill, J. (2011). "Bicycle route choice model developed using revealed preference GPS data,"presented at the 90th annual meeting of the Transporation Research Board, Washington, D.C. Jan. 23-27, 2011. http://otrec.us/main/document.php?doc_id=858 (4/13/14)

Distance on “cycle track” class II bike lanes	0.120	None
Distance on “boulevard” class III bike routes	0.430	Portland
Distance wrong way	–3.445	San Francisco[3]
Cumulative gain in elevation, ignoring declines (ft.)	–0.010	San Francisco
Turns, total	–0.083	Portland
Traffic signals, excluding right turns and through T junctions	–0.040	Portland
Un-signalized left turns from principal arterial	–0.360	Portland
Un-signalized left turns from minor arterial	–0.150	Portland
Un-signalized crossings of / left turns onto principal arterial	–0.480	Portland
Un-signalized crossings of / left turns onto minor arterial	–0.100	Portland
Access of interstate, freeway, or expressway	–999.999	Constrained
Log of path size	1.000	Constrained

Path Sampling

Given an origin, paths are sampled to all relevant destinations in the network by repeatedly applying Dijstra’s algorithm to search for the paths that minimize a stochastic link generalized cost with a mean given by the additive inverse of the path utility. For each path sampling iteration, a random coefficient vector is first sampled from a non-negative multivariate uniform distribution with zero covariance and mean equal to the link generalized cost coefficients corresponding to the path choice utility function. As the path search extends outward from the origin, the random cost coefficients do not vary over links, but only over iterations of path sampling. In the path search, the cost of each subsequent link is calculated by summing the product of the random coefficients with their respective link attributes, and then multiplying the result by a non-negative discrete random edge cost multiplier . Path sampling is repeated until both a minimum count of paths and a preset target for the total of all path sizes in each alternative list is reached. If the total path size does not reach its target after a given maximum number of sampling iterations, sampling terminates to prevent excessively long computation time. Maps of the paths in example alternative lists generated using this procedure appears in Figure 2.2 through Figure 2.5. The numbers adjacent to each link show which paths in the sample use the links. In Figure 2.2, the maximum sample count has been reached without achieving the target choice set size. In Figure 2.4, the target choice set size is reached only after taking several samples more than the minimum sample count. In Figure 2.3 and Figure 2.5, the target choice set size is

[3] Hood, J., Sall, E., Charlton, B. (2011). “A GPS-based bicycle route choice model for San Francisco, California.” *Transportation Letters: The International Journal of Transportation Research*, Vol. 3, pp. 63-75. http://www.sfcta.org/sites/default/files/content/IT/CycleTracks/BikeRouteChoiceModel.pdf (4/13/14)

exceeded by the time the minimum sample count is reached. These choice sets will be resampled to constrain the choice set size, as described in the next section.

FIGURE 2.2 EXAMPLE PATHS SAMPLED FROM MGRA 100 TO MGRA 50

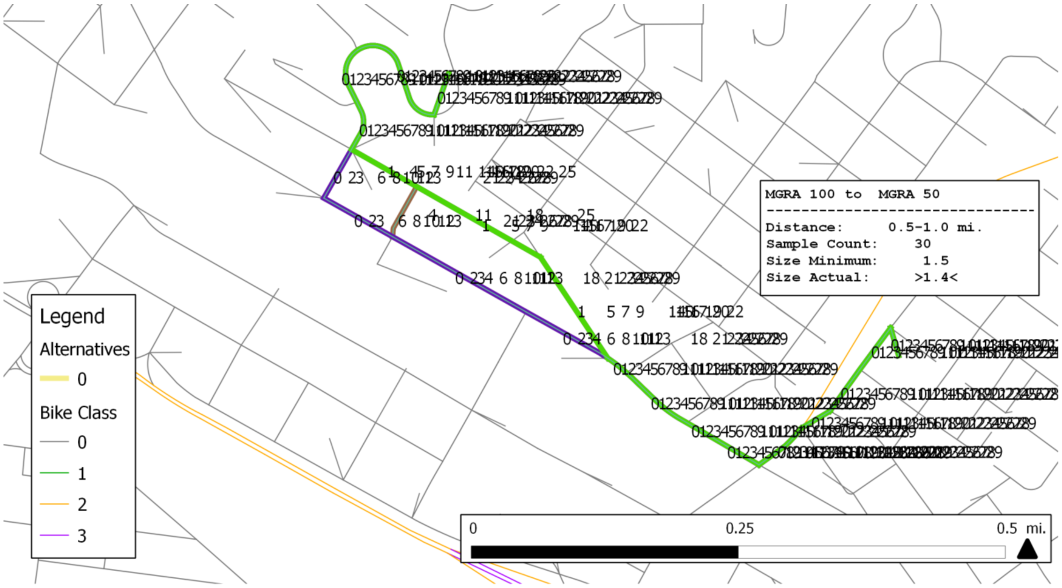


FIGURE 2.3 EXAMPLE PATHS SAMPLED FROM MGRA 100 TO MGRA 73

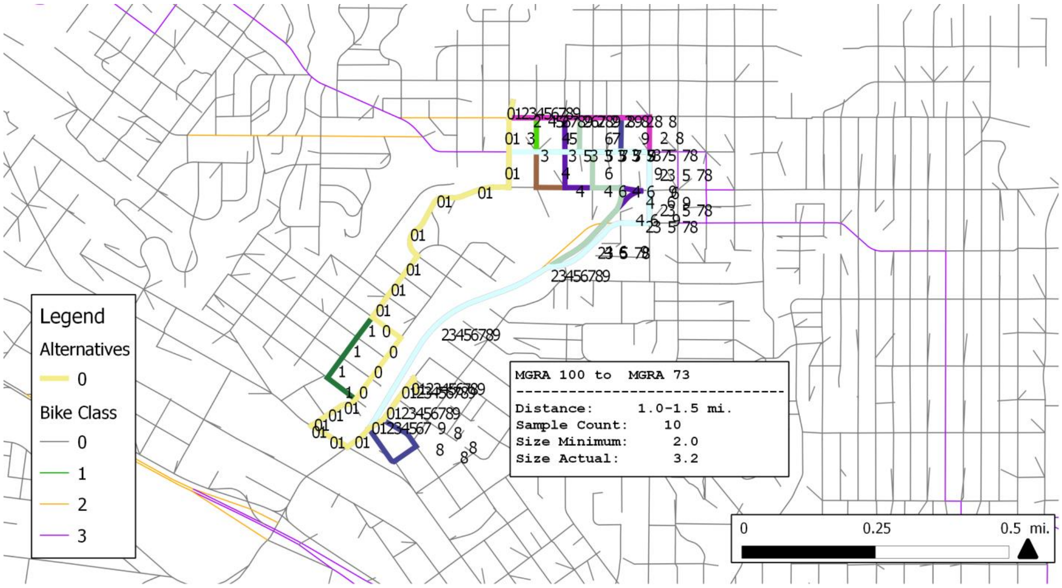


FIGURE 2.4 EXAMPLE PATHS SAMPLED FROM MGRA 100 TO MGRA 2953

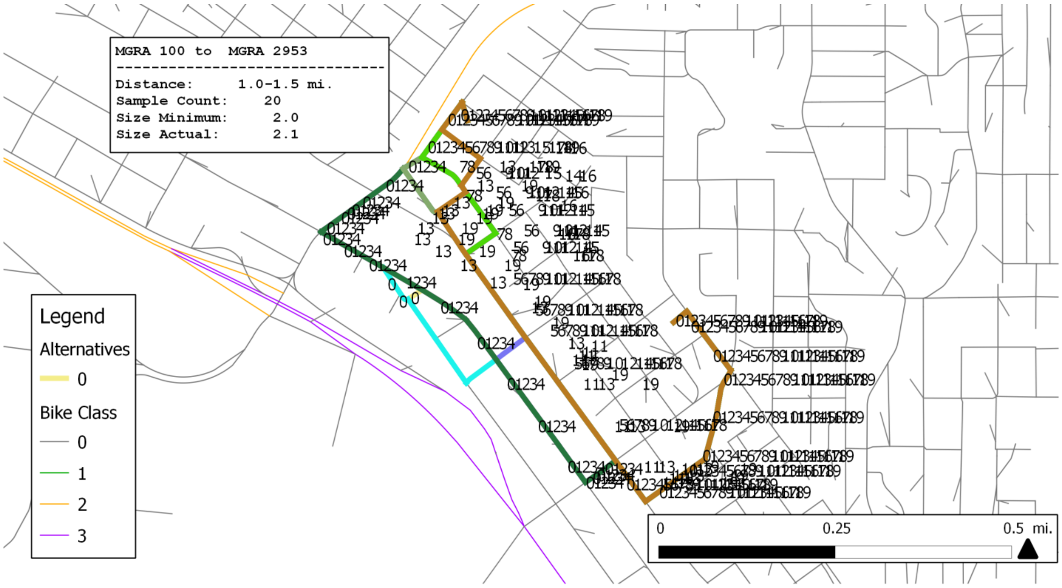
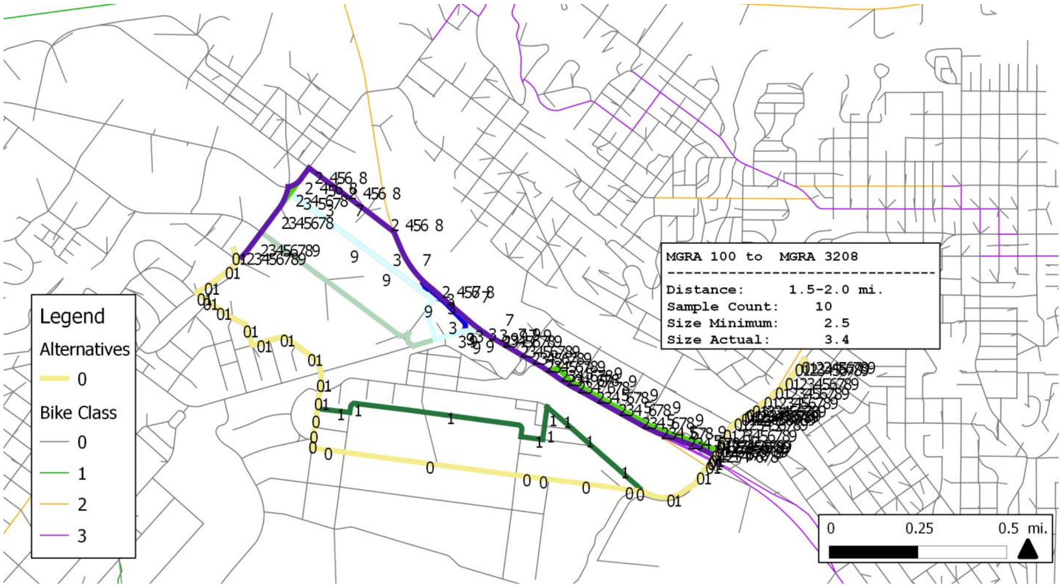


FIGURE 2.5 EXAMPLE PATHS SAMPLED FROM MGRA 100 TO MGRA 3208



Path Resampling

Finding several shortest paths between an origin and destination with a randomized link cost generates a non-uniform random sample of alternatives. In order to estimate a consistent logsum with a non-uniform random sample of alternatives, a correction is required for the probability an alternative was selected for the sample. With the implemented alternative generation method, the sampling alternatives cannot be calculated exactly. Path choice alternative generation methods do exist where the sampling probabilities can be known exactly, but they require too much computation time for this implementation.

Therefore, path resampling is applied to approximate the sampling correction required for the proposed alternative generation method using a bootstrapping procedure. First, a large list of count M paths is sampled using repeated stochastic shortest path searches. These paths will overlap with each other to varying extents, and have different path sizes. The path resampling procedure relies on the fact that, as M→∞, the proportion M_an/M of links using link a converges to the probability of sampling a path that uses the link. Therefore, the length-weighted average of the proportion of paths using links in a given path is a

$$\sum_{a \in \Gamma_i} \frac{l_a}{L_i} \frac{M_{an}}{M}$$

reasonable approximation of the probability of selecting the path. Resampling a smaller number m of alternatives from the larger biased sample with sampling probabilities proportional to the inverse of their estimated original sampling probabilities should provide a final sample of paths drawn with approximately uniform probabilities.

However, a count-m list of paths in one scenario may have a total sum of path sizes which is different from another scenario, and the logsum obtained from the alternative list will be penalized or rewarded unfairly. Therefore, resampling is performed not until a specific number of paths is reached, but rather until a specified total overlap-adjusted size of the choice set is reached. By following this procedure, calculation of the logsum rewards the availability of multiple attractive paths fairly without unfairly penalizing the presence of one or two very attractive, frequently-sampled routes.

It is not be possible to reach the same overlap-adjusted alternative list size for all origin-destination pairs. Short trips may only have one reasonable option, while the stochastic shortest path search will produce many different paths for long trips. Therefore, trips are divided into separate distance bands, with increasing resampling sizes as the distance increases. To prevent path sizes from enhancing the logsum for long trips compared to short trips, the path sizes are normalized so the size of all alternative lists is equal to one.

3.0 SOFTWARE DEVELOPMENT

3.1 | DESIGN OBJECTIVES

PERFORMANCE OBJECTIVES

Speed

The enhancements must complete their tasks within an acceptable time frame.

This objective has the highest priority. To meet this objective, the software makes frequent use of constant-time lookup data structures, a path alternative generation algorithm based on an optimized path search, static types, and parallel processing techniques. The design also includes parameters for algorithmic factors such as the number of path alternative generation iterations and the maximum path search distance that allows users to meet a fixed time criterion if desired.

Accuracy

The difference between the enhancements' average prediction of bicycle road volumes and changes in bicycle and walk mode shares for future scenarios and the best estimates that can be obtained using currently available evidence and methods should be small.

The priority of this objective is second only to the sometimes-conflicting objective of Speed. The latest research has uncovered path alternative sampling methods with known probabilities that achieve a greater accuracy than those selected for the design, but these require too many computations to operate effectively at a regional scale. Therefore, the design reduces errors to the greatest extent possible while maintaining speed by approximating the correction for sampling probabilities using a bootstrapping procedure.

Sensitivity

The outputs of the model should vary with the inputs SANDAG expects to vary during evaluation of future scenarios.

For the bicycle mode, the priority of this objective is high. The bicycle path utility function includes many attributes, and the impedance input to the mode choice model will measure the attractiveness of multiple paths, so mode shares will be responsive to the introduction of facilities parallel to existing alternatives.

For the walk mode, the priority of this objective is low. Available network data does not allow the walk path utility function to contain many attributes, and the walk impedance will be measured only along a single path.

Precision

The degree to which the statistical variation in the model's outputs overwhelms its capability to predict demand changes in small market segments and at small spatial resolutions should be low.

To meet the objective, path alternatives are generated at two different spatial resolutions. Short trips terminate at MGRA centroids and long trips terminate at TAZ centroids. However, the maximum length of these paths is constrained to meet the Speed objective.

Storage

When running, the enhancements must not consume more random access memory than is available on SANDAG machines.

This priority is a hard constraint, but did not require much attention during development because the 64-bit machines offer several dozen gigabytes of memory. Some objects such as uncompressed MGRA-to-MGRA matrices were clearly too large, but storage of a sparse MGRA-to-MGRA matrix with a dictionary-of-keys or array-of-lists format were feasible. Large objects were shared among concurrent threads.

OTHER OBJECTIVES

Interoperability

The enhancements should easily integrate with the rest of the activity-based model and SANDAG's other travel analysis tools.

To meet this objective, the enhancements read and write file formats common to other tools. The software does not add many new dependencies, and re-uses existing components in the activity-based modeling platform.

Modularity

The enhancements should be comprised of loosely-coupled independent components. Modular software is more extensible and maintainable, and allows for separate testing and division of work during development.

To meet this objective, the system follows component-based and object-oriented architectural styles. Design patterns and Generics are employed that abstract behavior out from data structure and implementation. Therefore, new subclasses can modify their implementation of required interfaces and changes will not be required in other modules.

Usability

The enhancements must be operable and maintainable by SANDAG's travel demand forecasters.

To meet this objective, the enhancements are configurable with easily-edited and well-documented properties files.

3.2 | SYSTEM OVERVIEW

OPERATING ENVIRONMENT

The active transportation enhancements are designed to be implemented in Java 7. Parallelization of its tasks is performed with Fork/Join. The enhancements do not introduce any new hardware requirements or dependencies into the software system.

SYSTEM ARCHITECTURE AND PROCESSING FLOW

A diagram of the system architecture for generating a matrix of bike logsums appears in Figure 3.1. The components filled with dark blue are existing components in the SANDAG activity based model application. The components filled with white, gray, and sky blue are new components introduced for the active transportation enhancements. The processing flow for the logsum matrix generation begins at the START node and ends at the END node. Descriptions of the components are introduced below in the order in which they occur in the processing flow. After the logsum matrices are created and then used in CT-RAMP, a separate application (SandagBikePathChoiceEdgeAssignmentApplication) assigns the trips to the network. The system architecture for that application and for the generation of walk utilities is very similar to that of the bike logsum generation application, and more details can be found in the Key Objects with Selected Fields and Methods section below.

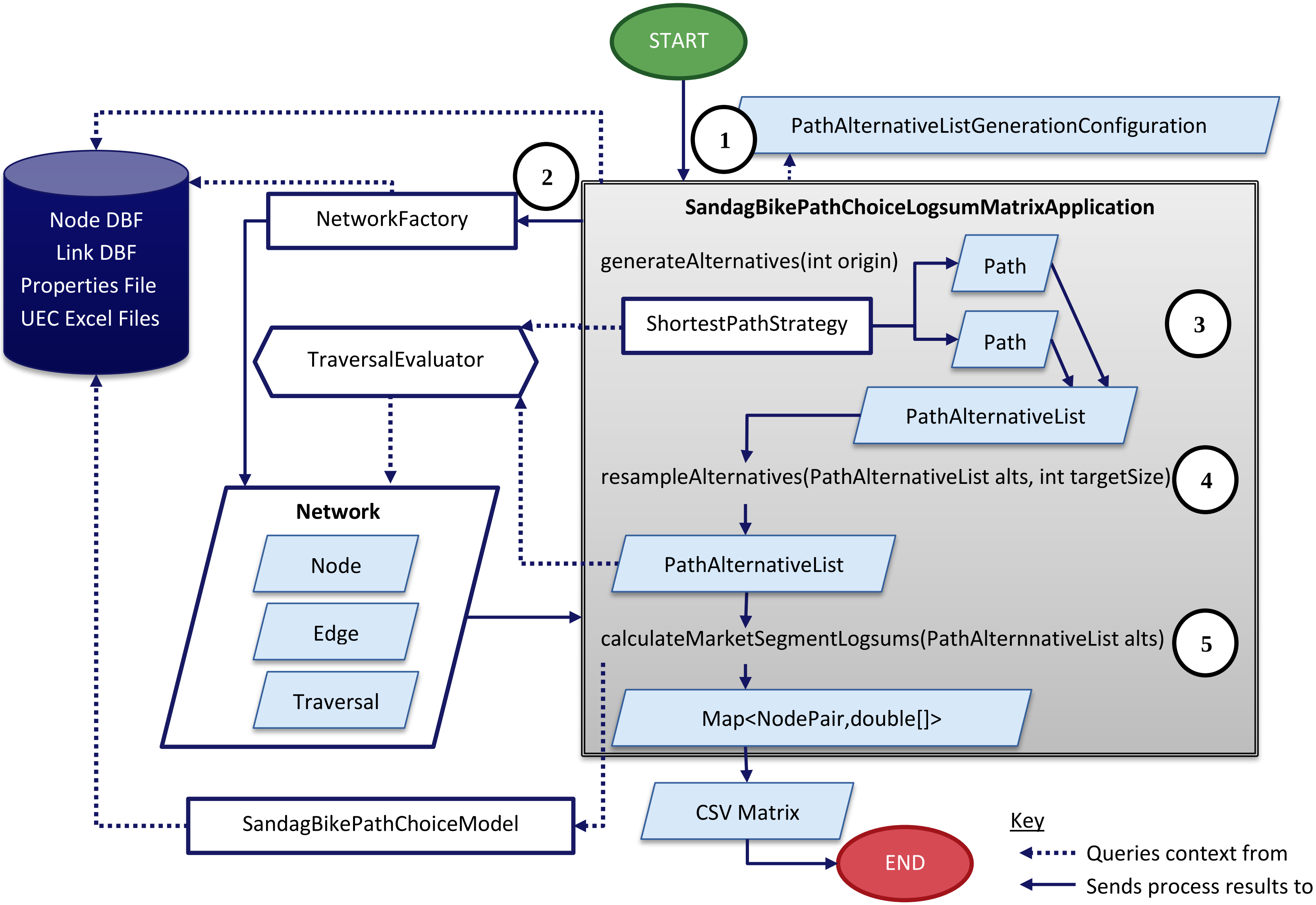
Step 1. Initialization

The system's processing flow is controlled by SandagBikePathChoiceLogsumMatrixApplication. It receives a sparse matrix containing information about the correspondence between network nodes and zone centroids, distances between zone pairs, instructions for how to perform random path sampling, and other configuration information from the Zonal Data Manager in the form of a PathAlternativeListGenerationConfiguration. This information is used by the PathChoiceLogsumMatrixApplication to determine which zone pairs require logsums to be calculated and to customize the path alternative generation algorithms depending on the distances between the zones.

Step 2. Network Construction

After receiving a request to calculate a logsum matrix, the PathChoiceLogsumMatrixApplication uses a NetworkFactory object to read network data from Node and Link DBF Files on the disk and create a Network data structure containing Nodes, Edges, and Traversals (ordered pairs of edges). The NetworkFactory fills the Network object with measured attributes of the network using configuration information read from a Properties File.

