

5-11-2013

Advanced Techniques In Emergency Preparedness And Geoprocessing

Dean P. Chauvin Jr

Graduate Student, dean.chauvin@gmail.com

Recommended Citation

Chauvin, Dean P. Jr, "Advanced Techniques In Emergency Preparedness And Geoprocessing" (2013). *Master's Theses*. 402.
http://digitalcommons.uconn.edu/gs_theses/402

This work is brought to you for free and open access by the University of Connecticut Graduate School at DigitalCommons@UConn. It has been accepted for inclusion in Master's Theses by an authorized administrator of DigitalCommons@UConn. For more information, please contact digitalcommons@uconn.edu.

Advanced Techniques in Emergency Preparedness and Geoprocessing

Dean Paul Chauvin, Jr.

B.S., Tennessee Technological University, 2003

A Thesis

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of the Arts

At the

University of Connecticut

2013

APPROVAL PAGE

Master of Arts Thesis

Advanced Techniques in Emergency Preparedness and Geoprocessing

Presented by

Dean Paul Chauvin, Jr., B.S.

Major Advisor _____
Jeffrey P Osleeb, Ph.D.

Associate Advisor _____
Robert Cromley, Ph.D

Associate Advisor _____
Daniel Civco, Ph.D.

University of Connecticut

2013

Acknowledgements

I would like to thank Dr. Jeff Osleeb, Dr. Bob Cromley, and Dr. Dan Civco in assisting me with the completion of this work. My wife Nicole and my children Adele and Jason sacrificed so I had the time and resources to achieve this important goal. To them, I am forever grateful.

Table of Contents

1	INTRODUCTION.....	1
2	LITERATURE REVIEW.....	3
2.1	Hurricanes.....	3
2.2	New England and Hurricanes.....	3
2.3	State of Connecticut’s Current Evacuation Plan.....	6
2.4	Evacuation Modeling.....	10
2.5	Flood Modeling.....	15
3	STUDY AREA.....	17
3.1	Introduction.....	17
3.2	Study Area for the Road Intersection Flood Analysis Tool.....	17
3.3	Study Area for the Hurricane Evacuation Zone Tool.....	18
4	DATA AND METHODOLOGY.....	23
4.1	Introduction.....	23
4.2	Methodology.....	23
4.3	Python.....	24
4.4	The Road Intersection Flood Analysis Tool.....	27
4.5	Required Data for the Road Intersection Flood Analysis Tool....	27
4.6	Script Implementation for the Road Intersection Flood Analysis Tool.....	28

4.7	The Road Intersection Flood Analysis Tool Test Application.....	29
4.8	The Hurricane Evacuation Zone Tool.....	33
4.9	Required Data for the Hurricane Evacuation Zone Tool.....	34
4.10	Script Implementation for the Hurricane Evacuation Zone Tool..	34
4.11	The Hurricane Evacuation Zone Test Application.....	35
4.12	Software Standards.....	37
4.13	Road Network Data.....	39
4.14	SLOSH Flood Data.....	40
4.15	LIDAR Elevation Data.....	42
4.16	Geographic Base Files.....	43
4.17	Summary.....	43
5	DISCUSSION.....	45
5.1	Introduction.....	45
5.2	The Road Intersection Flood Tool Case Study Results.....	45
5.3	The Hurricane Evacuation Zone Tool Case Study Results.....	51
5.4	Summary.....	65
6	CONCLUSION.....	67
6.1	Review of Study.....	67
6.2	The Road Intersection Flood Tool.....	67
6.3	Hurricane Evacuation Zone Tool.....	68
6.4	Future Studies.....	69

6.5	Future Studies: Road Intersection Flood Analysis Tool.....	71
6.6	Future Studies: Hurricane Evacuation Zone Tool.....	72
LITERATURE CITED.....		74
APPENDIX A: ROAD INTERSECTION FLOOD ANALYSIS TOOL		
PYTHON SCRIPT.....		79
APPENDIX B: HURRICANE EVACUATION ZONE TOOL PYTHON		
SCRIPT.....		83
APPENDIX C: THE ROAD INTERSECTION FLOOD ANALYSIS TOOL		
TECHNICAL DESCRIPTION.....		91
APPENDIX D: THE HURRICANE EVACUATION ZONE TOOL		
TECHNICAL DESCRIPTION.....		95

LIST OF FIGURES

Figure 1	Study Area: Road Intersection Flood Analysis Tool.....	18
Figure 2	Study Area: Hurricane Evacuation Zone Tool.....	19
Figure 3	Major Rivers within Study Area.....	20
Figure 4	History of Tropical Storm Activity with 50Miles of Study Area Since 1851.....	21
Figure 5	Special Facilities within Study Area.....	22
Figure 6	Schematic of how Python Accesses Geoprocessing Tools in ArcGIS.....	26
Figure 7	Example of a Python Geoprocessing Command.....	26
Figure 8	ArcToolbox Interface for Road Intersection Flood Analysis Tool.....	29
Figure 9	Area of Interest for Road Intersection Flood Analysis Tool.....	31
Figure 10	SLOSH Polygons On Area of Interest for Road Intersection Flood Analysis Tool.....	32
Figure 11	All Road Intersections Created By Road Intersections Flood Analysis Tool.....	33
Figure 12	ArcToolbox Interface for Hurricane Evacuation Zone Tool.....	35
Figure 13	Candidate Hurricane Evacuation Zones.....	36
Figure 14	PythonWin User Interface.....	38
Figure 15	Road Network within Study Area.....	40
Figure 16	SLOSH Data within Study Area.....	41
Figure 17	LIDAR Elevation Data within Study Area.....	42