FIGURE 3.1 SYSTEM FLOW CHART


START
1
PathAlternativeListGenerationConfiguration
2
Node DBF
Link DBF
Properties File
UEC Excel Files
NetworkFactory
SandagBikePathChoiceLogsumMatrixApplication
generateAlternatives(int origin)
Path
ShortestPathStrategy
Path
3
TraversalEvaluator
PathAlternativeList
resampleAlternatives(PathAlternativeList alts, int targetSize)
4
Network
Node
Edge
Traversal
PathAlternativeList
calculateMarketSegmentLogsums(PathAlternnativeList alts)
5
Map<NodePair,double[]>
SandagBikePathChoiceModel
CSV Matrix
END
Key
Queries context from
Sends process results to

Step 3. Shortest Path Finding

The PathChoiceLogsumMatrixApplication then creates several runnable tasks for parallel processing in a Fork/Join framework. The first procedure in the task, generateAlternatives, uses a ShortestPathStrategy object for every origin zone for which logsums are required to find the best path in the network from the origin zone to every destination zone within a certain distance from the origin. Each ShortestPathTask is coupled with a TraversalEvaluator, a class following the strategy pattern that allows custom calculation of randomized traversal costs. The number of tasks created for each origin zone, the outward distance searched in each task, and the amount of random dispersion used in the traversal costs are all configured from the Properties File.

After the shortest paths are found, Path representing a sequence of nodes between each origin and destination zone are collated into objects are collated into PathAlternativeList objects, which contain all of the shortest paths between one origin and one destination for every random seed.

Step 4. Path Resampling

The resampleAlternatives method then resamples paths in each PathAlternativeList to reach a pre-defined target choice set size, when accounting for overlap between the paths. Processing in the PathResamplingTask begins by calculating the path size for each path in the PathAlternativeList (see 3.2). Then the PathResamplingTask creates a new PathAlternativeList and resamples paths from the original PathAlternativeList with probabilities proportional to their path sizes until a desired total size of the new PathAlternativeList is reached. The desired total size depends on the distance between the origin and destination, and is configured from the Properties File. In all cases, the final path sizes in the PathAlternativeList are normalized so that the total overlap-adjusted size of each alternative list is the same for every origin and destination.

Step 5. Path Choice Utility Calculation

After receiving the re-sampled alternative lists, the calculateMarketLogsums method then calculates the expected maximum utility or logsum of the generated paths. In the case of the bicycle mode, this logsum is calculated using SandagBikePathChoiceModel, which uses an instance of Parsons Brinckerhoff's UtilityExpressionCalculator (UEC) to evaluate discrete choice utilities, probabilities, and logsums using utility parameters coded in a Microsoft Excel workbook. After collecting all of the logsums into a Map keyed by NodePair objects containing the origin and destination nodes, the PathChoiceLogsumMatrixApplication writes the results to a CSV matrix file. At this point the PathChoiceLogsumMatrixApplication terminates.

3.3 | OBJECTS WITH SELECTED FIELDS AND METHODS

This section contains a list of the objects in the software along with their most important fields and methods.

ORG.SANDAG.ABM.ACTIVE

This package contains all the elementary data structures, algorithms, and abstract and generic classes necessary to perform the active transportation analyis. The org.sandag.abm.active.sandag package contains code specific to the SANDAG implementation (section 4.2). Many of the classes use generics for Node, Edge, and Traversal network elements. These generics are notated with N for Node, E for Edge, and T for traversal throughout.

AbstractNetworkFactory

The responsibility of this object is to provide a few helpful functions for dealing with Traversal objects (ordered pairs of edges forming a turn) in concrete subclasses of NetworkFactory. Subclasses NetworkFactory and implements its getTraversals() method.

AbstractPathChoiceEdgeAssignmentApplication

The responsibility of this object is to provide an algorithm for the generation of path alternatives and the assignment of trips to network edges according to path probabilities. Accepts PathAlternativeListGenerationConfiguration to customize algorithm.

Methods

- **assignTrips(List<Integer> tripNums)**: sets up and executes parallelization of path alternative generation and edge assignment with CalculationTask inner classes.
- **assignTrip(int tripNum, PathAlternativeList<N,E> alternativeList)**: abstract method that delegates assignment of trips to edges in a path alternative list to subclasses.
- **getOriginNode(int tripId)**: abstract method that allows path generation algorithm to know from which origin node paths should be generated for a given trip in the queue.
- **getDestinationNode(int tripId)**: abstract method that allows path generation algorithm to know to which destination node paths should be generated for a given trip in the queue.

Inner Classes

- **CalculationTask**: Runnable class that encapsulates parallelizable parts of algorithm.

Sub-Methods

- generateAlternatives(int tripId): performs shortest path searches and creates PathAlternativeList for a given trip.
- run(): pulls trips off of queue, generates alternatives, and increments edge volumes according to trip path choice probabilities.

AbstractPathChoiceLogsumMatrixApplication

The responsibility of this object is to provide an algorithm for the generation of path alternatives and the calculation of a matrix of the expected maximum path choice utilities or logsums. Accepts PathAlternativeListGenerationConfiguration to customize algorithm.

Methods

- **calculateMarketSegmentLogsums()**: sets up and executes parallelization of path alternative generation and logsum calculation with CalculationTask inner classes.
- **calculateMarketSegmentLogsums(PathAlternativeList<N,E> alternativeList)**: abstract method that delegates calculation of logsum for a given path alternative list to subclasses.

Inner Classes

- **CalculationTask**: Runnable class that encapsulates parallelizable parts of algorithm.

Sub-Methods

- **generateAlternatives(int origin)**: performs shortest path searches and creates PathAlternativeLists for all destinations from a given origin zone.
- **resampleAlternatives(PathAlternativeList<N,E> alts, double targetSize)**: resamples generated alternatives in a path alternative list with probabilities proportional to their path sizes until a target choice set size is reached, accounting for overlap between paths.
- **run()**: pulls origin zones off of queue, generates alternatives, and calculates logsum values.

AbstractShortestPathResultSet

The responsibility of this object is to provide useful functions for subclasses implementing ShortestPathResultSet.

BasicShortestPathResultSet

The responsibility of this object is to provide a concrete implementation of ShortestPathResultSet.

BinarySearch

The responsibility of this object is to provide a binary search function for resampling of alternatives in AbstractPathChoiceLogsumMatrixApplication.

CompositeShortestPathResultSet

The responsibility of this object is to combine ShortestPathResultSet objects together into one ShortestPathResultSet.

DestinationNotFoundException

The responsibility of this object is to provide a specific exception for cases where shortest path searches do not find a required destination.

Edge

The responsibility of this object is to provide an interface for objects that can represent links between pairs of Node objects in a Network.

Methods

- **getFromNode()**: returns first node in edge.
- **getToNode()**: returns second node in edge.

EdgeEvaluator

The responsibility of this object is to provide an interface for objects than can evaluate properties of edges in a network. Its primary use is in ShortestPathStrategy and other algorithms where edge costs need to be calculated in a way that is decoupled from the algorithm code.

Methods

- **evaluate(E edge)**: returns a numerical property of the edge

IntrazonalCalculation

The responsibility of this object is to provide an interface for estimating logsums or impedances for trips whose origin and destination are the same using the values for nearby nodes. Its primary use is in the AbstractPathChoiceLogsumMatrixApplication class after logsums have been generated for interzonal origin-destination exchanges.

IntrazonalCalculations

The responsibility of this object is to provide a concrete implementation of IntrazonalCalculations.

ModifiableShortestPathResultSet

The responsibility of this object is to provide an interface that extends ShortestPathResultSet to allow shortest path results to be added to a ShortestPathResultSet during the performance of a ShortestPathStrategy. After shortest path computations have been completed, the ModifiableShortestPathResultSet is cast to ShortestPathResultSet so as to be immutable to clients.

Network

The responsibility of this object is to provide an interface that allows users to obtain information about the relationships between Node, Edge, and Traversal objects in a transportation network.

Methods

- **getNode(int id)**: returns the Node object with the given id.
- **getEdge(N fromNode, N toNode)**: returns the Edge object with the given first and second nodes.
- **getTraversal(E fromEdge, E toEdge)**: returns the Traversal object with the fiven first and second edges.
- **getSuccessors(N node)**: returns a Collection of all nodes that are the terminal nodes of edges starting from node.
- **getPredecessors(N node)**: returns a Collection of all nodes that are the starting nodes of edges terminating at node.

NetworkFactory

The responsibility of this object is to establish a template method for creating a Network object.

Methods

- **createNetwork()**: template method for network construction.
- **getNodes()**: abstract method for getting Collection of Nodes.
- **getEdges()**: abstract method for getting Collection of Edges.
- **getTraversals()**: abstract method for getting collection of Traversals.
- **calculateDerivedNodeAttributes(Network network)**: abstract method for calculating attributes of Nodes not found in original data.
- **calculateDerivedEdgeAttributes(Network network)**: abstract method for calculating attributes of Edges not found in original data.
- **calculateDerivedTraversalAttributes(Network network)**: abstract method for calculating attributes of Traversals not found in original data.

Node

The responsibility of this object is to establish an interface for objects that can serve as nodes in a Network.

Methods

- **getId()**: return unique identifier.

NodePair

The responsibility of this object is to provide a data structure representing ordered pairs of nodes, typically corresponding to the origins and destinations of trips.

ParallelSingleSourceDijkstra

The responsibility of this object is to implement ShortestPathStrategy with a parallelized version of Dijkstra's algorithm that loops over origins (sources). It is not actually used in the model because in the final version, parallelization is performed over origins or trips in AbstractPathChoiceEdgeAssignmentApplication and AbstractPathChoiceLogsumMatrixApplication themselves.

Path

The responsibility of this object is to represent sequences of nodes in a network linking an origin to a destination. To reduce the memory footprint of paths from a single origin to multiple destinations, the paths are coded in a tree format with each path containing a single node and a reference to the Path containing all nodes up to its predecessor.

PathAlternativeList

The responsibility of this object is to represent a collection of possibly different paths between the same origin and destination, and to encapsulate operations such as the calculation of overlap between them that need to be performed on these collections of paths when considered as alternatives in a path choice model.

Methods

- **add(Path<N> path)**: adds a path to the list and updates path size measures if a PathSizeCalculator has been initialized to calculate the path sizes of alternatives in the list.
- **getSizeMeasures()**: returns a list containing the the size measures (or degree of overlap) between a given path and the other alternatives in the list.
- **getSizeMeasureTotal()**: returns the total of the path sizes of all alternatives in the list.
- **getCount()**: returns the number of alternatives in the list.
- **get(int index)**: returns the Path at the given index.
- **areSizeMeasuresUpdated()**: returns true if path sizes have been updated for all paths in the alternative list.
- **clearPathSizeCalculator()**: frees memory used by path size calculator. If paths are added after calling this method, path sizes will not be updated until restartPathSizeCalculator() is called.

- **restartPathSizeCalculator()**: re-initializes PathSizeCalculator, updates sizes of all alternatives in the list, and enables updating of size measures when new paths are added to the list.

Inner Classes

- **PathSizeCalculator**: provides efficient storage of information regarding overlap of paths in alternative list and calculation of size measures.

PathAlternativeListGenerationConfiguration

The responsibility of this object is to encapsulate all of the information needed to customize the algorithms in AbstractPathChoiceEdgeAssignmentApplication and AbstractPathChoiceLogsumMatrixApplication.

Methods

- **getNetwork()**: returns the Network in which path choice alternatives should be found.
- **getEdgeLengthEvaluator()**: returns an EdgeEvaluator which, when its evaluate(E edge) method is called, will return a value corresponding to the physical arc length of the edge. It is used to calculate the overlap and path size measures in each PathAlternativeList generated by the algorithms.
- **getMaxCost()**: returns the maximum allowable path cost in the implemetation of Dijkstra's algorithm used to generate path alternatives. If the shortest path seach encounters a node via a path which has a total cost that exceeds this value (usually due to disallowed links such as freeway ramps), the node will not be placed onto the heap of all nodes which may be part of the shortest path to the destination. This exclusion allows heap processes to be performed more efficiently and save compuation time.
- **getSampleDistanceBreaks()**: returns an array of distances corresponding to the upper end of the distance bins for which different choice set sizes will be found. For example, if this function returns {1.0, 5.0, 10.0, 99.9}, then the total choice set size will be controlled separately for trips between 0.0 and 1.0 mile(s), 1.0 and 5.0 mile(s), 5.0 and 10.0 miles, and 10.0 miles and 99.9 miles}.
- **getSamplePathSizes()**: returns an array of target choice set sizes, accounting for overlap, for each trip within the distance bins given by getSampleDistanceBreaks(). For example, if the distance breaks were as above, and this function returned {1,2,3,1}, then the target choice set size for trips with a distance between 5.0 and 10.0 miles would be 3.
- **getSampleMinCounts()**: returns an array of the minimum number of paths to be sampled for each trip within the distance bins given by getSampleDistanceBreaks(). For example, if the distance breaks were as above, and this function returned {1,20,20,1}, then at least twenty paths will be sampled for trips with a distance

between 5.0 and 10.0 miles. Oversampling paths allows the algorithm to estimate a sampling probability correction in a bootstrapping procedure.

- **getSampleMaxCounts()**:returns an array of the maximum number of paths to be sampled for each trip within the distance bins given by getSampleDistanceBreaks(). For example, if the distance breaks were as above, and this function returned {1,100,100,1}, then at most one hundred paths will be sampled for trips with a distance between 5.0 and 10.0 miles. If the target choice set size cannot be reached after this number of samples, the algorithm ceases sampling more paths. The total number of origin-destination pairs for which the target choice set size could not be reached is reported to the console during each run of AbstractPathChoiceLogsumMatrixApplication.
- **isRandomCostSeeded()**: returns True if randomization of link costs should be seeded so results are reproducible.
- **getNearbyZonalDistanceMap()**: returns a Map<Integer, Map<Integer,Double>> keyed with origin zones and with values corresponding to the distances to each destination for which paths are required. For example, if origin zone 1 is mapped to <2: 1.3, 3: 1.6>, then the distance between zone 1 and 2 is 1.3 units, and the distance between zone 1 and zone 3 is 1.6 units. The presence of zone pairs in this map is used to control which origin-destination pairs have path alternatives generated for them. The distances are used to control the total path sampling using the distance bins in getSampleDistanceBreaks().
- **getOriginZonalCentroidIdMap()**: returns a Map giving the correspondence between origin zones and their network node ids. Also, the key set of this map is the set of origin zones for which paths will be generated.
- **getDestinationZonalCentroidIdMap()**: returns a Map giving the correspondence between destination zones and their network node ids. Also, the key set of this map is the set of destination zones for which paths will be generated.
- **getOutputDirectory()**: returns a String giving the location on disk where output files should be written.
- **getTraceOrigins()**: returns a set of origin zones for which paths should be written to disk for debug or analysis purposes. For each origin in this list, files will be written to the output directory containing the node id sequence of every path in each path alternative list generated. The writing is performed by the PathAlternativeListWriter object.
- **getPropertyMap()**: returns the Map of String pairs generated from the resource bundle used for the model run.
- **getInverseOriginZonalCentroidIdMap()**: returns the inverse of the Map given by getOriginZonalCentroidIdMap.

- **getDestinationOriginZonalCentroidIdMap()**: returns the inverse of the Map given by getDestinationZonalCentroidIdMap.
- **isTraceExclusive()**: returns True if paths should be generated and logsums calculated only for those origins returned by getTraceOrigins(). Returns False if paths and logsums are needed for all origins in getOriginZonalCentroidIdMap().
- **getRandomizedEdgeCostEvaluator(int iter, long seed)**: returns an EdgeEvaluator which, when its evaluate(E edge) method is called, will return a randomized link cost that will be minimized in the shortest path search. This randomization over successive iterations results in different paths being sampled for alternative generation.
- **isIntrazonalsNeeded()**: returns True if the logsum matrix should include intrazonal values.
- **getDefaultMinutesPerMile()**: returns an assumed inverse speed for estimation of actual times in addition to logsum values.

PathAlternativeListWriter

The responsibility of this class is to write the contents of a PathAlternativeList to disk for tracing and debugging.

RepeatedSingleSourceDijkstra

The responsibility of this object is to find the paths in a Network that minmize a given cost function, and calculate the costs of those paths. The class uses Dijkstra's algorithm with heap queue, with repeated searches from a single origin to all relevant destinations. Implements ShortestPathStrategy. This strategy is the one used in path alternative list generation, although in that case only one origin node is sent to RepeatedSingleSourceDijkstra, so this class doesn't actually repeat anything. The repetition over origin nodes is performed by AbstractPathChoiceEdgeAssignmentApplication and AbstractPathChoiceLogsumMatrixApplication.

Fields

- edgeEvaluator: an EdgeEvaluator whose evaluate() method gives the contribution from an edge toward the path cost to be minimized.
- traversalEvaluator: a TraversalEvaluator whose evaluate() method gives the contribution from a traversal toward the path cost to be minimized.

ShortestPathResult

The responsibility of this object is to provide a data structure combining a NodePair, Path, and cost value together in a single class returned by shortest path searches.

ShortestPathResultSet

The responsibility of this object is to provide an interface for clients of ShortestPathStrategy objects to access the results. Combines several ShortestPathResult objects into a single result set.

ShortestPathStrategy

The responsibility of this object is to provide an interface for flexible shortest path search algorithims to operate interchangably in the strategy design pattern. Clients of the interface can swap out the specific implementation without requiring other code changes.

Methods

- **getShortestPaths(Set<N> originNodes, Set<N> destinationNodes, double maxCost)**: returns a ShortestPathResultSet with the paths and costs for all pairs in the cartesian product of originNodes and destinationNodes where the total path cost is less than maxCost.
- **getShortestPaths(Set<N> originNodes, Set<N> destinationNodes)**: returns a ShortestPathResultSet with the paths and costs for all pairs in the cartesian product of originNodes and destinationNodes where any path can be found.

SimpleEdge

The responsibility of this object is to provide a bare-bones concrete implementation of Edge that more complex Edge implementations can subclass or compose with.

SimpleNetwork

The responsibility of this object is to provide a bare-bones concrete implemntation of Network that more complex Network implementations can subclass or compose with.

SimpleNode

The responsibility of this object is to provide a bare-bones concrete implemntation of Node that more complex Node implementations can subclass or compose with.

SimpleTraversal

The responsibility of this object is to provide a bare-bones concrete implemntation of Traversal that more complex Traversal implementations can subclass or compose with.

Traversal

The responsibility of this object is to provide an interface for objects that can represent ordered pairs of edges in the Network. The purpose of tracking traversals is to allow path costs to include multi-link variables, such as penalties for turns or un-signalized crossings of arterials. The traversal structure converts the network to a dual structure where all original nodes are edges and all original edges are nodes (Figure 7). The ordinary path a -> b -> c is equivalent to the path in the dual network of (a,b) -> (b,c).

Methods

- **getFromEdge()**: returns first Edge in Traversal.
- **getToEdge()**: returns second Edge in Traversal.

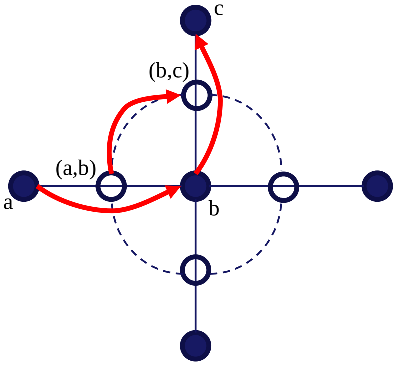
TraversalEvaluator

The responsibility of this object is to provide an interface for objects than can evaluate properties of traversals in a network. Its primary use is in ShortestPathStrategy and other algorithms where traversal costs need to be calculated in a way that is decoupled from the algorithm code.

Methods

- evaluate(T traversal): returns a numerical property of the traversal.

FIGURE 3.2 TRAVERSAL NETWORK REPRESENTATION



ORG.SANDAG.ABM.ACTIVE.SANDAG

This package customizes the code in the org.sandag.abm.active package to the SANDAG implementation.

BikeAssignmentTripReader

The responsibility of this object is to read trip information from CT-RAMP output files, join bicycle trips with household and person information, and prepare for these trips to be assigned to the network.

Methods

- **createTripList()**: reads trip, tour, person, and household files from disk and creates a list of CT-RAMP stop objects. SandagBikePathChoiceLogsumMatrixApplication uses this list to loop over the trips and assign them to the network.

PropertyParser

The responsibility of this object is to provide utilities for parsing lists and other complex properties in the properties file. It's used primarily by SandagBikeNetworkFactory.

SandagBikeEdge

The responsibility of this object is to implement Edge while providing access to a variety of fields specific to the SANDAG implementation.

Fields

- **bikeClass**: type of bicycle facility. 0 – no facility, 1 – bike path, 2 – bike lane (or cycle track), 3 – bike route (or bike boulevard)
- **lanes:** number of vehicular lanes
- **functionalClass**: HPMS functional classification (see network input file documentation)
- **centroidConnector**: True if link connects to a zone centroid
- **autosPermitted**: True if autos are permitted on the link
- **cycleTrack**: True if bike lane is a "cycle track"
- **bikeBlvd**: True if bike route is a "bike boulevard"
- **distance**: arc length of Edge in miles
- **gain**: non-negative elevation change in feet
- **bikeCost**: generalized cost of travel by bike for averaged market segment using coefficients in properties file
- **walkCost**: generalized cost of travel by walk

SandagBikeMgraPathAlternativeListGenerationConfiguration

The responsibility of this object is to configure path generation algorithms for bike paths between MGRAs. Subclasses SandagBikePathAlternativeListGenerationConfiguration and implements its abstract method createZonalCentroidIdMap() to associate MGRA origins with network nodes.

SandagBikeNetworkFactory

The responsibility of this object is to implement NetworkFactory for the SANDAG implementation. It reads the node and edge DBF files, constructs the Node and Edge objects, and calculates derived fields that do not appear in the input data.

SandagBikeNode

The responsibility of this object is to implement Node while providing access to a variety of fields specific to the SANDAG implementation.

Fields

- **x**: x coordinate from input file
- **y**: y coordinate from input file
- **mgra**: id of MGRA for which this node is a centroid, zero otherwise
- **taz**: id of TAZ for which this node is a centroid, zero otherwise
- **tap**: id of TAP to which this node is associated, zero otherwise
- **signalized**: true if a traffic signal is present at node
- **centroid**: true if node is an MGRA or TAZ centroid

SandagBikePathAlternativeListGenerationConfiguration

The responsibility of this object is to implement most of the methods in PathAlternativeListGenerationConfiguration and configure path generation algorithms for bike paths. There is one abstract method, createZonalCentroidIdMap(), which is abstract and delegated to SandagBikeTazPathAlternativeListGenerationConfiguration and SandagBikeMgraPathAlternativeListGenerationConfiguration.

Inner Classes

- **RandomizedEdgeCostEvaluator**: the responsibility of this class is to randomize edge costs during path alternative generation. For each new path to be generated, RandomizedEdgeCostEvaluator is initialized with random coefficients for edge attributes drawn from a uniform distribution around the average coefficients in the properties file. As the path is being generated during a shortest path search, the randomized edge cost is calculated by summing the product of these randomized coefficients with their respective edge attribute values, and then multiplying the result by a discrete random variable specific to the edge. The variances of these random variables are configurable in the properties file.

SandagBikePathAlternatives

The responsibility of this class is to provide path attributes for the bike path choice decision-making unit (DMU) which calculates path choice probabilities and logsums in SandagBikePathChoiceModel.

SandagBikePathChoiceDmu

The responsibility of this class is to provide SandagBikePathChoiceModel access to trip, tour, person, household, and path attributes for flexible calcuation of path alternatives' utility in using a utility expression calculator (UEC) Excel workbook.

SandagBikePathChoiceEdgeAssignmentApplication

The responsibility of this class is to implement AbstractPathChoiceEdgeAssignmentApplication for the assignment of bicycle trips to the network in the SANDAG implementation. Also provides main executable method to set up and run the application, and write the results to disk.

SandagBikePathChoiceLogsumMatrixApplication

The responsibility of this class is to implement AbstractPathChoiceLogsumMatrixApplication for the calculation of bicycle path choice logsums in the SANDAG implementation. Also provides main executable method to set up and run the application, and write the results to disk.

SandagBikePathChoiceModel

The responsibility of this class is to calculate path choice probabilities and logsums in the SANDAG-specific methods of SandagBikePathChoiceEdgeAssignmentApplication and SandagBikePathChoiceLogsumMatrixApplication.

SandagBikeTazPathAlternativeListGenerationConfiguration

The responsibility of this object is to configure path generation algorithms for bike paths between TAZs. Subclasses SandagBikePathAlternativeListGenerationConfiguration and implements its abstract method createZonalCentroidIdMap() to associate TAZ origins with network nodes.

SandagBikeTraversal

The responsibility of this object is to implement Traversal while providing access to a variety of fields specific to the SANDAG implementation.

Fields

turnType: encodes whether traversal is a left turn, right turn, reversal, or through movement. See documentation of TurnType object.

cost: traversal's contribution to generalized cost of travel by bike for averaged market segment using coefficients in properties file

thruCentroid: true of intermediate node of traversal is a centroid.

signalExclRightAndThruJunction: true if intermediate node contains signal, and traversal is not a right turn or through movement along the straight portion of a "T" intersection.

unsigLeftFromMajorArt: true if intermediate node does not contain a signal, traversal is left turn, and first edge is a major arterial

unsigLeftFromMinorArt: true if intermediate node does not contain a signal, traversal is left turn, and first edge is a minor arterial

unsigCrossMajorArt: true if intermediate node does not contain a signal, traversal is left turn or through movement at a four-or-more-way intersection, and cross-street is major arterial

unsigCrossMinorArt: true if intermediate node does not contain a signal, traversal is left turn or through movement at a four-or-more-way intersection, and cross-street is minor arterial

SandagWalkMgraMgraPathAlternativeListGenerationConfiguration

The responsibility of this object is to configure path generation algorithms for walk paths from MGRAs to MGRAs. Subclasses SandagWalkPathAlternativeListGenerationConfiguration and implements its abstract method createOriginCentroidIdMap() and createDestinationCentroidIdMap() to associate MGRA origins and MGRA destinations with network nodes.

SandagWalkMgraTapPathAlternativeListGenerationConfiguration

The responsibility of this object is to configure path generation algorithms for walk paths from MGRAs to TAPs. Subclasses SandagWalkPathAlternativeListGenerationConfiguration and implements its abstract method createOriginCentroidIdMap() and createDestinationCentroidIdMap() to associate MGRA origins and TAP destinations with network nodes.

SandagWalkPathAlternativeListGenerationConfiguration

The responsibility of this object is to implement most of the methods in PathAlternativeListGenerationConfiguration and configure path generation algorithms for walk paths. There is are two abstract methods, createOriginZonalCentroidIdMap() and createDestinationZonalCentroidIdMap(), which is abstract and delegated to SandagWalkMgraMgraPathAlternativeListGenerationConfiguration, SandagWalkMgraTapPathAlternativeListGenerationConfiguration, and SandagWalkTapMgraPathAlternativeListGenerationConfiguration.

SandagBikePathChoiceLogsumMatrixApplication

The responsibility of this class is to implement AbstractPathChoiceLogsumMatrixApplication for the calculation of walk path choice logsums in the SANDAG implementation. Also provides main executable method to set up and run the application, and write the results to disk.

SandagWalkTapMgraPathAlternativeListGenerationConfiguration

The responsibility of this object is to configure path generation algorithms for walk paths from TAPs to MGRAs. Subclasses SandagWalkPathAlternativeListGenerationConfiguration and implements its abstract method createOriginCentroidIdMap() and createDestinationCentroidIdMap() to associate TAP origins and MGRA destinations with network nodes.

TurnType

The responsibility of this object is to enumerate possible turn types: left turn, right turn, reversal, and none.

3.4 | MODIFICATIONS TO CT-RAMP

INCORPORATING ACTIVE TRANSPORT LEVEL-OF-SERVICE

Software changes to incorporate active transport level-of-service measures included the following:

Replacement of MGRA-MGRA straight-line distance walk times with all-streets network path walk times: Previously, walk times for close-in (within 1.5 miles) MGRA pairs were based upon straight-line distances between MGRA centroids. These walk times were stored in a HashMap in the MgraDataManager class. The revised version of the software replaces these MGRA-MGRA walk times with times that are based upon the all-streets network path. The distance threshold for creation of these walk times was increased to xx miles.

Elimination of TAZ-TAZ walk times for MGRA-pairs that exceed the "close-in" threshold: Previously, walk times for MGRA-pairs that were not included in the "close-in" MGRA-MGRA Hashmap were based upon TAZ-TAZ skimmed distances created in TransCAD, using the off-peak highway network. These were eliminated in favor of increasing the distance threshold and building walk paths on the all-streets network, as described above.

Replacement of MGRA-TAP walk straight-line distance walk times with all-streets network path walk times: Previously, walk times from MGRAs to Transit Access Points (TAPs) were based upon factored straight-line distances between the MGRA centroid and the TAP node. The revised version of the software utilizes MGRA-TAP times skimmed from the all-streets network. After replacement of the MGRA-TAP times, a number of maps were created to analyze and confirm that MGRAs were correctly provided with access to TAPs.

Replacement of MGRA-MGRA and TAZ-TAZ bicycle times with bicycle logsums: Previously, the same method for calculating walk times as described above was also used for calculation of bicycle times; straight-line distances between MGRA centroids were used for close-in MGRA-pairs, while off-peak highway network TAZ skims were used for further apart MGRA pairs. The revised software utilizes the bicycle logsums for MGRA pairs, or if the MGRA pair does not exist in the logsum matrix, the TAZ pair is used, instead of bicycle times. Logsums are specified by direction and by gender. This change was made for tour and trip mode choice models, but in order to focus the limited resources on mode choice calibration, unweighted bicycle times calculated from the all-streets network path are used in the origin-based accessibility calculations.

UTILITY EXPRESSION CALCULATOR CHANGES

In order to utilize the new walk times and bicycle times and logsums, changes were made to the Utility Expression Calculators for tour and trip mode choice and accessibility calculations:

Accessibilities2010.xls: The non-motorized page was modified to refer to the actual bicycle time calculated using the all-streets network.

TourModeChoice2010.xls: All purposes modified to utilize revised walk times and bicycle logsums.

TripModeChoice2010.xls: All purposes modified to utilized revised walk times and bicycle logsums.

Asserted and Calibrated Coefficients

Since actual walk times were used instead of logsums for MGRA-MGRA and MGRA-TAP level-of-service, no changes were made to walk access/egress or walk mode coefficients. Bicycle logsum coefficients were initially asserted as shown in Table 3.1. Note that both tour-level and trip-level mode choice models utilize the same coefficients, though at the tour level round-trip logsums are considered while only one-way logsums are considered for trip mode choice.

TABLE 3.1 ASSERTED LOGSUM COEFFICIENTS

	Bicycle Logsum Coefficient	
Purpose	Tour-level	Trip-level
Work	0.15	0.15
University	0.15	0.15
School	0.15	0.15
Maintenance	0.23	0.23
Discretionary	0.23	0.23
Work-based	0.23	0.23

Subsequently, the tour mode choice models were re-estimated using the new all-streets based bicycle logsums and pedestrian accessibilities. This resulted in revisions to the bicycle logsum coefficients, as shown in Table 3.2.

TABLE 3.2 ESTIMATED LOGSUM COEFFICIENTS

	Bicycle Logsum Coefficient	
Purpose	Tour-level	Trip-level
Work	0.13433	0.06717

University	0.13433	0.06717
School	0.21493	0.06717
Maintenance	0.22000	0.06717
Discretionary	0.22000	0.06717
Work-based	0.23000	0.06717

4.0 NETWORK DEVELOPMENT

This section describes the process developed to create the bicycle network used as the basis for the creation of bicycle accessibility measures. The various steps were a combination of automated and manual processes. The general process was developed in order to promote efficiency and continual QA/QC, and was largely developed in Esri's ArcGIS.

4.1 | DATA SOURCES

A number of datasets were required to develop the bicycle network, including:

- Bike Network Field List.csv
- Roads All shapefile
- Functional Class Field Map.csv
- Bike shapefile
- USGS Digital Elevation Models (DEM)
- MGRA Zones shapefile
- TAZ shapefile
- Grid USGS 75 shapefile

The following is a discussion of each data source listed above:

BIKE NETWORK FIELD LIST.CSV

The attribution for the final Bike Network is outlined in Table 4.1. The bike model requires these fields in order to route traffic. The attributes for the final Bike Network were added to the road geometry (derived from the "Roads All" shapefile), and calculated according to the attribute description.

TABLE 4.1 BIKE NETWORK FIELD LIST

Attribute	Type	Description	Units
A	LONG	Foreign key of first node	
B	LONG	Foreign key of second node	
Distance	DOUBLE	Arc length of link	Miles
AB_Gain	LONG	Cumulative non-negative increase in elevation from A to B nodes	Feet Integer
BA_Gain	LONG	Cumulative non-negative increase in elevation from B to A nodes	Feet Integer
ABBikeClas	LONG	Type of Bike Facility in AB direction	0: None;

			1: Off-street path;
			2: On-street lane;
			3: On-street signed route

BABikeClas	LONG	Type of Bike Facility in BA direction	0: None;
			1: Off-street path;
			2: On-street lane;
			3: On-street signed route

AB_Lanes	LONG	Vehicle Lanes in AB direction	
BA_Lanes	LONG	Vehicle Lanes in BA direction	
Func_Class	LONG	Type of Road Facility	Using FHWA HPMS guidelines
A_Elev	DOUBLE	Elevation of A Node	Feet Real
B_Elev	DOUBLE	Elevation of B Node	Feet Real

ROADS ALL SHAPEFILE

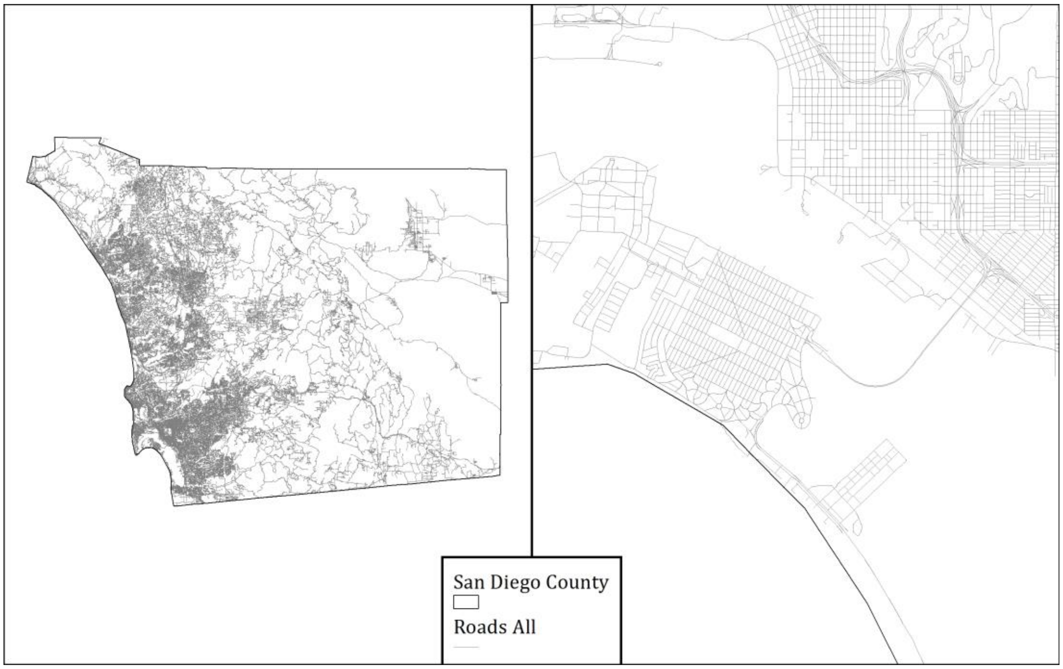
The street geometry for the final Bike Network was developed from the SanGIS "Roads_all" shapefile (referred to as "Roads All" shapefile) which is an All-Streets centerline network. The following fields appear in the final Bike Network for association with the original "Roads All" data:

- **[ROADSEGID]:** Road segment (link) unique identifier
- **[RD20FULL]:** Road segment name

The "Roads All" shapefile also contains many attributes that were used to inform the attribution of the final Bike Network. The following fields were used to inform the attribution of the final Bike Network:

- **[FNODE] & [F_LEVEL]:** The [A] Node attribute in the final Bike Network was informed by a concatenation of the [FNODE] & [F_LEVEL] fields
- **[TNODE] & [T_LEVEL]:** The [B] Node attribute in the final Bike Network was informed by a concatenation of the [TNODE] & [T_LEVEL] fields
- **[ONEWAY]:** The [AB_Lanes] and [BA_Lanes] attributes in the final Bike Network were informed by the [ONEWAY] attribute. The [ONEWAY] field informed the presence of traffic flow in a particular direction.
- **[FUNCLASS]:** The [Func_Class] attribute in the final Bike Network was derived from a conversion of this field to the HPMS functional class designations

FIGURE 4.1 ROADS_ALL



FUNCTIONAL CLASS FIELD MAP.CSV

The Functional Class designations in the “Roads All” shapefile data (outlined in the “Roads All” [FUNCLASS] field) do not match the Functional Classifications used in SANDAG’s transportation model and do not match the HPMS Functional Classification designation system. An equivalence table was created to convert the original “Roads All” Functional Classifications to the final Functional Classification used in the Bike Network.

TABLE 4.2 FUNCTION CLASS FIELD MAP

[FUNCLASS]	Description	HPMS	SANDAG
W	Pedestrian/bikeway	0	0
1	Freeway to Freeway Ramp	2	8
2	Light (2-lane) Collector	6	4
3	Rural Collector	6	6
4	Major road/4-lane major road	4	3
5	Rural Light Collector/local road	6	6
6	Prime (primary) arterial	3	2
7	Private Street	7	7
8	Recreational Parkway	0	0
9	Rural Mountain Road	7	7

A	Alley	7	7
B	Class I Bicycle Path	0	0
C	Collector/4-lane collector street	5	4
D	Two-Lane Major Street	4	3
E	Expressway	2	2
F	Freeway	1	1
L	Local Street/cul-de-sac	7	7
M	Military street within base	7	7
P	Paper Street	-1	0
Q	Undocumented	-1	0
R	Freeway/expressway on/off ramp	2	9
S	Six-Lane Major	3	3
T	Transitway	7	7
U	Unpaved Road	7	7

BIKE SHAPEFILE

The spatial dispersal of San Diego's Bike Network infrastructure was captured from the SanGIS maintained "Bike" shapefile. The "Bike" data could not be table joined to the "Roads All" data as there is no Unique Identifier in the "Bike" shapefile. A spatial relationship had to be developed between the "Bike" shapefile to the "Roads All" shapefile in order to transfer data.

The "Bike" shapefile contains two attributes:

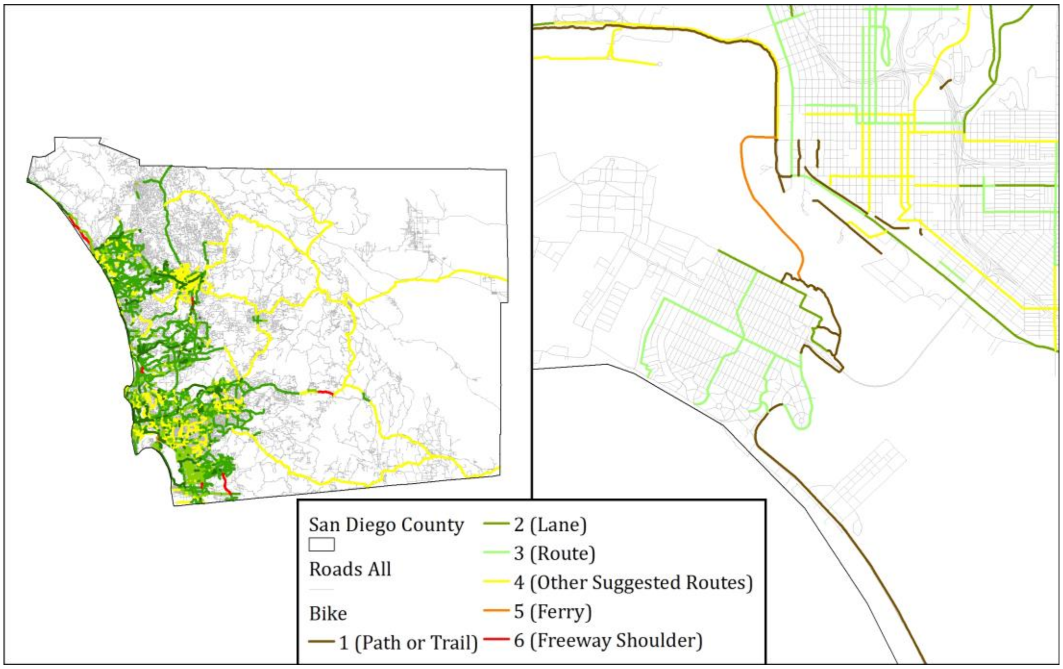
- **[RD20FULL]:** Road Name or Bicycle Path Name
- **[ROUTE]:** Bike route type including the following types
    - 1 (Path or Trail)
    - 2 (Lane)
    - 3 (Route)
    - 4 (Other Suggested Routes)
    - 5 (Ferry)
    - 6 (Freeway Shoulder)

The first three route types (1, 2, and 3) were migrated by a spatial join to the "Roads All" features to inform the AB (and BA) Bike Classification designation in the final Bike

Network. Where a segment in the "Bike" shapefile could not be spatially joined to the "Roads All" shapefile, the feature was manually copied into the Bike Network.

The "Bike" shapefile is a cartographic representation of San Diego County's bike infrastructure and does not contain the proper connectivity required of routable network features. A routable network would only include polyline features that terminate at the end-point (node) of other polyline features. The "Bike" shapefile contains features that terminate at the edge of other polyline features. This construction forms non-routable intersections and thus a non-routable network. Links exhibiting these traits had to be edited for inclusion in the final Bike Network.

FIGURE 4.2 BIKE ROUTE TYPE SHAPEFILE



USGS DEM

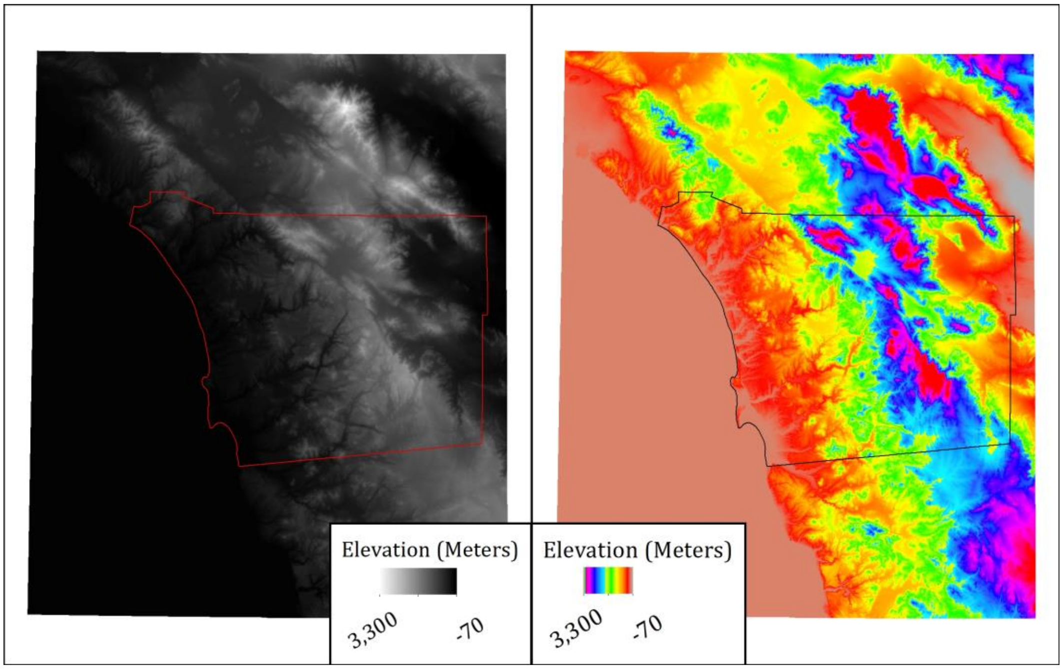
RSG's original plan was to utilize SanGIS topological data to capture elevation information during the Bike Network building process. However, the SanGIS data was deemed unsuitable for the process.

There were two major reasons for this decision: the topological data formats provided by SanGIS and the age of the topological data. SanGIS provides topological data in two data formats: contour lines and ArcInfo Coverage files. The Contour lines were not sensible for performing the requisite vector analysis to pass elevation data to the final Bike Network links. Vector analysis can become cumbersome and error prone when performed using polyline data from disparate sources. The ArcInfo coverage file was not sensible because the "Coverages" are a legacy data format for which Esri supplies limited support in newer ArcMap versions. The second major reason that the SanGIS data was deemed unsuitable is the age of the data. The Meta Data indicates that the contour lines were produced using data collected circa 1999 while the ArcInfo Coverage data was produced using data collected in the mid 1970's.

In place of the SanGIS data RSG chose to use elevation data from the USGS. The USGS maintains the National Elevation Dataset (NED) which provides seamless coverage of the contiguous United States. The NED is updated regularly and is available for the entire conterminous United States at resolutions of 1 arc-second (about 30 meters) and 1/3 arc-second (about 10 meters) and in limited areas at resolutions of 1/9 arc-second (about 3 meters). The elevation layer from The National Map is from the NED.

RSG downloaded 1/3 arc-second elevation data from The National Map Viewer. That elevation data for San Diego County included 4 separate elevation rasters which were subsequently knit together, clipped to the extent of San Diego County, and converted to an Esri Geodatabase raster format.

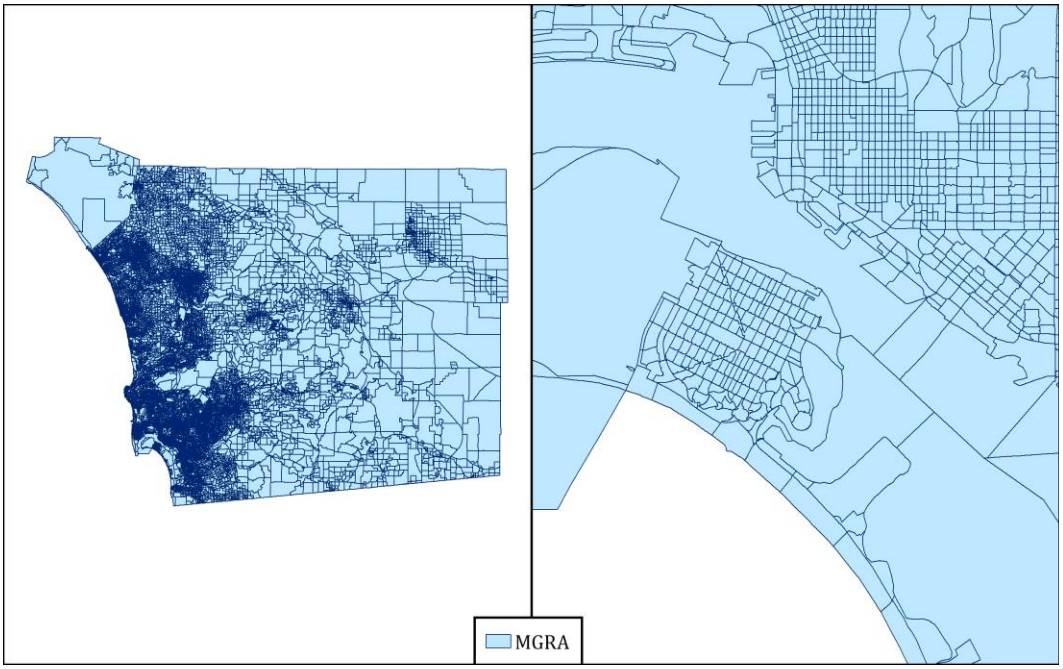
FIGURE 4.3 USGS DEM SHAPEFILE



MGRA ZONES SHAPEFILE

The MGRA zones were required to create MGRA centroids. The MGRA centroids were created based on MGRA zone geometry. The MGRA Centroid creation took place during the network building process. The centroids reflect the geographic centroid of each zone. Each centroid is connected to the network through a single centroid connector which is generated to connect the centroid to the nearest network node.

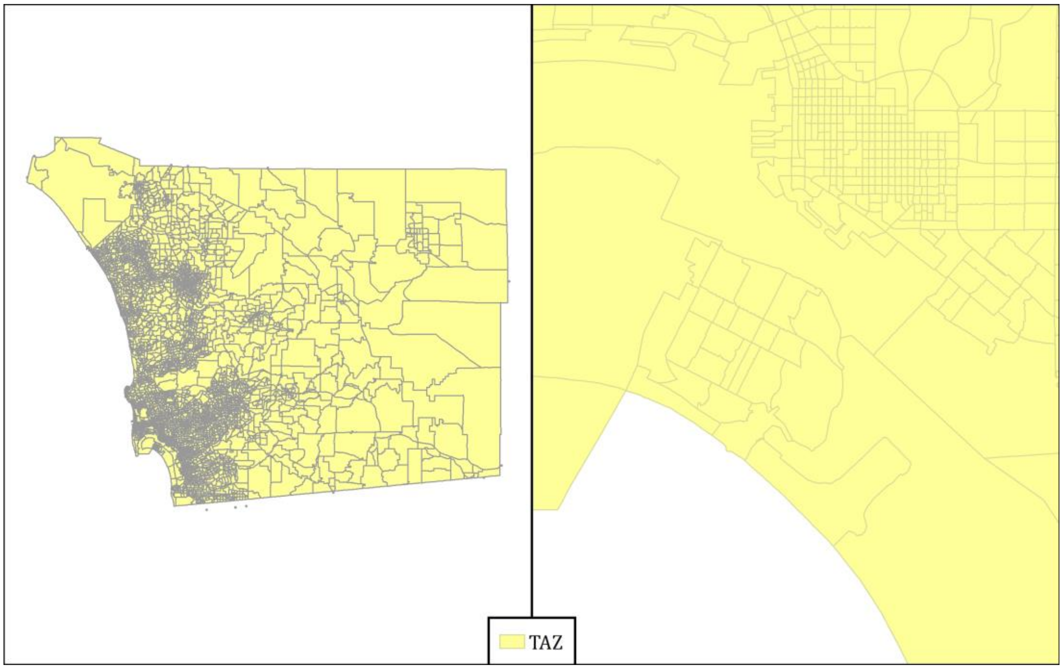
FIGURE 4.4 MGRA SHAPEFILE



TAZ SHAPEFILE

The TAZ were required to create TAZ centroids. The TAZ centroids were created based on TAZ geometry. The TAZ centroid creation took place during the network building process. The centroids reflect the geographic centroid of each zone. Each centroid is connected to the network through a single centroid connector which is generated to connect the centroid to the nearest network node.

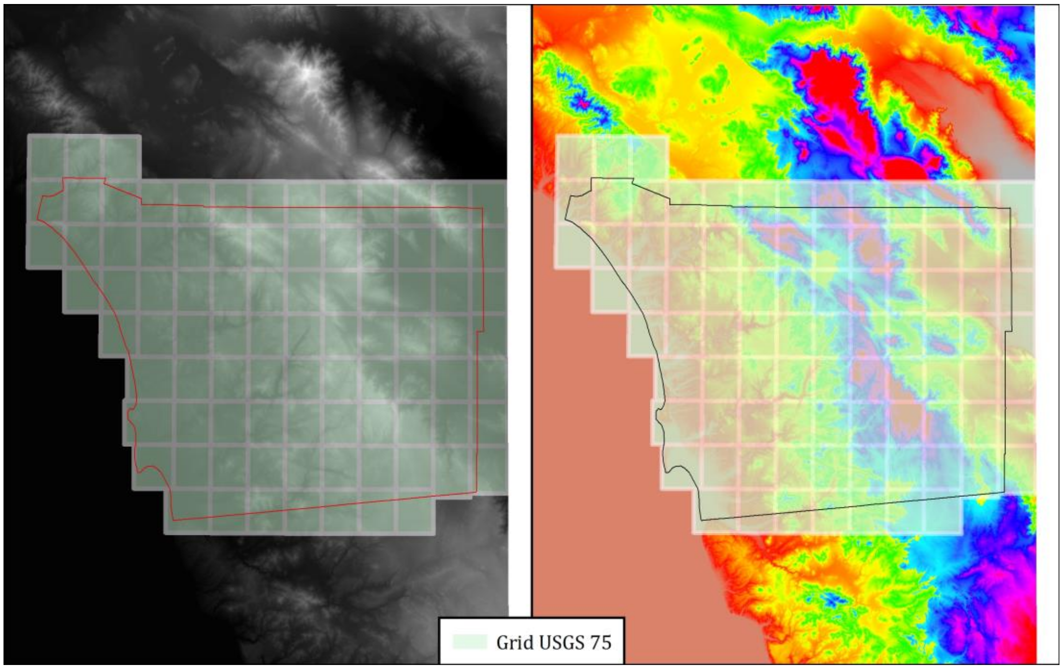
FIGURE 4.5 TAZ SHAPEFILE



GRID USGS 75 SHAPEFILE

The SanGIS Grid was used to disaggregate the other geospatial data. Disaggregating spatial data for geoprocessing minimized the processing time required by each geoprocessing Step and sub-step. The Grid system was especially beneficial in the processing and conversion of elevation data from raster to vector format.

FIGURE 4.6 GRID USGS 75 SHAPEFILE



4.2 | BIKE ROUTE DATA CODING TO ALL-STREETS NETWORK

The first geoprocessing task was to create a relationship between the “Bike” shapefile and the “Roads All” shapefile. As mentioned, there was no way to table join the “Bike” shapefile to the “Roads All” shapefile to transfer “Bike” shapefile [ROUTE] designations. Creating the spatial relationship between the two data sources facilitated the transfer of the “Bike” shapefile’s [ROUTE] attribute to the “Roads All” shapefile features.

This step flags road segment features from the “Roads All” shapefile that are represented in the “Bike” shapefile. Once identified, the road segments are flagged with the bike route designation from the “Bike” shapefile’s[ROUTE] field.

An enumeration of the required “Inputs”, “Outputs”, and “Sub-steps” for this Step (Step 2: Related Bike Route Data to All-Streets Network) follows. The subsequent Steps in the Bike Network building process will be broken up in the same fashion.

INPUTS:

The following inputs are required to perform this Step:

- Roads All shapefile
- Bike shapefile

OUTPUTS:

This Step creates the following outputs (intermediary files are not included in the “Output” list):

- Roads All to Bike Relationship feature class

SUB-STEPS:

This step can be broken down into the following sub-steps:

1) Spatially joined the “Bike” shapefile to the “Roads All” shapefile to create the “Roads All to Bike Relationship” feature class using the “Have their centers in” spatial join option. This associated the “Bike” shapefile class for the links that are spatially aligned with the links in the Bike Network.
2) Links in the “Roads All to Bike Relationship” feature class with a [Join_Count] of 0 were then subset and saved as a feature class in the Working Geodatabase named “Roads All to Bike Non-Join” feature class.
3) Converted the “Roads all to Bike Non-Join” feature class to a point feature class with points every 25 feet along the length of the original line features.
4) The “Roads all to Bike Non-Join” point feature class was then spatially joined to the “Bike” shapefile network. The output of the spatial join was saved as “Road Points to Bike Join” feature class.

5) Summarized the "Road Points to Bike Join" feature class on the [ROADSEGID] field (which originated in the "Roads ALL" shapefile) and the [ROUTE] field (which originated in the "Bike" shapefile) to create the "Road Points to Bike Join Summary" table. This provides a table which summarizes the frequency of "Road Points to Bike Join" features that joined to unique "Bike" shapefile [ROUTE] values. See Table 4.3

6) Subset the "Road Points to Bike Join Summary" to include only those records that unilaterally joined to "Bike" shapefile features with the same [ROUTE]. Saved the subset as the "Road Points to Unique Bike Route Join Summary" table. This means that [ROADSEGID] values that appeared multiple times in the "Road Points to Bike Join Summary" table were removed. See Table 4.4 for an illustration.

7) Added new [Bike_Class] field to the "Roads All to Bike Relationship" feature class.

8) Table joined the "Road Points to Unique Bike Route Join Summary" table to the "Roads All to Bike Relationship" feature class.

9) Filled the [Bike_Class] field in the "Roads All to Bike Relationship" feature class based on the joined table.

10) Removed the table join.

11) Symbolized the "Roads All to Bike Relationship" feature class based on the [Bike_Class] field and visually inspected the accuracy of the relationship.

12) Performed manual edits to the "Roads All to Bike Relationship" feature class where necessary.

TABLE 4.3 ROAD POINTS TO BIKE JOIN SUMMARY EXAMPLE

ROADSEGID	FREQUENCY	ROUTE
50	5	2
51	6	1
51	2	0
52	5	0
53	10	1

TABLE 4.4 ROAD POINTS TO UNIQUE BIKE ROUTE JOIN SUMMARY EXAMPLE

ROADSEGID	FREQUENCY	ROUTE
50	5	2
~~**51**~~	~~6~~	~~1~~
~~**51**~~	~~2~~	~~0~~

52	5	0
53	10	1

4.3 | MANUAL EDITING TO "BIKE" SHAPEFILE FEATURES

While the majority of links within the "Bike" shapefile were successfully related to links within the "Roads All" shapefile, there were a subset of "Bike" links that were not represented in the "Roads All" shapefile. These unrepresented "Bike" links must be added to the "Roads All" links "Working All-Streets Network" feature class. Once identified, "Bike" shapefile links unrepresented in the "Roads All to Bike Relationship" feature class must be manually edited to connect with the "Roads All to Bike Relationship" feature class links and (where necessary) edited to maintain routability within the "Working All-Streets" network.

The "Bike" shapefile was first copied to the Working Geodatabase as a feature class. This copied version of the "Bike" shapefile ("Bike Copy") was edited manually into a routable format that connects with the "Roads All to Bike Relationship" feature class. Edited "Bike Copy" links were flagged if they were to be included in the final Bike Network. Later in this document Step 4: Created a Working Version of All-Streets Network Links will outline how this flag field was used to subset the appropriate "Bike Copy" links to be merged with the "Roads All" links to form the Working All-Streets Network.

INPUTS:

- Bike shapefile
- Roads All to Bike Relationship feature class

OUTPUTS:

- Bike Copy feature class

SUB-STEPS:

1) Copied the "Bike" shapefile to the Working Geodatabase as a feature class named "Bike Copy"
2) Added a field to the "Bike Copy" feature class named [Edit]. Set the [Edit] field to equal 0 for all features. This field acted as a flag field in subsequent steps to identify all links that were not to be included in the final Bike Network.
3) Added the "Bike Copy" feature class and "Roads All to Bike Relationship" feature class to a new ArcMap document.
4) Set the symbolization of the "Roads All to Bike Relationship" feature class to represent the [ROUTE] designation passed from the "Bike" shapefile.

5) Made the “Roads All to Bike Relationship” feature class un-selectable in the “Table of Contents” to avoid inadvertent selection.

6) Opened an edit session for the “Bike Copy” feature class.

7) Visually inspected the “Bike Copy” links and “Roads All to Bike Relationship” feature class. Edited “Bike Copy” Links in the following manner as necessary:

   a. If a “Bike Copy” link terminated near the end point (node) of a “Roads All” link, edited said “Bike Copy” link to terminate at the end point (node) of the nearest “Roads All” Link. Updated the “Edit” field to equal “1” for the edited link.

   b. If a “Bike Copy” link terminated at the edge of a second “Bike Copy” link, split the second “Bike Copy” link at the end point (node) of the first “Bike Copy” Link. Updated the “Edit” field to equal “1” for the edited link.

8) Once all links were visually inspected and edited (where appropriate). Saved edits and finished the edit session.

4.4 | ELEVATION DATA PROCESSING

The USGS elevation data for San Diego was delivered in the format of four raster files. The raster files were combined, clipped to the extent of San Diego County, and then converted to a vector format. The original “Roads All” data was in a vector format and converting the USGS DEMs data to a vector format facilitated geoprocessing. The “Grid USGS 75” shapefile was used to disaggregate the elevation data into manageably sized feature classes. The resulting elevation data consisted of 89 separate feature classes 89 (“QUAD75 Elevation” rasters). Each of the 89 “QUAD75 Elevation” raster was classified by the corresponding “QUAD75” designation in the input grid that was used to clip the “QUAD75 Elevation” raster’s extent. Disaggregating the spatial data significantly decreased geoprocessing run time in subsequent Steps.

Note: This Step required the use of a Spatial Analyst Extension in ArcGIS.

Python Script:

“CreateElevationData.py”

INPUTS:

- USGS DEMs
- Grid USGS 75 shapefile

OUTPUTS:

- Elevation Polygon Feature Classes (89 total feature classes)

SUB-STEPS:

1) Created a copy of the “Grid USGS 75” shapefile as a feature class in the Working Geodatabase named “Grid USGS 75”.
2) Obtained the Clipping Extent (to be used to clip the USGS DEM files) from the “Grid USGS 75” feature class.
3) Created clipped versions of the four Input USGS DEM Files based on the Clipping extent obtained from the “Grid USGS 75” feature class.
4) Merged the clipped versions of the USGS DEM files into one “Elevation” raster saved in the Working Geodatabase.
5) Converted the elevation values in the “Elevation” raster from meters to feet.
6) Created a Search Cursor to loop through features in the “Grid USGS 75” feature class.
7) Used Search Cursor to loop through each feature in the “Grid USGS 75” feature class. Used the selected “Gris USGS 75” feature to inform the creation of a clipped version of the “Elevation” raster. This step created 89 “QUAD75 Elevation” rasters.
8) Converted each “QUAD75 Elevation” raster from a floating point raster to an integer raster (elevation values were preserved in the original raster as floating point numbers but to convert the raster to a polygon feature class the raster must consist of integer values).
9) Converted each “QUAD75 Elevation” raster to a “QUAD 75 Elevation Polygon” feature class.

Working Version of All-Streets Network Links Creation

The “Roads All to Bike Relationship” feature class was merged with the links that were flagged in the “Bike Copy” feature class in Step 2: Related Bike Route Data to All-Streets Network to create the “Working All-Streets Network” feature class. The “Working All-Streets” feature class was then formatted to include the Bike Network fields outlined in the “Bike Network Field List” .csv.

Python Script:

“CreateWorkingBikeNetwork.py”

INPUTS:

- Roads All to Bike Relationship feature class
- Bike Copy feature class
- Bike Network Field List .csv

OUTPUTS:

- Working All-Streets Network feature class

SUB-STEPS:

1) Subset flagged "Bike Copy" links to a temporary feature class file named "Bike Copy for Inclusion"
2) Merged "Roads All to Bike Relationship" feature class and "Bike Copy for Inclusion" links to create the "Working All-Streets Network" feature class.
3) Deleted unneeded fields in the "Working All-Streets" feature class. Prior to deleting these fields, RSG identified the fields from the "Roads All to Bike Relationship" feature class that would subsequently be used to inform the final Bike Network attribution. These fields included:
    - [ROADSEGID]
    - [RD20FULL]
    - [FNODE]
    - [F_LEVEL]
    - [TNODE]
    - [T_LEVEL]
    - [ONEWAY]
    - [FUNCLASS]
    - [FRXCOORD]
    - [FRYCOORD]
    - [TOXCOORD]
    - [TOYCOORD]
    - [Bike_Class] (originated from the [ROUTE] field in the "Bike" shapefile).
4) Added fields listed in the "Bike Network Field List" .csv to the "Working All-Streets Network" feature class. See Table 1 for a list of the fields assed in this step.
5) Filled the newly added [A], [B], [Distance], [AB_Lanes], and [BA_Lanes] using the python code outlined in Table 5.
6) Re-Calculated Geometry Fields maintained from the original "Roads All" shapefile. See Table 6 for an enumeration of the Python Code used to re-calculate geometry field values.
7) Table joined the "Functional Class Field Map" .csv to the "Working All-Streets Network" feature class based on the [FUNCLASS] field.

8) Calculated the [Func_Class] field based on the joined “Functional Class Field Map” .csv values.

9) Calculated the [ABBikeClas] and [BABikeClas] fields based on the [Bike_Class] field. The following logic was used to calculate the [ABBikeClas] and [BABikeClas] fields:

   a. If the [Bike_Class] attribute was equal to 1 (Path or Trail), the [##BikeClas] designation was 1 (Off-Street Path)

   b. If the [Bike_Class] attribute was equal to 2 (Lane) the [##BikeClas] designation was 2 (On-street lane)

   c. If the [Bike_Class] attribute was equal to 3 (Route) the [##BikeClas] designation was 3 (On-street signed route)

   d. All other links were assigned a [##BikeClas] designation of 0(None). This includes links whose [Bike_Class] attribute was equal to 4 (Other Suggested Routes) , 5 (Ferry), or 6 (Freeway shoulder).

TABLE 4.5 WORKING ALL-STREETS NETWORK FIELD CALCULATIONS

Bike Network Field	Roads All Field(s)	Attribution Python Calculation or Function
[A]	[FNODE]; [F_LEVEL]	!FNODE!*10+!F_LEVEL!
[B]	[TNODE]; [T_LEVEL]	!TNODE!*10+!T_LEVEL!

[AB_Lanes]	[ONEWAY]	def Reclass(ONEWAY):
		if (ONEWAY=="T"):
		return 0
		else:
		return 1

[BA_Lanes]	[ONEWAY	def Reclass(ONEWAY):
		if (ONEWAY=="F"):
		return 0
		else:
		return 1

TABLE 4.6 WORKING ALL-STREETS NETWORK GEOMETRY FIELD CALCULATIONS

Bike Network Field	Python Geometry Calculation

[Distance]	*!shape.length@feet!*
[FRXCOORD]	*!shape.firstpoint.X!*
[FRYCOORD]	*!shape.firstpoint.Y!*
[TOXCOORD]	*!shape.lastpoint.X!*
[TOYCOORD]	*!shape.lastpoint.Y!*

4.5 | WORKING VERSION OF ALL-STREETS NETWORK NODES CREATION

The "Working All-Streets Network Nodes" feature class was created from the "Working All-Streets Network" feature class. The "Working All-Streets Network Nodes" were required to add elevation data to the final Bike Network Links, were required for the creation of the "Centroid Connectors", and were included in the final Bike Network Nodes file.

Note that this step also acts as a QA/QC step. The "Roads All" shapefile contains two node ID fields ([FNODE] and [TNODE]) which have been used to ID the "Working All-Streets Network Nodes" feature class. However, there were geometry errors within the "Roads All" shapefile where links did not terminate at the nodes indicated in the attribute table. When first run, this step exported a "Node Error" feature class which identified the 32 "Working All-Streets Network" feature class links that required geometry edits (the errors were legacy errors that originated in the original "Roads All" shapefile). Once the edits were made, the "Working All-Streets Network" feature class and "Working All-Streets Network Nodes" feature classes were regenerated.

The "Roads All" shapefile contained [F_LEVEL] and [T_LEVEL] fields which denoted where there was vertical separation between streets. The "LEVEL" is assigned as an integer and denotes relative separation at a particular location. Nodes at "Ground Level" are assigned to LEVEL 1, while Nodes representing overpasses are assigned to LEVELs 2-8. This "LEVEL" designation was used to inform elevation.

Python Script:

"CreateWorkingBikeNetworkNodes.py"

INPUTS:

- Working All-Streets feature class
- Elevation Polygon Feature Classes (89)

OUTPUT IF ERROR:

- Working All-Streets Network Node Error feature class

OUTPUTS:

- Working All-Streets Network Nodes feature class

SUB-STEPS:

1) Created a XY Event Layer named "Line Start Points" based on the [FRXCOORD] and [FRYCOORD] fields in the "Working All-Streets" feature class. Event layers are temporary layers.

2) Created a XY Event Layer named "Line End Points" based on the [TOXCOORD] and [TOYCOORD] fields in the "Working All-Streets" feature class. Event layers are temporary layers.

3) Saved the "Line Start Points" and "Line End Points" XY Event Layers as "Line Start Nodes" and "Line End Nodes" feature classes (respectively) within the Working Geodatabase.

4) Added a field named [NodeLev_ID] to the "Line Start Nodes" and "Line End Nodes" feature classes. This field served as a unique identifier for the resulting "Working All-Streets Nodes" feature class.

5) Calculated the [NodeLed_ID] fields based on a concatenation of the [FNODE] & [F_LEVEL] fields for the "Line Start Nodes" feature class and a concatenation of the [TNODE]; & [T_LEVEL] fields for the "Line End Nodes" feature class. Concatenating the two fields disaggregates nodes that were vertically separated.

6) Merged the "Line Start Nodes" and the "Line End Nodes" feature classes into one "Road Nodes" feature class.

7) "Dissolved" the "Road Nodes" feature classes based on the [NodeLev_ID] to create the "Working All-Streets Network Nodes" feature class.

8) QA/QC: Created a disaggregated version of the "Working All-Streets Network Nodes" features named "Working All-Streets Network Nodes Test" feature class. This step made use of the "Multipart to Single Part" to split "multipart" point features into "single part" point features.

9) QA/QC: Performed a Count of the features in the "Working All-Streets Network Nodes" feature class and a count of the "Working All-Streets Nodes Test" feature class.

10) QA/QC: When the number of features in the two feature classes was not equal, RSG proceeded to sub-step 10.a . When the number of features was equal, RSG proceeded to sub-step 10.b.

    a. An unequal count signified an error where a node was being created in multiple locations. The following steps were undertaken:

        i. Summarized the "Working All-Streets Nodes Test" feature class based on the [NodeLev_ID] field, saved the summary table as

"Node Error Summary". The summary table contained two fields: [NodeLev_ID] and [FREQUENCY] (which counted the number of features using that [NodeLev_ID] value). Each [NodeLev_ID] should have a [FREQUENCY] of 1.

ii. Added an [Error] Field to the "Working All-Streets Nodes Test" feature class

iii. Table joined the "Node Error Summary" table to the "Working All-Streets Nodes Test" feature class using the [NodeLev_ID] field as the join field.

iv. Filled the [Error] field with a 1 if the joined table had a [FREQUENCY] higher than 1.

v. Subset the "Working All-Streets Nodes Test" feature class where the [Error] field equaled 1. Saved the subset as a feature class named "Working All-Streets Node Error".

vi. Proceeded to Step 5B.

b. An equal count signified that the errors had been resolved. The following steps were undertaken:

i. Created a subset of the "Working All-Streets Network Nodes" features that represented "ground level nodes". This ground level distinction excludes nodes that represent an overpass or bridge and was accomplished by querying nodes with a [NodeLev_ID] ending in "1". The subset was saved as a feature class named "Ground Level Nodes".

ii. Created a "list" of "Elevation Polygon" feature classes (including all 89 "Elevation Polygon" feature classes) for looping.

iii. Spatially joined each "Elevation Polygon" Feature Class to the "Ground Level Nodes" feature class using the "Keep Common" option. The result was 89 "Ground Level to Elevation Join" feature classes.

iv. The "Ground Level to Elevation Join" feature classes were merged into a single temporary "Elevation to Node Join" feature class.

v. Added an [Elevation] field to the "Working All-Streets Network Nodes" feature class.

vi. Table joined the "Elevation to Node Join" feature class to the "Working All-Streets Network Nodes" feature class based on the [NodeLev_ID] field.

vii. Filled the “Working All-Streets Network Nodes” [Elevation] field based on the joined data. Note this only filled the [Elevation] field for Nodes that had been subset into “Ground Level Nodes” feature class.

viii. Removed the table join.

ix. Added a new field named [INTERID] to the “Working All-Streets Network Nodes” feature class and the “Elevation to Node join” feature class.

x. Added a new field to the “Working All-Streets Network Nodes” feature class named [NODELEVEL].

xi. The [INTERID] field was filled with the original node IDs. This means that the values are calculated by removing the final character (right most character) from the [NodeLev_ID] field.

xii. The [NODELEVEL] field was filled with the original Node “Level” assignment. This means that the values are calculated as the final character (right most character) from the [NodeLev_ID] field.

xiii. Table joined the “Elevation to Node join” feature class to the “Working All-Streets Network Nodes” feature class based on the [INTERID] field.

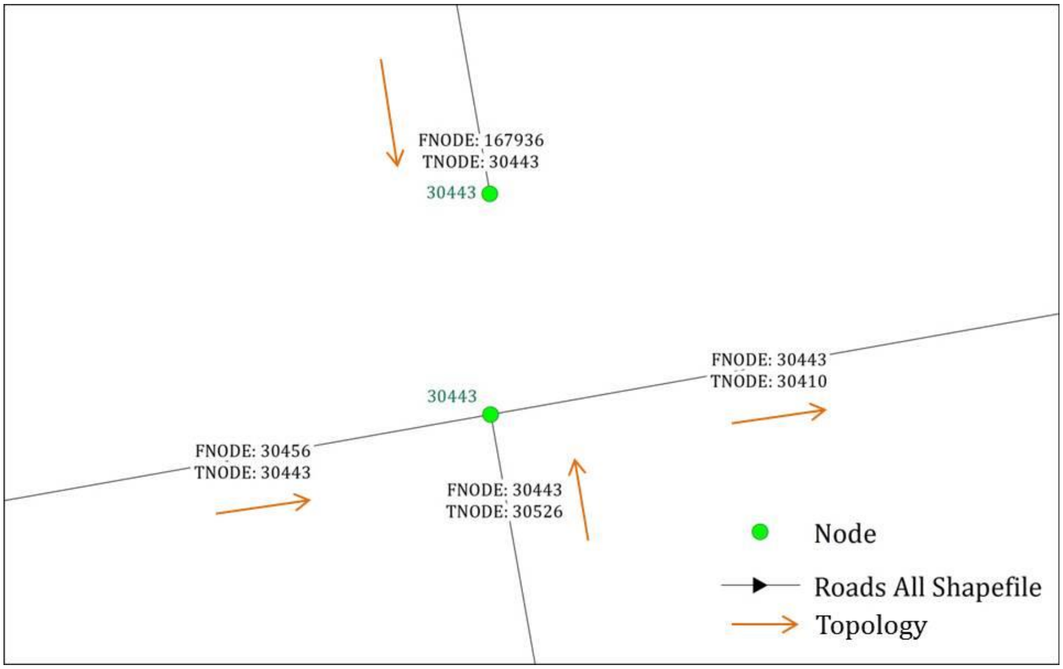
xiv. The elevation of the non-ground level nodes was calculated by adding the product of the Node Level field (an integer value between 2 and 8 for non-ground level nodes) and 25 feet to the elevation of the ground level node at that location.

xv. Removed table join.

4.6 | MANUAL EDITS TO THE ROADS ALL TO BIKE RELATIONSHIP FEATURE CLASS

As outlined above in section 4.5, the original “Roads All” shapefile contained line geometry errors. There were 35 features in the “Roads All” shapefile that required geometry edits. “Errors” were features in the shapefile where the “Roads All” attributes disagreed with one another in informing the location of a Node. For Instance, Figure 7 illustrates where the “Roads All” shapefile features disagree about the location of Node 30443. In this case, the top most line feature should terminate at the same node as the other three features. Errors like this were caught and edited to ensure that nodes were not identified as occurring at multiple discreet locations.

FIGURE 4.7 ROADS_ALL ERRORS



INPUTS:

- Roads All to Bike Relationship feature class
- Working All-Streets Network Node Error feature class

OUTPUTS:

- Roads All to Bike Relationship feature class

SUB-STEPS:

1) Opened an edit session for “Roads All to Bike Relationship” feature class.
2) Zoomed to the first feature in the “Working All-Streets Network Node Error” feature class.
3) Reviewed the topology of the “Roads All” links that enter the node identified in the “Working All-Streets Network Node Error” feature class.
    a. Determined the line feature that was in error.
    b. Edited the line feature that was in error to terminate, or start, at the correct node.
4) Repeated 2 and 3 (above) for all “Working All-Streets Network Node Error” features.
5) Saved and stopped editing.

6) Reviewed the edits made to the "Roads All to Bike Relationship" feature class alongside "Bike Copy" feature class to ensure that the "Bike Copy" feature class did not require any further edits. Repeated Step 2A: Applied Manual Edits to "Bike" shapefile features for inclusion in the Working All-Streets Network as necessary.

7) Re-ran the following steps:

    a. Working Version of All-Streets Network Links Creation

    b. Working Version of All-Streets Network Nodes Creation

Mid-Link Elevation Points Creation

Some of the links in the original "Roads All" and "Bike" shapefile were very long. These long links hid mid-link elevation variability. In order to capture meaningful elevation gain data RSG devised a method to calculate elevation along the length of each "Working All-Streets Network" feature. To accomplish this task "Mid-Link Point" were generated for each feature in the "Working All-Streets Network". The "Mid-Link Point" features were generated at 20 foot intervals along the length of each feature. The "Mid-Link Point" feature class was then spatial joined to the "Elevation Polygon" feature classes to create the "Mid-Link Elevation Point" feature class.

Python Script:

"CreateMLEP.py"

INPUTS:

- Working All-Streets Network feature class
- Working All-Streets Network Nodes feature class
- Elevation Polygon Feature Classes (89)

OUTPUTS:

- Mid-Link Elevation Point feature class

SUB-STEPS:

1) Created an empty version of the "Mid-Link Point" feature class.

2) Added the following fields to the "Mid-Link Point" feature class:

    - [ROADSEGID]
    - [OrderNumber]
    - [A]
    - [B]

3) Created a "Search" cursor to cycle through "Working All-Streets Network" feature class links.

4) Created an "Insert" cursor to create points in the "Mid-Link Point" feature class:
    a. Nested the "Insert" cursor within a loop through the "Search" cursor:
        i. Created point records within "Mid-Link Point" feature class by creating a point at the "start" of each "Working All-Streets Network" link.
        ii. Created a point every 20 feet along each "Working All-Streets Network" link.
        iii. Created a point at the end of each "working All-Streets Network" link.
        iv. Populated the [ROADSEGID], [OrderNumber], [A], and [B] fields in the "Mid-Link Point" feature class based on the attributes of the "Working All-Streets Network" links.
5) Spatially joined each "Elevation Polygon" feature class to the "Mid-Link Point" feature class creating 89 "Mid-Link Point to Elevation Polygon" feature classes.
6) Merged the 89 "Mid-Link Point to Elevation Polygon" feature classes into one file. Saved the merged file as a feature class named "Mid-Link Elevation Point".
7) Table joined the "Working All-Streets Network Nodes" feature class to the "Mid-Link Elevation Point" feature class based on the [A] field in the "Mid-Link Elevation Point" feature class and based on the [NodeLev_ID] field in the "Working All-Streets Network Nodes" feature class.
8) Re-Calculated the [Elevation] field in the "Mid-Link Elevation Point" feature class based on the joined "Working All-Streets Network Nodes" [Elevation] field. This incorporates the overpass and bridge elevation calculations from the [LEVEL] assignments.
9) Removed the table join.
10) Repeated sub-steps 7, 8, and 9 above using the [B] field.

4.7 | WORKING BIKE NETWORK LINKS ELEVATION DATA ASSOCIATION

The "Working All-Streets Network" feature class was copied to create the "Working Bike Network" feature class. This preserved the "Working All-Streets Network" feature class in case an error arose in the subsequent steps. The "Working All-Streets Network Nodes" feature class was then table joined to the "Working Bike Network" feature class to fill the Bike Network [A_Elev] and [B_Elev] fields. The "Working All-Streets Network Nodes" join was then removed and the cumulative non-negative elevation gain was calculated for each link in the AB and BA directions based on the "Mid-Link Elevation Point" feature class.

Python Script:

"JoinElevationData2WorkingBikeNetwork.py"

INPUTS:

- Working All-Streets Network feature class
- Working All-Streets Network Nodes feature class
- Mid-Link Elevation Point feature class

OUTPUTS:

- Working Bike Network feature class

SUB-STEPS:

1) Created the "Working Bike Network" feature class as a copy of the "Working All-Streets network" feature class.
2) Table joined the "Working All-Streets Network Nodes" feature class to the "Working Bike Network" feature class [A] field and the "Working All-Streets Network Nodes" [NodeLev_ID] field.
3) Calculated the [A_Elev] field based on the joined "Working All-Streets Network Nodes" [Elevation] field and the "Working All-Streets Network Nodes" [NodeLev_ID] field.
4) Removed the table join.
5) Table joined the "Working All-Streets Network Nodes" feature class to the "Working Bike Network" feature class [B] field.
6) Calculated the [B_Elev] field based on the joined "Working All-Streets Network Nodes" [Elevation] field.
7) Removed the table join.
8) Created an "Update" cursor to cycle through features in the "Working Bike Network" feature class.
9) Created a "Search" cursor to cycle through features in the "Mid-Link Elevation Point" feature class.
10) Used the "Search" cursor to calculate cumulative non-negative elevation gain across features in the "Update" cursor.
11) Used the "Update" cursor to insert the cumulative non-negative elevation gain calculations into the [AB_Gain] and [BA_Gain] fields of the "Working Bike Network" feature class.

4.8 | ZONE CENTROIDS AND CENTROID CONNECTORS CREATION

The MGRA and TAZ centroids were generated based on zone geography. Once the Zone centroids were created they were merged and connected to the network through the creation of the "Centroid Connectors" feature class which connected each centroid to its nearest accessible "Working All-Streets Network Node" feature. Accessible nodes were nodes that did not represent overpasses, bridges, or limited access highway nodes.

Python Script:

"CreateZoneCentroidsCentroidConnectors.py"

INPUTS:

- MGRA Zones shapefile
- TAZ shapefile
- Working Bike Network feature class
- Working All-Streets Network Nodes feature class

OUTPUTS:

- MGRA Zones feature class
- MGRA Centroid feature class
- TAZ feature class
- TAZ centroid feature class
- Zones Centroid feature class
- Centroid Connectors feature class

SUB-STEPS:

1) Copied the "MGRA Zones" shapefile to the Working Geodatabase as a feature class named "MGRA Zones".
2) Copied the "TAZ" shapefile to the Working Geodatabase as a feature class named "TAZ".
3) Added [XCOORD] and [YCOORD] field to the "MGRA Zones" and "TAZ" feature classes.
4) Filled [XCOORD] and [YCOORD] field of the "MGRA Zones" and "TAZ" feature classes based on the geographic coordinates of each zone's centroid.
5) Made a "MGRA Centroid" and "TAZ Centroid" Event Layer based on the [XCOORD] and [YCOORD] fields.

6) Saved the "MGRA Centroid" and "TAZ Centroid" Event layers as feature classes of the same name.

7) Added a [ZONE_ID] field to the "MGRA Centroid" and "TAZ" feature classes.

8) Calculated the "MGRA Centroid" feature class [ZONE_ID] field using the sum of 100,000,000 and the [MGRA] field.

9) Set the "MGRA Centroid" feature class [TAZ] field equal to 0.

10) Calculated the "TAZ Centroid" feature class [ZONE_ID] field using the sum of 200,000,000 and the [TAZ] field.

11) Merged the "MGRA Centroid" and "TAZ Centroid" feature classes into a single file saved as a feature class named "Zones Centroid".

12) Subset "Working Bike Network" feature class links to exclude limited access highways. The resulting file was saved as a feature class named "Accessible Links".

13) Spatially joined the "Accessible Links" feature class to the "Ground Level Nodes" feature class using the "Keep Common" option. The resulting file was saved as a feature class named "Accessible Nodes". Note that the resulting file excludes nodes that represent overpasses, bridges, and limited access highways.

14) Added [X] and [Y] fields to the "Accessible Nodes" feature class.

15) Calculated the [X] and [Y] fields in the "Accessible Nodes" feature class based on each feature's geographic coordinates.

16) Spatially joined the "Zones Centroid" feature class to the "Accessible Nodes" feature class creating a feature class named "Zones Centroid to Accessible Node". This intermediate file contained both the geographic coordinates of the "Zones Centroid" features ([XCOORD] and [YCOORD]) and the geographic coordinates of the nearest "Accessible Nodes" feature ([X] and [Y]).

17) Created the "Centroid Connector" feature class by making line features starting at the [XCOORD] and [YCOORD] geographic coordinates and terminating at the [X] and [Y] geographic coordinates found in the attribute table of the "Zones Centroid to "Accessible Node" feature class.

18) Added [A], [B], and [Func_Class] fields to "Centroid Connectors" feature class.

19) Filled the "Centroid Connectors" feature class [A] field with values from the [ZONE_ID] field.

20) Filled the "Centroid Connectors" feature class [Func_Class] field with a value of 10.

21) Table joined the "Zone Centroid to "Accessible Node" feature class to the "Centroid Connectors" feature class using the [ZONE_ID] field.

22) Filled the [B] field from the joined [NodeLev_ID] field.

23) Removed table join.

4.9 | LINK FILES AND NODE FILES MERGE AND FINALIZATION

The “Centroid Connectors” feature class was merged with the “Working Bike Network” feature class to create the “Final Bike Network” feature class. The “Zones Centroid” feature class was merged with the “Working All-Streets Network Nodes” feature class to create the “Final Bike Network Nodes” feature class. The resulting feature classes were then copied to shapefile.

Python Script:

“FinalizeBikeNetwork.py”

INPUTS:

- Working Bike Network feature class
- Working All-Streets Network Nodes feature class
- Zone Centroid feature class
- Centroid Connectors feature class

OUTPUTS:

- Final Bike Network feature class
- Final Bike Network Node feature class
- Final Bike Network shapefile
- Final Bike Network Node shapefile

SUB-STEPS:

1) Merged the “Centroid Connectors” and “Working Bike Network” feature classed to create the “Final Bike Network” feature class.
2) Merged the “Zones Centroid” feature class and “Working All-Streets Network Nodes” feature class to create the “Final Bike Network Nodes” feature class.
3) Copied the “Final Bike Network” feature class to a shapefile of the same name.
4) Copied “Final Bike Network Node” feature class to a shapefile of the same name.

5.0 CALIBRATION AND VALIDATION

Observed Data Sources (Heather, Joe)

Mode Choice Calibration (Heather, Joel)

5.1 | BICYCLE ROUTE CHOICE CALIBRATION

CALIBRATION OF ALTERNATIVE GENERATION ALGORITHM PARAMETERS

Calibration of the bicycle route choice model focused on adjusting the parameters of the stochastic choice set generation algorithm to reach a balance between the accuracy of the logsum calculated from the sampled alternatives and the run time required to generate the choice sets. This balance was achieved by adjusting the following parameters.

Maximum Cycling Distance

Increasing the maximum cycling distance improves accuracy in the spatial distribution of bicycle trips by representing the availability and utility of cycling for greater distances in the mode choice model, but greatly increases the run time of alternative generation because the time required to perform a shortest path search generally increases with the square of the maximum distance. Based on the distribution of cycling trips from the household travel survey, we set the maximum cycling distance to 20 miles as the likelihood of cyling beyond this distance is very low.

Spatial Resolution Crossover Distance

The spatial resolution distance crossover determines the maximum distance for which paths are generated between MGRAs. Above this distance, a TAZ-based path is used. Increasing the crossover distance improves the accuracy of the logsum, but after a point the MGRA paths and TAZ paths are very similar, and the improvement in accuracy is slight. Increasing the crossover distance also increases run time, which is proportional to the number of origins and the square of the maximum distance used for each spatial resolution. Therefore, we set the crossover distance to 2 miles.

Variance of Random Coefficients

The variance of the random coefficients determines the likelihood that paths with a variety of characteristics will be sampled during alternative generation. If the choice set size is small and held constant, a greater variance in the random coefficients reduces the accuracy of the calculated logsum because of greater simulation error and the sampling of less attractive, less realistic paths. If the choice set size is large or allowed to increase; however, the impact on a greater variance of random coefficients on the accuracy of the logsum is reduced. Furthermore, in this case, a greater variance of random coefficients reduces run time because fewer samples are required to reach the targeted choice set size. Therefore, we set the variance of random coefficients to be quite large. Each random coefficient is sampled from a uniform distribution on (0.3β,1.7β) where β is the coefficient from the route choice utility function.

Variance of Random Edge Cost Multiplier

The variance of the random edge cost multiplier determines the likelihood that multiple paths with the same characteristics will be sampled during alternative generation. The influence of this parameter on the accuracy of the calculated logsum and run time of the algorithm is similar to the influence of the variance of the random coefficients (see above), although the influence on the accuracy of the logsum from increasing this parameter is smaller. A large edge multiplier variance cannot produce unrealistic paths because turn costs are taken into consideration during path generation. Each random edge cost multiplier is sampled from a discrete distribution on {0.1,1.9}.

Target Size of Choice Set

Increasing the target for the total size of alternatives in the choice set, accounting for overlap, in most cases improves the accuracy of the calculated logsum because it more closely approximates the utility of all possible paths between an origin and destination. However, increasing the target choice set size also increases the run time because more samples are required to reach the target size. The magnitude of the effect on run time also depends on the distance between the origin and destination. For short trips, the algorithm cannot generate many different paths because only a few reasonable paths exist. For long trips, the increase in run time required for more samples is greater because the time to perform a shortest path search is proportional to the square of the distance. Also, if the maximum number of samples is restricted to guarantee termination of the algorithm, then increasing the target choice set size can actually worsen the accuracy of the calculated logsum and its change between scenarios because the number of OD pairs for which the target choice set size is not reached will increase. To balance all of these factors, we set the target

TABLE 5.1 TARGET CHOICE SET SIZES

Distance (mi.)	0.0 – 0.5	0.5 – 1.0	1.0 – 2.0	2.0 – 8.0	8.0 – 10.0	10.0 – 20.0
Total Size	1.0	1.5	2.0	6.0	6.0	1.0

Minimum Number of Samples

Increasing the minimum number of samples improves the accuracy of the logsum because it reduces error in the estimation of the path sampling probability correction factor during the bootstrapping path resampling procedure. Increasing the minimum number of samples also increases run time because more path searches are required. We set the minimum number of samples to the following values, depending on the trip distance.

TABLE 5.2 MINIMUM NUMBER OF SAMPLES

Distance (mi.)	0.0 – 0.5	0.5 – 1.0	1.0 – 2.0	2.0 – 8.0	8.0 – 10.0	10.0 – 20.0
Min. # Samples	1	20	20	20	15	1

Maximum Number of Samples

Increasing the maximum number of samples improves the accuracy of the logsum because it reduces the number of origin-destination pairs for which the target choice set size cannot be reached. Increasing the maximum number of samples also increases the run time because more path searches are required for cases where the likelihood of path sampling is much greater for a few paths. We set the maximum number of samples to the following values, depending on the trip distance. Where the maximum number of samples is 1, the path is the single best path, rather than a randomly sampled path.

Distance (mi.)	0.0 – 0.5	0.5 – 1.0	1.0 – 2.0	2.0 – 8.0	8.0 – 10.0	10.0 – 20.0
Max. # Samples	1	100	100	100	100	1

CALIBRATION OF UTILITY FUNCTION PARAMETERS

Because the spatial distribution of bicycle trips output from the mode choice model is still significantly different from the totals of observed traffic counts at screenlines throughout the region after attempts to adjust the mode choice utility function through trial and error, we did not adjust the route choice utility function parameters from their transferred and asserted values. Finding appropriate adjustments to mode choice utility function parameters was more critical to obtaining satisfactory results, and a better match between the total traffic at screenlines in the model and observed data would be required before a reliable adjustment to route choice coefficients could be made based on over- and under-assignment within each screenline.

VALIDATION OF RUN TIME AND CHANGE IN ROUTE CHOICE LOGSUM

To validate the selected alternative generation parameters, we compared the logsums from selected origins before and after the introduction of hypothetical bike lanes and bicycle boulevards in the Uptown neighborhood. Over a series of several runs of the logsum matrix application, the parameters of the path alternative generation algorithm were adjusted until the simulation error resulting from path sampling was reduced to an acceptable amount while balancing the program's run time. The magnitude of the simultation error can be inferred from the number, extent, pattern, and magnitude of decreases in the logsum value in response to the increase in individual path utility resulting from the bicycle facility projects. Maps of the changes in the logsum to all zones from selected zones, in equivalent minutes of travel time, appear in Figure 5.1 through Figure 5.5. While there are still many decreases in the logsum with the final algorithm parameters, the decreases are small, and randomly distributed along an area perpendicular to the facility improvements.

With a stochastic alternative sampling method, some decreases in the logsum will be inevitable, as the calculated logsum is actually a statistical estimate of the true logsum that would be calculated if all alternatives could be fully enumerated.

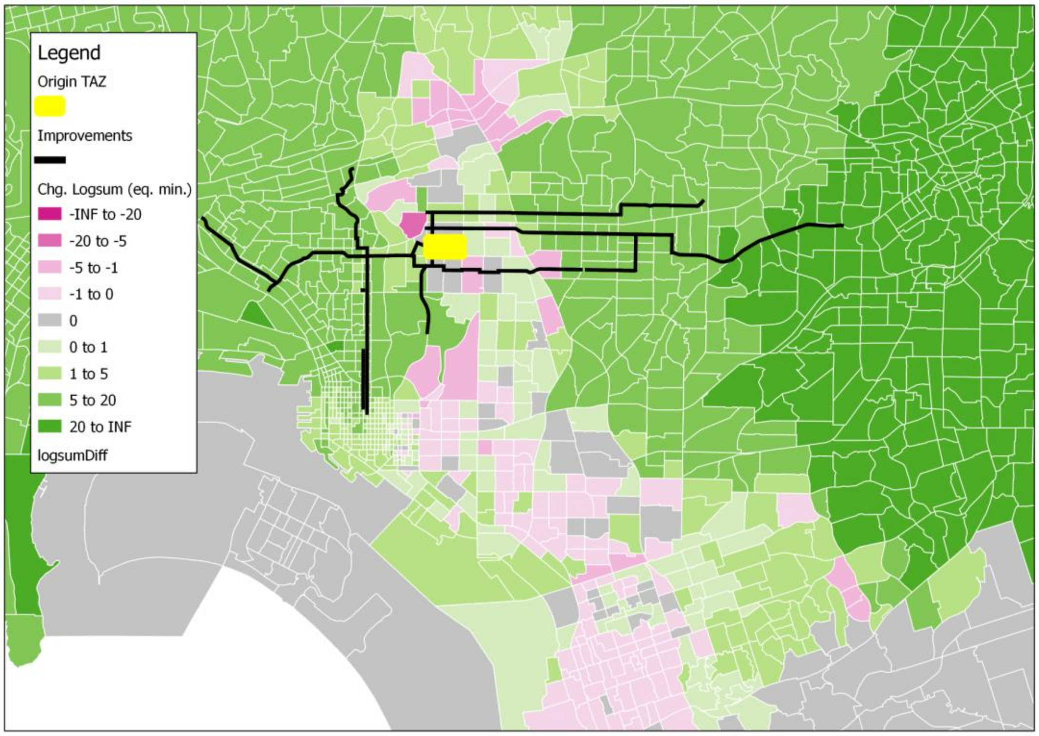
There are actually three reasons why the logsum could go down, which are listed below in order from least problematic to most problematic:

1) The utilities of less-attractive alternatives have increased enough, in the build scenario, so that they replace other unchanged alternatives in the sample of alternatives for the particular random seed chosen, but the utilities of these new alternatives are not actually greater than those of the alternatives they replace. (This will occur sometimes whenever you use importance sampling of alternatives).
2) The estimates of the ratios of alternative sampling probabilities obtained using the bootstrapping procedure I'm using to correct for the unequal sampling probabilities are, for the particular random seed chosen, higher or lower than the true ratios of sampling probabilities. (The end effect of this cause is actually the same as no. 1 in that less attractive alternatives appear in the sample, but the underlying reason is endemic to the fact that the probability sampling correction factors in the bootstrapping procedure are statistical estimates themselves rather than known quantities.)
3) The statistical estimate of the logsum is inconsistent; i.e. the distribution of estimated values obtained for different random seeds does not converge in probability to the true value as the number of alternatives in each sample increases. Because of the resampling procedure, this could actually occur at two levels:
    a. The sampling probabilities estimated using the bootstrapping procedure are themselves inconsistent. This is possible because the procedure employed tries to estimate the sampling probabilities under the assumption that overlapping routes have a greater sampling probability rather than counting only the number of times an exact route is generated.
    b. A route with a non-zero choice probability is sampled with zero probability.

To see how number 1 can occur whenever importance sampling of alternatives is used, consider for simplicity a destination choice model. Suppose, in the build case, for the random seed selected, the procedure happens to sample the N alternatives with the actual best utilities. Then, in the build case, improvements have increased the utility and sampling probability of the alternative with the N+1st best utility slightly, but not enough so that it's utility is better than any of those in the top N. If this slight increase in sampling probability means the lesser alternative is selected instead of one of the top alternatives, then the logsum will go down. This extreme example has been written to illustrate the source of error, and therefore sounds like a rare occurrence, but this type of error is actually really common. If a similar plot of destination choice logsums from CT-RAMP were created for a scenario involving only improvements, several of them would go down as well.

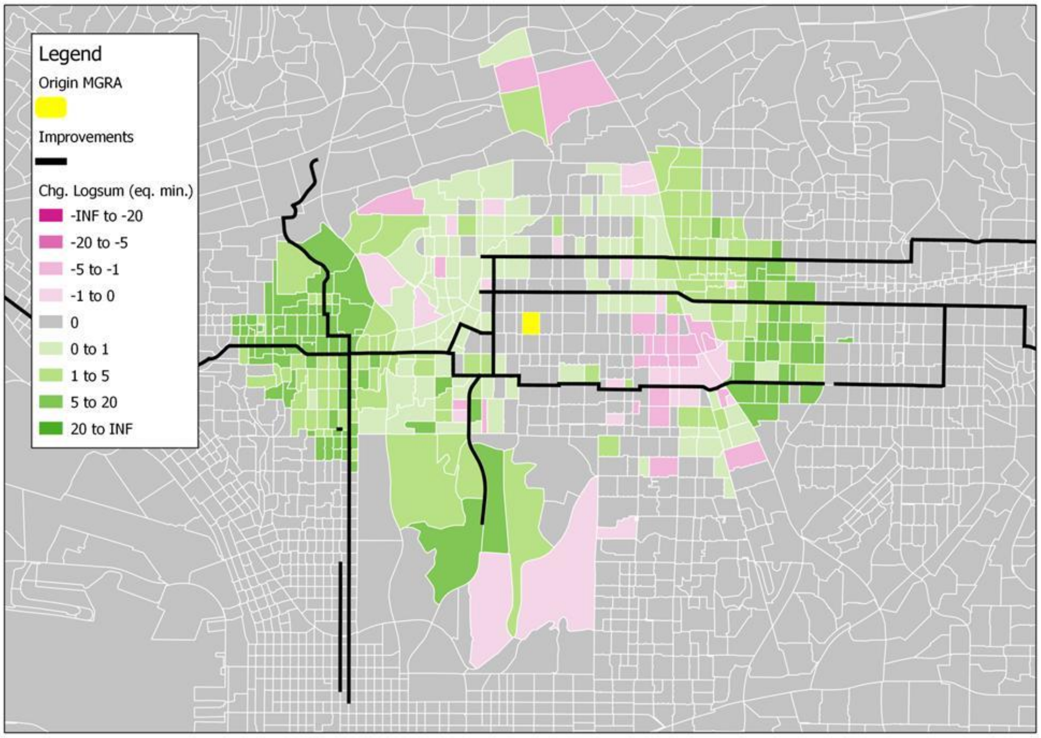
The maps of logsum decreases demonstrates that the likelihood that 3 (logsum inconsistency) is occurring is low. As the sample size is increased, the magnitudes of the logsum decreases get smaller and smaller. The total run time of the logsum matrix application on the XXXXXX machine is XXXXXXX.

FIGURE 5.1 UPTOWN SCENARIO CHANGE IN LOGSUM FROM TAZ 3445



Asdas

FIGURE 5.2 UPTOWN SCENARIO CHANGE IN LOGSUM FROM MGRA 33



Asdasdasdada

FIGURE 5.3 UPTOWN SCENARIO CHANGE IN LOGSUM FROM TAZ 3000

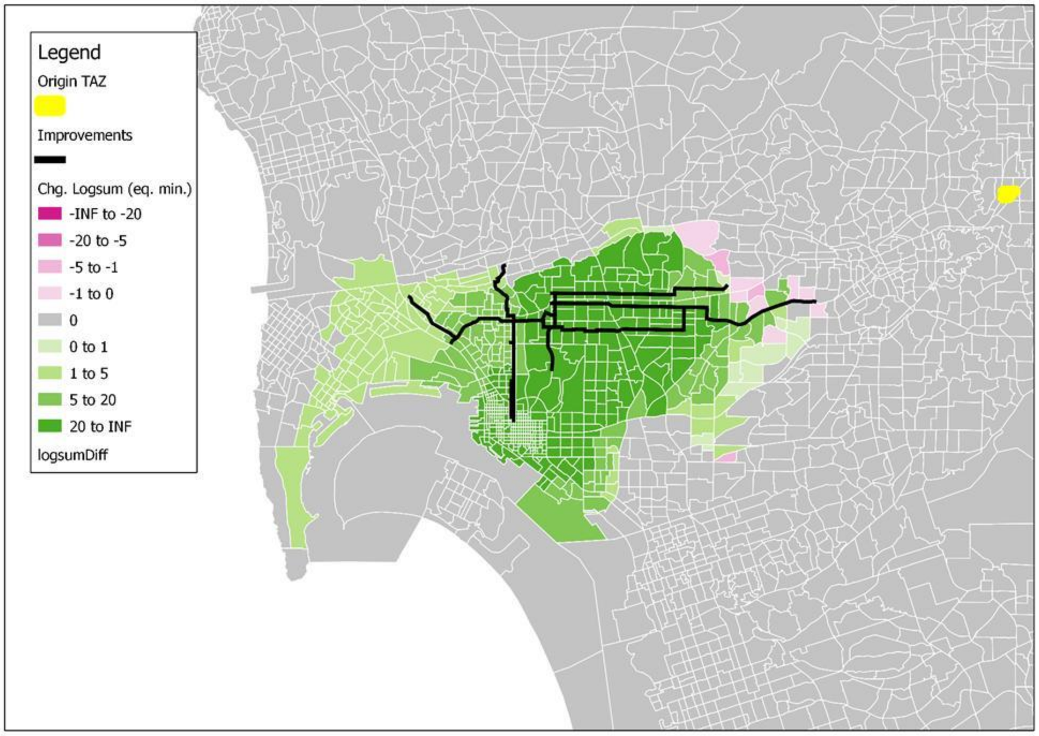
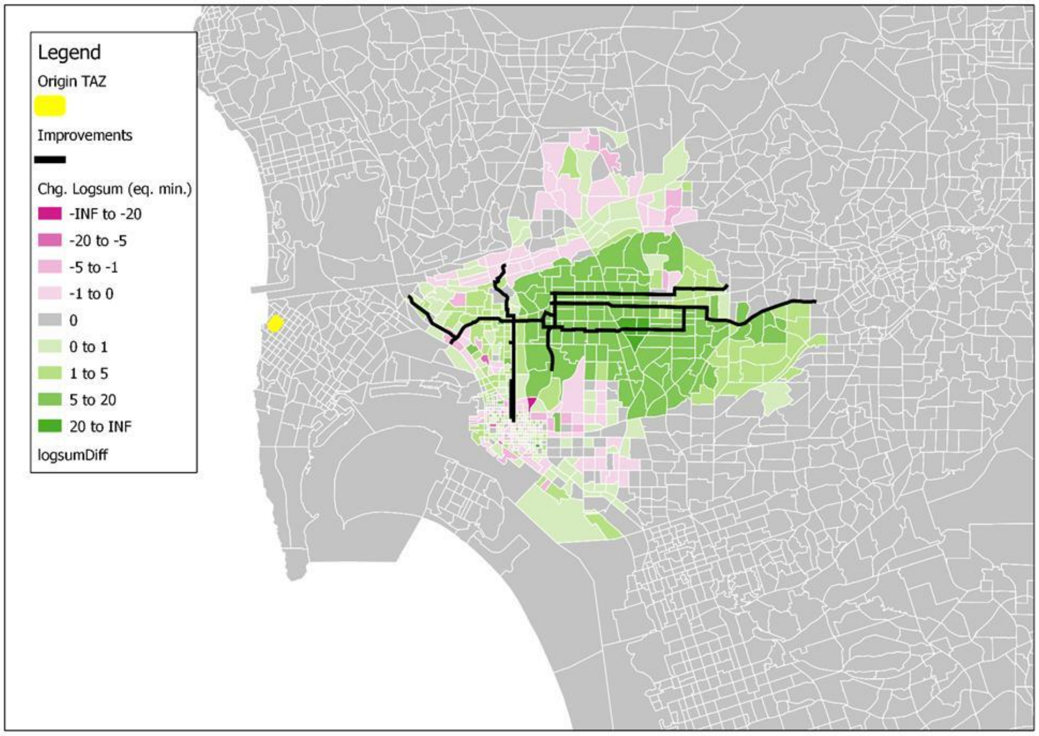
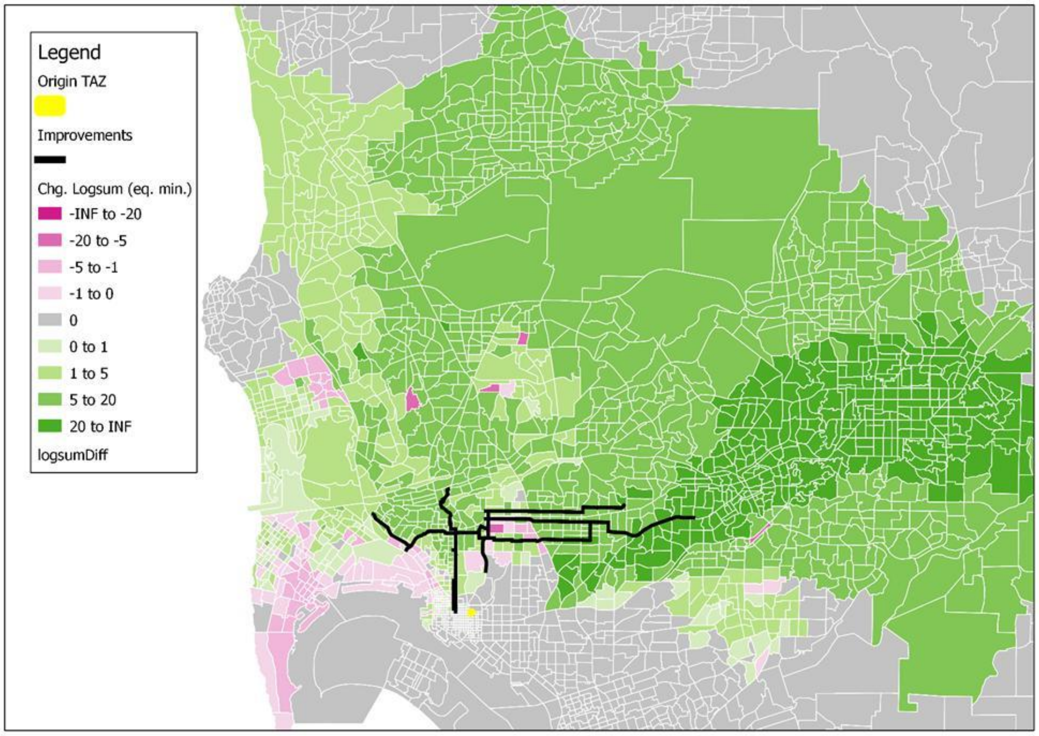


FIGURE 5.4 UPTOWN SCENARIO CHANGE IN LOGSUM FROM TAZ 3500



Asdasdadas

FIGURE 5.5 UPTOWN SCENARIO CHANGE IN LOGSUM FROM TAZ 4000

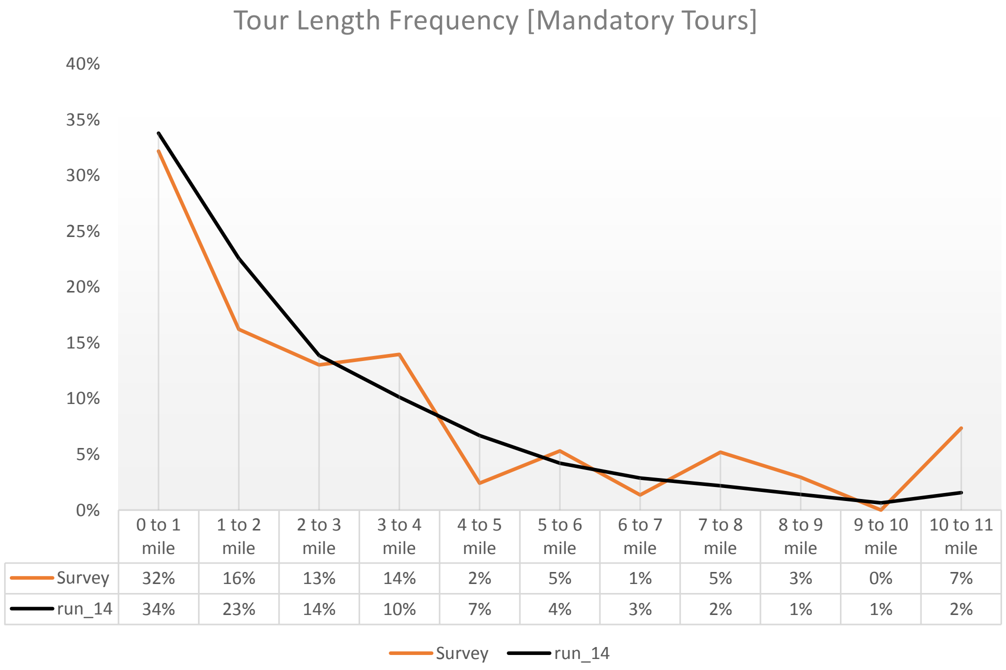


5.2 | TOUR MODE CHOICE CALIBRATION

After implementation of software changes, initial assertion of bicycle logsum coefficients and subsequent re-estimation of the tour mode choice models, the model system was run for 2010 using Series 13 land-use data and compared to mode choice calibration targets created from scaled 2006 San Diego Household Travel Survey data and 2009 San Diego Transit On-board Survey data. Note that there were only 160 bicycle tours in the 2006 San Diego Household Travel Survey. Comparisons were first made at the tour mode choice level, by comparing tours by tour purpose, auto sufficiency, and mode, and alternative-specific constants were adjusted to match observed targets.

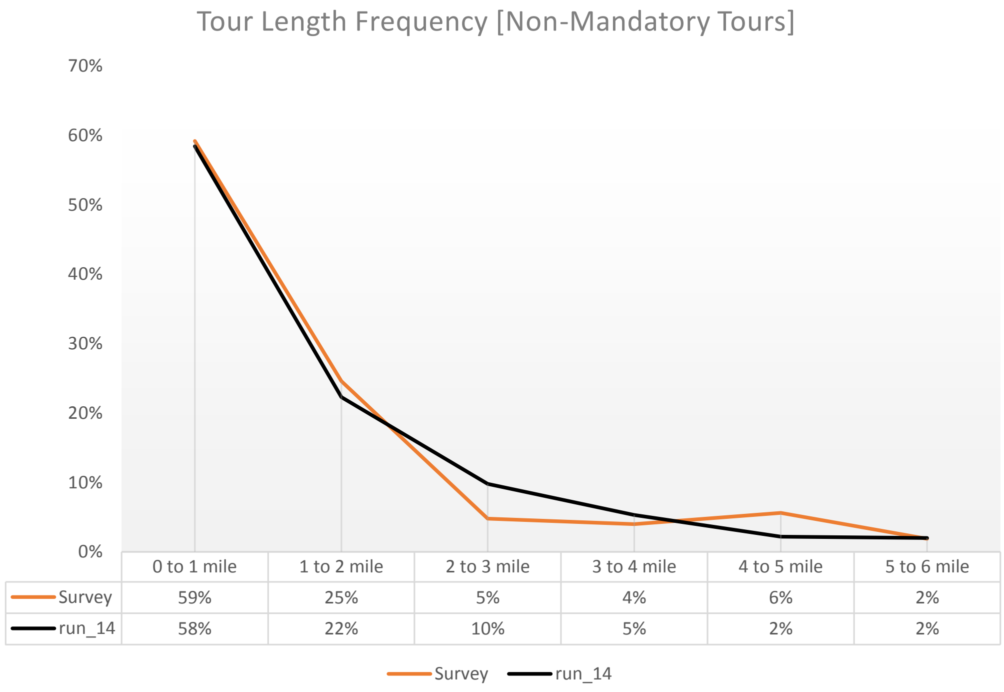
After tour mode constants were adjusted, comparisons were made specifically for bicycle tour length and to ensure that bicycle tours were generally being chosen in the correct districts according to the household survey data. Figure 5.6 shows the estimated versus observed bicycle tour length frequency distribution for mandatory purpose tours. Figure 5.7 shows the estimated versus observed bicycle tour length frequency distribution for non-mandatory tours. Table 5.3 shows the observed and estimated average bicycle tour lengths. The bicycle tour length distribution and average bicycle tour lengths appear to match observed data fairly well, though there are some differences by tour purpose. The initial district comparison indicated that a disproportionate share of bicycle tours were generated to downtown San Diego. It was concluded that use of several land-use variables for the bicycle mode (employment density and household\employment mix) were essentially double-counting bicycle facility benefits already measured by the bicycle logsum term. Therefore these land-use variables were removed from the bicycle utility term.

FIGURE 5.6 MANDATORY BICYCLE TOUR LENGTH FREQUENCY DISTRIBUTION



	0 to 1 mile	1 to 2 mile	2 to 3 mile	3 to 4 mile	4 to 5 mile	5 to 6 mile	6 to 7 mile	7 to 8 mile	8 to 9 mile	9 to 10 mile	10 to 11 mile
Survey	32%	16%	13%	14%	2%	5%	1%	5%	3%	0%	7%
run_14	34%	23%	14%	10%	7%	4%	3%	2%	1%	1%	2%

FIGURE 5.7 NON-MANDATORY BICYCLE TOUR LENGTH FREQUENCY DISTRIBUTION



	0 to 1 mile	1 to 2 mile	2 to 3 mile	3 to 4 mile	4 to 5 mile	5 to 6 mile
Survey	59%	25%	5%	4%	6%	2%
run_14	58%	22%	10%	5%	2%	2%

TABLE 5.3 AVERAGE BICYCLE TOUR LENGTHS

Purpose	Observed	Estimated	Diff	% Diff
Mandatory	3.32	2.51	-0.82	-25%
Non-Mandatory	1.30	1.22	0.08	6%

After assessment of tour mode choice calibration targets and tour length distributions, the SANDAG Activity-Based model was re-run for 2012; this involved creation of a new 2012 all-streets network with 2012 Transit Access Points. The model was re-calibrated to match 2012 scaled targets from the same sources as above, taking into account the removal of land-use variables in the bicycle utility term as described above, and bicycle trips were assigned to the 2012 all-streets network to assess goodness-of-fit against bicycle volumes. This comparison revealed a significant under-estimate of bicycle tours in the screenlines along the California coast. In order to better match bicycle tour counts at these screenlines, a linear term was introduced in the bicycle utility equation for the tour mode choice models as follows:

$$U_{\text{miles to coast, bicycle}} = \text{Beta}_{\text{in-vehicle time}} * (-1.0) * \max(0, 60\text{-}30\text{*miles to coast})$$

The term provides a benefit to the bicycle mode worth 60 equivalent minutes of in-vehicle time for tours with a destination right at the coast, with a decreasing utility effect of 30 minutes per mile and a maximum distance range of two miles until no benefit is given. This benefit greatly improved the match to observed bicycle counts.

Final estimated versus observed tours by mode are shown in Table 5.4.

TABLE 5.4 ESTIMATED VERSUS OBSERVED TOURS BY MODE AND AUTO SUFFICIENCY

	Observed (scaled to estimated)				Estimated			
Tour Mode	**No Veh**	**Veh<Adult**	**Veh>=Adul**	**Total**	**No Veh**	**Veh<Adult**	**Veh>=Adul**	**Total**
Drive-Alone	0	290,241	1,167,807	1,458,048	0	290,030	1,167,005	1,457,035
Shared 2	33,528	344,622	710,371	1,088,521	33,655	341,270	711,625	1,086,550
Shared 3+	22,024	230,917	811,433	1,064,373	22,060	234,210	810,655	1,066,925
Walk	87,747	156,401	156,540	400,689	86,960	156,760	156,615	400,335
Bike	4,821	11,353	26,673	42,847	5,035	11,560	26,965	43,560
Walk-Transit	28,159	42,935	8,477	79,572	28,875	42,895	8,520	80,290
PNR-Transit	333	4,805	3,882	9,020	0	4,795	3,800	8,595
KNR-Transit	882	6,345	780	8,007	955	6,145	795	7,895
School Bus	5,911	22,891	29,317	58,119	5,865	22,845	29,300	58,010
Total	183,405	1,110,510	2,915,280	4,209,195	183,405	1,110,510	2,915,280	4,209,195

	Difference				Percent Difference			
Tour Mode	No Veh	Veh<Adult	Veh>=Adul	Total	No Veh	Veh<Adult	Veh>=Adul	**Total**
Drive-Alone	0	-211	-802	-1,013	0%	0%	0%	0%
Shared 2	127	-3,352	1,254	-1,971	0%	-1%	0%	0%
Shared 3+	36	3,293	-778	2,552	0%	1%	0%	0%
Walk	-787	359	75	-354	-1%	0%	0%	0%
Bike	214	207	292	713	4%	2%	1%	2%
Walk-Transit	716	-40	43	718	3%	0%	1%	1%
PNR-Transit	-333	-10	-82	-425	-100%	0%	-2%	-5%
KNR-Transit	73	-200	15	-112	8%	-3%	2%	-1%
School Bus	-46	-46	-17	-109	-1%	0%	0%	0%
Total	0	0	0	0	0%	0%	0%	0%

5.3 | VALIDATION OF ASSIGNED BICYCLE VOLUMES

To validate the spatial distribution of bicycle trips and the route choice utility function parameters, we compared modeled and observed bicycle volumes by location, time of day, link type, slope, and distance from the coast appear in Table 3 through Table 7. From the screenline totals, it is clear that the spatial distribution of bicycle trips could be improved. There are some significant deviations within each screenline and by link type, but the spatial distribution needs to be improved before validation of these micro-level issues can be performed.

TABLE 5.5 CORDON BICYCLE ASSIGNMENT VALIDATION

Cordon			**Daily Bicycle Volume**			
Intersection	Cross Street 1	Cross Street 2	Observed	Modeled	Diff.	% Diff
Oceanside						
155601	N. Canyon Dr	CA76	19	0	-18	-97%
181021	S Ditmar St	Mission Ave	86	108	22	25%
188361	Michigan Ave	S. Tremont St	82	96	14	18%
Subtotal			187	205	18	10%
Carlsbad						
235051	Jefferson St	Arbuckie Pl	147	249	101	69%
240441	State St	Grand Ave	261	564	303	116%
247721	Carlsbad Ave	Oak Ave	391	249	-142	-36%
Subtotal			799	1,062	263	33%
San Marcos						
262101	Vineyard Rd	Borden Rd	52	25	-27	-53%
276911	E Mission Rd	Campus View Dr	165	57	-108	-66%
1490041	Campus View Dr	Campus Way	115	99	-16	-14%
Subtotal			332	180	-152	-46%
Escondido						
304011	Rock Springs Rd	W Mission Ave	175	293	118	68%
315591	N Centre City Pkwy	W Valley Pkwy	121	107	-14	-11%
321221	S. Juniper St	E 9th Ave	43	5	-38	-89%
Subtotal			338	405	66	20%
Del Mar						
533211	Camino Del Mar	Mar Scenic Dr	523	158	-365	-70%
530541	Mar Scenic Dr	Del Mar Heights Rd	71	171	100	141%
Subtotal			594	330	-264	-44%
Poway / Mira Mesa						
533941	Ragweed St	Park Village Rd	44	12	-32	-74%
539541	Chabola Rd	Mercy Rd	47	32	-14	-30%

Cordon			**Daily Bicycle Volume**			
Intersection	Cross Street 1	Cross Street 2	Observed	Modeled	Diff.	% Diff
543651	Cara Wy	Scripps Poway Pkwy	40	37	-3	-8%
Subtotal			131	81	-50	-38%
Sorrento Valley						
567021	Vista Sorento Pky	Sorrento Valley Rd	108	44	-64	-60%
575691	Sorrento Valley Rd	Sorrento Valley Blvd	338	136	-202	-60%
580311	Vista Sorrento Pky	Mira Misa Blvd	39	57	18	47%
Subtotal			485	237	-248	-51%
La Jolla						
588091	N. Torrey Pines Rd	La Holla Shores Dr	566	1,299	733	129%
589551	Gilman Dr	Lebon Dr	505	208	-297	-59%
593171	Lebon Dr	Nobel Dr	252	654	402	160%
594571	Genesee Ave	Nobel Dr	224	455	231	103%
Subtotal			1,547	2,617	1070	69%
Clairemont Mesa						
672121	Clairemont Dr	Clairemont Mesa Blvd	169	59	-110	-65%
652511	Genesee Ave	Clairemont Mesa Blvd	264	132	-132	-50%
648261	Cadet St	Conrad Ave	62	24	-38	-61%
Subtotal			495	214	-281	-57%
Pacific Beach						
719601	Soledad Rd / Lemon St	Beryl St	205	18	-187	-91%
742031	Ingraham St	Hornblend St	269	98	-171	-63%
760711	Fanuel St	At the Shore (Mission Bay)	266	323	58	22%
Subtotal			740	440	-300	-41%
Kearny Mesa / Serra Mesa						
640871	Kearny Villa Rd/Ruffin Rd	Waxie Way	145	10	-135	-93%
719731	W/ Canyon Ave	Aero Dr	38	111	73	191%
729661	Sandrock Rd	Murray Ridge Rd	143	89	-54	-38%
Subtotal			326	210	-116	-36%
SDSU						
813951	Collwood Blvd	Montezuma Rd	113	501	388	343%
812441	E Campus Dr	Montezuma Rd	1,083	1,002	-80	-7%
797581	Colleg Ave	East Campus Dr	25	136	111	454%
Subtotal			1,220	1,639	419	34%
Uptown						
843141	30th St	Monroe Ave	167	39	-128	-77%
848101	Utah St	Meade Ave	256	287	31	12%

Cordon			Daily Bicycle Volume			
Intersection	Cross Street 1	Cross Street 2	Observed	Modeled	Diff.	% Diff
854311	Oregon St	El Cajon Blvd	165	93	-72	-43%
858571	Hamilton St	Howard Ave	102	118	16	16%
864581	Texas St	Polk Ave	153	68	-85	-56%
875811	Mississippi St	University Ave	186	176	-10	-5%
875821	Mississippi St	University Ave	343	216	-127	-37%
890341	Florida St	Cypress Ave	65	84	19	29%
897731	Park Blvd	Myrtle Ave	427	114	-313	-73%
Subtotal			1,864	1,195	-669	-36%
Downtown						
954311	India St	W Cedar St	137	154	17	12%
959171	Union St	W Ash St	192	200	8	4%
959191	1st Ave	W Ash St	189	169	-20	-10%
966391	4th Ave	C St	243	502	259	107%
976271	8th Ave	G St	216	177	-39	-18%
978621	10th Ave	Market St	276	112	-164	-59%
Subtotal			1,253	1,315	62	5%
La Mesa						
818841	La Mesa Blvd	El Cajon Blvd	112	263	151	135%
834891	Lee Ave	University Ave	91	46	-44	-49%
834981	Lee Ave	University Ave	45	35	-10	-22%
818261	Randell Dr	La Mesa Blvd	73	192	119	164%
Subtotal			321	537	216	67%
Chula Vista						
1072401	2nd Ave	C St	86	92	6	8%
1088881	4th Ave	F St	95	264	169	178%
1103691	Woodlawn Ave	H St	30	31	2	7%
1103731	Woodlawn Ave	H St	39	71	32	84%
1111891	Broadway	W J St	211	206	-5	-2%
Subtotal			460	664	205	45%
San Ysidro						
1170191	Alverson Rd	W San Ysidro Blvd	68	99	32	47%
1169271	Alaquinas Dr	Beyer Blvd	108	109	1	1%
Subtotal			176	209	33	19%
Total			11,266	11,539	273	2%

TABLE 5.6 TIME PERIOD BICYCLE ASSIGNMENT VALIDATION

Time Period	Observed	Modeled	Diff	% Diff
EA	337	197	-140	-41%

AM	2564	2697	133	5%
MD	5050	4233	-816	-16%
PM	3639	3520	-119	-3%
EV	1549	1764	215	14%
Daily	13139	12412	-727	-6%

TABLE 5.7 FACILITY TYPE BICYCLE ASSIGNMENT VALIDATION

Facility Type	**Observed**	**Modeled**	**Diff**	**% Diff**
Bike Path	129	275	145	112%
Bike Lane	4701	4884	183	4%
Bike Route	2452	2716	264	11%
Major Arterial	126	97	-29	-23%
Minor Arterial	2125	1316	-810	-38%
Other	3605	3125	-480	-13%
Total	13139	12412	-727	-6%

TABLE 5.8 SLOPE BICYCLE ASSIGNMENT VALIDATION

Slope Bin	**Observed**	**Modeled**	**Diff**	**% Diff**
Less than 2%	10619	10234	-386	-4%
2% to 4%	1486	1250	-236	-16%
4% to 6%	690	713	23	3%
6% and up	342	215	-127	-37%
Total	13139	12412	-727	-6%

TABLE 5.9 DISTANCE TO OCEAN BICYCLE ASSIGNMENT VALIDATION

Distance to Ocean	Observed	Modeled	Diff	% Diff
0-1 miles	2898	3427	529	18%
1-2 miles	1191	951	-240	-20%
2-3 miles	1584	1929	345	22%
> 3 miles	5593	5231	-362	-6%
Total	11266	11539	273	2%

6.0 SENSITIVITY TESTING

In addition to calibrating and validating the model to observed base year conditions, it is also necessary to demonstrate the sensitivity of the enhanced ABM to active transportation improvements. Two sensitivity tests were performed. The first sensitivity test involved adding an extensive set of bicycle path, "cycle track" and "bicycle boulevard" improvements across a wide extent of the Uptown corridor in the region's core. The second sensitivity test involved adding a bicycle path along a segment of a rail corridor on the coast. The expectation is that by making improvements to bicycle infrastructure the overall number of bicycle trips will increase, and that there will be increased bicycle volumes on the facilities that have received bicycle improvements. The following sections document the results of these two sensitivity tests.

6.1 | UPTOWN CORRIDOR

Figure 6.1 illustrates the location of the proposed improvements in the Uptown corridor. The all streets-based active transportation network was updated to reflect the different types of improvements along the corridor. A full run, including feedback, of SANDAG's ABM was then performed using these inputs, and the results compared to a baseline or "no build" alternative.

FIGURE 6.1 UPTOWN CORRIDOR

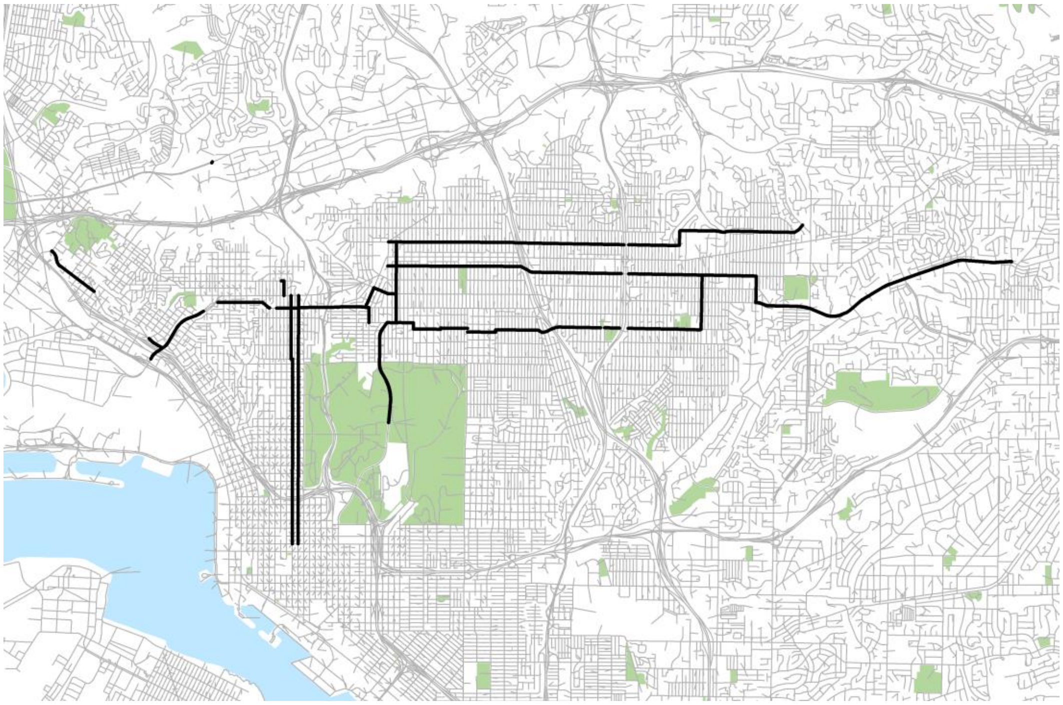


Table 6.1 summarizes the changes in trips by mode for the region. The proposed uptown improvements in an increase of approximately 1.3% of bike trips. Declines in auto trips and

pedestrian trips are observed, as well as an increase in transit trips. However, in relative terms, the changes in trips by non-bicycle modes is small.

TABLE 6.1 UPTOWN REGIONAL MODE CHANGES

	Base	Uptown	Diff	% Diff
Bike	103,986	105,353	1,367	1.3%
Ped	966,878	966,613	-265	0.0%
Transit	181,857	182,109	252	0.1%
Auto	8,452,932	8,451,596	-1,336	0.0%

Figure 6.2 illustrates the changes in bicycle trips by TAZ. Green areas indicate where bicycle trips increase, while red areas indicate where trips decreased. This figure demonstrates that the improvements generally led to increases in bicycle travel in the corridor, although a limited number of TAZs did show declines in bicycle trips. Because of the nature of this test, in which the entire model system was run with feedback, there may be a number of potential explanations for these declines. For example, the bicycle logsum accessibility calculation may lead to slight declines in accessibility, as described earlier in this document. Simulation variation and the associated changes in activity generation, destination choice, and mode choice may also be producing these outcomes.

FIGURE 6.2 UPTOWN CHANGE IN BIKE TRIPS BY TAZ

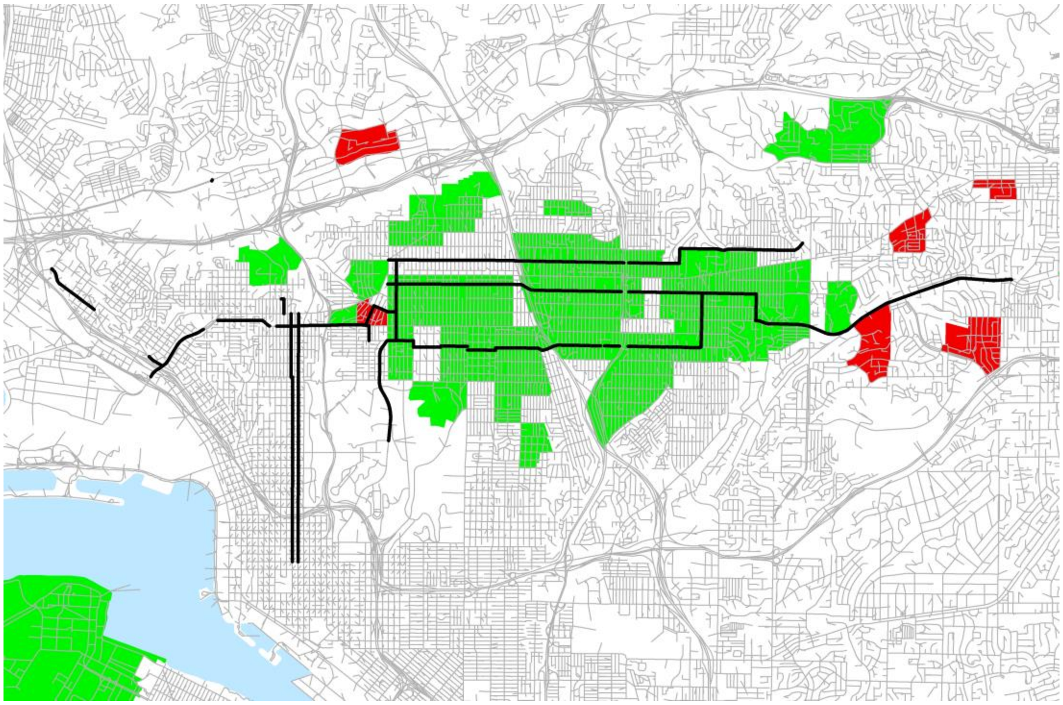


Table 6.2 summarizes the changes in daily bicycle volumes for all streets that intersect with a cordon across the corridor. Highlighted rows indicate streets on which improvements were made. The table shows that overall volumes on the cordon increased by over 60%, and that the greatest increases were on the facilities to which bicycle improvements were made.

TABLE 6.2 UPTOWN BICYCLE CORDON VOLUMES BY STREET

STREET	BASE VOL	UPTOWN VOL	DIFF	% DIFF
ADAMS AV	5	4	-2	-29.3%
ALLEY	6	7	1	10.5%
MADISON AV	9	6	-3	-34.2%
MONROE AV	85	69	-15	-18.3%
MEADE AV	**88**	**59**	**-29**	**-33.2%**
ALLEY	6	5	-2	-23.7%
EL CAJON BL	12	3	-9	-77.5%
ORANGE AV	**314**	**574**	**260**	**83.0%**
POLK AV	69	42	-26	-38.6%
UNIVERSITY AV	30	15	-15	-50.4%
WIGHTMAN ST	33	52	19	56.9%
LANDIS ST	**5**	**240**	**235**	**5020.4%**
DWIGHT ST	1	10	9	1300.6%
MYRTLE AV	1	3	2	374.2%
TOTAL	663	1,088	424	64.0%

A summary of "bicycle miles travelled" derived by the bicycle assignment is shown in Table 6.3. This table indicates that the Uptown improvements results in a 2.3% increase in regional BMT, and a 16% increase in BMT in the Uptown corridor, which is defined as all TAZs whose centroid are within 0.5 miles of any Uptown bicycle improvement.

TABLE 6.3 UPTOWN CHANGES IN BICYCLE MILES TRAVELLED

	Base	Uptown	Diff	% Diff
Regional	215,811	220,741	4,930	2.3%
Corridor	11,885	13,789	1,903	16.0%

6.2 | COASTAL RAIL TRAIL

Figure 6.3 illustrates the location of the proposed improvements in the Coastal Rail Trail corridor. As with the Uptown test, the all streets-based active transportation network was updated to reflect the different types of improvements along the corridor. A full run, including feedback, of SANDAG's ABM was then performed using these inputs, and the results compared to a baseline or "no build" alternative.

FIGURE 6.3 COASTAL RAIL TRAIL CORRIDOR

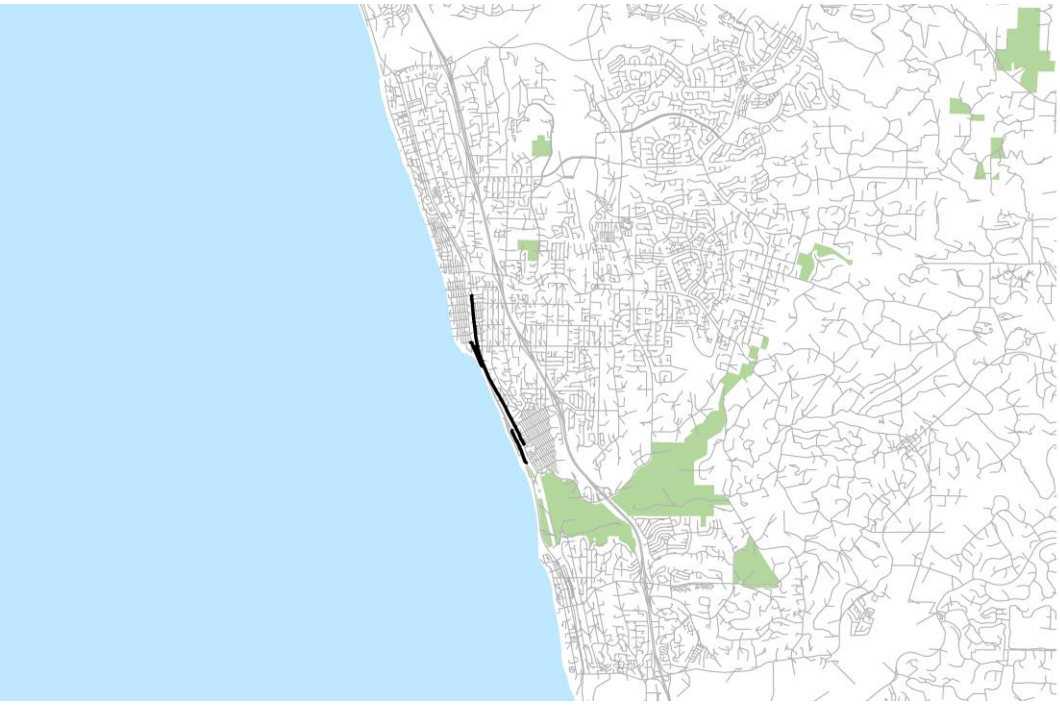


Table 6.4 summarizes the changes in trips by mode for the region. The proposed Coastal Rail Trail improvements result in an increase of approximately 0.3% of bike trips. Declines in auto trips observed, as well as an increases in pedestriantransit trips. However, as with the Uptown test, in relative terms the changes in trips by non-bicycle modes is small.

TABLE 6.4 COASTAL RAIL TRAIL REGIONAL MODE CHANGES

	Base	Coastal	Diff	% Diff
Bike	103,986	104,299	313	0.3%
Ped	966,878	968,031	1,153	0.1%
Transit	181,857	182,072	215	0.1%
Auto	8,452,932	8,450,307	-2,625	0.0%

Figure 6.4 illustrates the changes in bicycle trips by TAZ. Green areas indicate where bicycle trips increase, while red areas indicate where trips decreased. This figure demonstrates that the improvements generally had a limited effect on bicycle trip making the project area.

FIGURE 6.4 COASTAL RAIL TRAIL CHANGE IN BIKE TRIPS BY TAZ

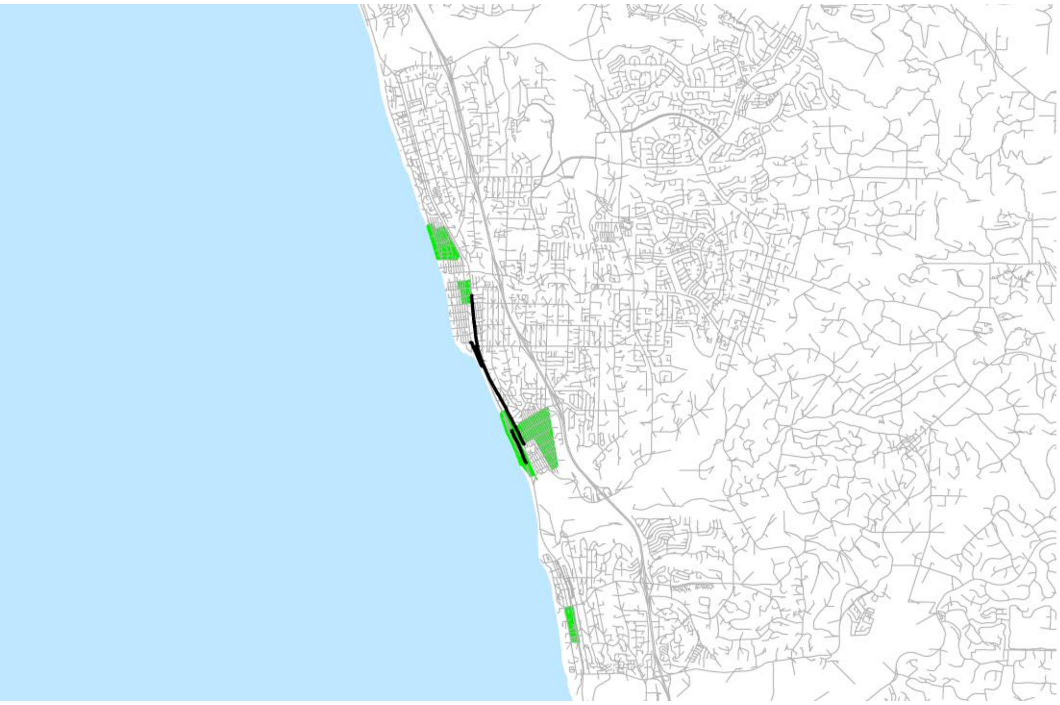


Table 6.5 summarizes the changes in daily bicycle volumes for all streets that intersect with a cordon across the corridor. Highlighted rows indicate streets on which improvements were made. The table shows that overall volumes on the cordon increased by over 40%, and that the greatest increases were on the facilities to which bicycle improvements were made.

TABLE 6.5 COASTAL RAIL TRAIL BICYCLE CORDON VOLUMES BY STREET

STREET	BASE VOL	COASTAL VOL	DIFF	% DIFF
SAN DIEGUITO DR	79	28	-52	-65.0%
ARDEN DR	26	15	-11	-43.2%
CORNISH DR	23	15	-9	-36.6%
ALLEY	4	2	-2	-43.1%
DEWITT AV	3	1	-2	-62.2%
ALLEY	3	1	-2	-59.5%
S VULCAN AV	84	25	-59	-70.5%
S COAST HIGHWAY 101	6	6	0	-1.7%

ALLEY	16	16	0	2.2%
02ND ST	74	52	-23	-30.4%
ALLEY	17	20	4	21.7%
03RD ST	22	26	4	17.0%
SEALANE DR	4	1	-4	-81.7%
RAIL TRAIL	**0**	**299**	**299**	**100.0%**
TOTAL	361	506	145	40.1%

A summary of "bicycle miles travelled" derived by the bicycle assignment is shown in Table 6.6 This table indicates that the Coastal Rail Trail improvements result in a 0.5% increase in regional BMT, and a 17% increase in BMT in the corridor, which is defined as all TAZs whose centroid are within 0.5 miles of any Coastal Rail Trail bicycle improvement.

TABLE 6.6 COASTAL RAIL TRAIL CHANGES IN BICYCLE MILES TRAVELLED

	Base	**Coastal**	**Diff**	**% Diff**
Regional	215,811	216,890	1,078	0.5%
Corridor	1,536	1,803	267	17.4%

7.0 MODEL APPLICATION

Coding Projects

7.1 | RUNNING THE AT-ENHANCED ABM

JAVA PROPERTIES

This section provides descriptions of the functions of and instructions for setting new fields in the Java properties file required to run the AT-Enhanced ABM

```
active.node.file = input/SANDAG_Bike_NODE.dbf
```

- Determine which DBF file the model reads to obtain data on nodes in the active transportation network.

```
active.node.id = NodeLev_ID
```

- Determines which column in the node file is the unique identifier

```
active.node.fieldnames = mgra,taz,x,y,tap,signalized
active.node.columns = MGRA,TAZ,XCOORD,YCOORD,TAP,Signal
```

- Together, determine mapping between java SandagBikeNode object fields and node DBF file columns

```
active.edge.file = input/SANDAG_Bike_NET.dbf
```

- Determines which DBF file the model reads to obtain data on edges in the active transportation network.

```
active.edge.anode = A
active.edge.bnode = B
```

- Determine which columns in the edge DBF contain foreign keys to the unique identifiers of the first and second nodes in the edges.

```
active.edge.directional = false
```

- Determines whether the edge DBF file contains separate records for edges in the AB and BA directions.

```
active.edge.fieldnames =
       functionalClass,distance,gain,bikeClass,lanes,cycleTrack,bikeBlvd

active.edge.columns.ab =
       Func_Class,Distance,AB_Gain,ABBikeClas,AB_Lanes,Bike2Sep,Bike3Blvd

active.edge.columns.ba =
       Func_Class,Distance,BA_Gain,BABikeClas,BA_Lanes,Bike2Sep,Bike3Blvd
```

- Together, determine the mapping between java SandagBikeEdge object fields and edge DBF file columns for AB and BA direction

```
active.edge.centroid.field = functionalClass

active.edge.centroid.value = 10
```

- Together, determine which SandagBikeEdge field can be queried to determine if an edge is a centroid connector, and which value corresponds to centroid connectors

```
active.edge.autospermitted.field = functionalClass

active.edge.autospermitted.values = 1, 2, 3, 4, 5, 6, 7
```

- Together, determine which SandagBikeEdge field can be queried to determine if an edge is shared by motorized traffic, and which values allow auto travel

```
active.sample.distance.breaks = 0.5, 1.0, 2.0, 5.0,  8, 10, 99

active.sample.pathsizes =       1.0, 1.5,   2,   6,  6,  6,  1

active.sample.count.min =         1,  20,  20,  20, 20, 15,  1

active.sample.count.max =         1, 100, 100, 100,100,100,  1
```

- Together, customize path alternative sampling algorithm. distance.breaks determines the upper boundary of distance intervals for which the parameters below apply, pathsizes determines the target choice set size, accounting for path overlap, count.min determines the minimum sample count, and count.max determines the maximum sample count.

```
active.sample.random.scale.coef = 0.7
```

- Determines the variance of the random coefficients in path generation. Random coefficients are sampled from a uniform distribution on ((1-0.7)β, (1+0.7)β) where β is the utility function coefficient (given by active.coef below).

```
active.sample.random.scale.link = 0.9
```

- Determines the variance of the random link cost multiplier in path generation. After link costs are calculated using the random coefficients, the link cost is multipled by a discrete distribution on { (1-0.9)β, (1+0.9) }.

```
active.sample.random.seeded = true
```

- Determines whether random link costs should be seeded. True will cause results to be reproducible, while false can be used to evaluate simulation error.

```
active.sample.maxcost = 998
```

- Determines maximum cost in path search. For any node which is reachable only by a path that exceeds this cost, the path search will not consider paths extending from this node.

```
active.maxdist.bike.taz = 20.0

active.maxdist.bike.mgra = 2.0

active.maxdist.walk.mgra = 3.0

active.maxdist.walk.tap = 1.0
```

- Determines maximum distance of bike TAZ-TAZ trips, bike MGRA-MGRA trips, walk MGRA-MGRA trips, and walk MGRA-TAP segments in miles.

```
active.output.bike = output/

active.output.walk = output/
```

- Determines output directory for writing of bike and walk logsum matrices, network assignments, and path traces.

```
active.coef.distcla0 =  0.858

active.coef.distcla1 =  0.248

active.coef.distcla2 =  0.544

active.coef.distcla3 =  0.773

active.coef.dartne2  =  1.050

active.coef.dwrongwy =  3.445

active.coef.dcyctrac =  0.424
active.coef.dbikblvd =  0.343

active.coef.gain     =  0.015

active.coef.turn     =  0.083

active.coef.signals  =  0.040

active.coef.unlfrma  =  0.360

active.coef.unlfrmi  =  0.150

active.coef.untoma   =  0.480

active.coef.untomi   =  0.100
```

- Determine average of random coefficients in bicycle path generation for distance on ordinary streets (in miles), distance on Class I facilities, distance on Class II facilities, distance on Class III facilities, distance on arterials without bike lanes, distance traveling the wrong way, distance on cycle tracks, distance on bike boulevards, elevation gain (in feet), the number of turns, the number of signals (excluding right turns and through junctions), the number of un-signalized left turns from major arterials, the number of un-signalized left turns from minor arterials, the number of un-signalized crossings of major arterials, and the number of un-signalized crossings of minor arterials.

```
active.coef.distance.walk = 20.0

active.coef.gain.walk = 0.067
```

- Determine walk path generalized cost coefficients for distance in miles, and elevation gain in feet.

```
active.walk.minutes.per.mile = 20

active.bike.minutes.per.mile = 6
```

- Determinesinverse speed of walking and biking for estimation of actual time skims.

```
active.trace.origins.taz = 500, 1000

active.trace.origins.mgra = 1000, 2000

active.trace.origins.tap = 1, 3

active.trace.exclusive =  false
```

- Determine the origin TAZs, MGRAs, and TAPs for which model will trace results of path generation and output node sequences to the disk. If exclusive is true, the model will only run for these origins. If false, the model will run for all origins.

```
active.debug.origin = 200003500

active.debug.destination = 200003601
```

- Determine the origin and destination node ids for which the model will trace the results of the bicycle path choice UEC calculations.

```
path.choice.uec.spreadsheet = uec/BikeTripPathChoice.xls

path.choice.uec.model.sheet = 1

path.choice.uec.data.sheet = 0
```

- Determines the file location, model, and data tabs of the bicycle path choice UEC Excel workbook.

```
btpc.alts.file = uec/bike_path_alts.csv
```

- Determines the location of the file listing the numbers of the bicycle path choice alternatives.

```
active.logsum.matrix.file.bike.taz = bikeTazLogsum.csv

active.logsum.matrix.file.bike.mgra = bikeMgraLogsum.csv

active.logsum.matrix.file.walk.mgra = walkMgraEquivMinutes.csv

active.logsum.matrix.file.walk.mgratap = walkMgraTapEquivMinutes.csv
```

- Determine the ouput files for the bike TAZ logsum matrix, bike MGRA logsum matrix, walk MGRA-MGRA logsum matrix, and walk MGRA-TAP logsum matrix.

```
active.bike.write.derived.network = true

active.bike.derived.network.edges = derivedBikeEdges.csv

active.bike.derived.network.nodes = derivedBikeNodes.csv

active.bike.derived.network.traversals = derivedBikeTraversals.csv
```

- Determine whether and to which file the edge, node, and traversal attributes calculated internally in Java should be written for debugging purposes.

```
active.assignment.file.bike = bikeAssignmentResults.csv
```

- Determines to which file the results of the bicycle network assignment should be written.