Figure 18	Intersections Above Water After Category 1 or Category 2 Hurricanes.....	46
Figure 19	Intersections Above Water After Category 3 Hurricane.....	48
Figure 20	Dry Intersections After Category 4 Hurricane.....	49
Figure 21	Composite Results of the Road Intersection Flood Analysis Flood Tool.....	50
Figure 22	All 300 Possible Hurricane Evacuation Zones.....	52
Figure 23	Potential Hurricane Evacuation Zones After 25 Iterations.....	53
Figure 24	Potential Hurricane Evacuation Zones After 50 Iterations.....	54
Figure 25	Potential Hurricane Evacuation Zones After 75 Iterations.....	55
Figure 26	Potential Hurricane Evacuation Zones After 100 Iterations.....	56
Figure 27	Potential Hurricane Evacuation Zones After 125 Iterations.....	57
Figure 28	Potential Hurricane Evacuation Zones After 150 Iterations.....	58
Figure 29	Potential Hurricane Evacuation Zones After 175 Iterations.....	59
Figure 30	Potential Hurricane Evacuation Zones After 200 Iterations.....	60
Figure 31	Potential Hurricane Evacuation Zones After 225 Iterations.....	61
Figure 32	Potential Hurricane Evacuation Zones After 250 Iterations.....	62
Figure 33	Elevation's Average Joint Standard Deviation Per Iteration for Average Elevation.....	63
Figure 34	Time Per Iteration.....	65

LIST OF TABLES

Table 1	The Saffir-Simpson Scale.....	4
Table 2	Estimated Populations At Special Facilities within Study Area.....	22
Table 3	Sample Statistics From Output Log.....	62

LIST OF EQUATIONS

Equation 1	Estimated Number of Evacuating Vehicles.....	11
------------	--	----

CHAPTER 1: INTRODUCTION

The 2005 Atlantic hurricane season was the most active hurricane season in recorded history. The storms spawned during this time set many records that challenged the will and perseverance of the people of the United States. Most notably, Hurricane Katrina decimated parts of the central Gulf coast including metropolitan New Orleans, Louisiana.

Recent advances in Geographic Information Systems (GIS) have played a major role in mitigating the results of hurricanes. Recent studies using a GIS have investigated storm intensities (Bettinger et al., 2009), natural resource management during natural disasters including hurricanes, (Assilzadeh and Gao, 2009), and even economic impacts of hurricanes based on GIS models (Jarmin and Miranda, 2009).

When combining the power of a GIS with the flexibility of a compatible scripting language, previously tedious and complex geoprocessing tasks become powerful tools to study spatial phenomena such as the potential impacts of hurricanes and ways to mitigate their impact. For example, the ability to use the Python scripting language to automate complex geospatial calculations provides new opportunities for in-depth analyses. A script could be written to extract geospatial information from a dataset and process the information using models, advanced statistical methods, and other analyses that are not possible in the normal GIS environment. These scripts can produce a report on the findings, or even output new geospatial datasets. The power and convenience of pairing a scripting language with a GIS is an invaluable research tool.

The goal of this research has been to develop a set of tools that will advance the study of hurricane preparedness using the Python scripting language. Two newly-developed Python geoprocessing scripts developed for this study were demonstrated to show how a GIS scripting language in conjunction with GIS can be used to analyze evacuation plans associated with such storms. Two new tools were developed and evaluated: The Road Intersection Flood Analysis Tool and The Hurricane Evacuation Zone Tool.

The Road Intersection Flood Analysis Tool used road and flood data to determine which intersections are inundated during a flood. The core geospatial component of this script does not currently exist as a tool in GIS software and demonstrates how a user can create custom geoprocessing tools if they do not already exist.

The Hurricane Evacuation Zone Tool performed an analysis that delineated recommended hurricane evacuation zones based on specific criteria. It built on previous research by taking an established, peer reviewed method and converted it from an obscure GIS platform to the well established ESRI GIS platform. This script performed the same analysis as the original, but given Python's ubiquity and low cost, more people will have the ability to use it.

The scripts presented in this study demonstrate the power and importance of tool development for GIS analyses. Using hurricane mitigation as the focus of this study, these scripts can process tedious and repetitive calculations in such a manner that even a novice GIS user can reap the benefits of these powerful tools.

CHAPTER 2: LITERATURE REVIEW

2.1 Hurricanes

Hurricanes are tropical cyclones with wind speeds of greater than or equal to 74 miles-hour⁻¹ measured at 10 meters above the earth's surface (WMO, 2006).

A hurricane can wreak havoc on anything its winds and storm surge can reach. Recently, the United States was at the center of the worst series of hurricanes in recorded history. The US National Oceanic and Atmospheric Administration summarized the 2005 hurricane season as a season of firsts: (1) the first season of 28 named storms; (2) the first season with 15 hurricanes; (3) the first season with four Category 5 storms; and (4) the first season with four major hurricanes hitting the coast of the United States (NOAA, 2006). In all, 2005 hurricanes cost the US over \$100 billion in damage (NCDC, 2006) and claimed over 1,700 lives (NHC, 2007a). As these numbers indicate, hurricanes are serious weather events and should be monitored.

2.2 New England and Hurricanes

The Caribbean Sea, the Gulf Coast states, and the southern Atlantic seaboard of the United States are areas typically affected by tropical cyclones. This study concentrated on the New England Region and specifically the state of Connecticut. The coastal states of New England (Connecticut, Maine, Massachusetts, New Hampshire, and Rhode Island) also have a well documented history of storm hits. It is not unusual for New England to receive a hit from a hurricane (Boose et al., 2001). They explain that the storms are typically mid- to late-stage hurricanes that originated in the southern

latitudes. While occasional Category 3 storms are documented, the usual New England hurricane is generally weaker. The most problematic storms are those that take a northerly path and spin directly into New England. In this case, all parts of the storm are over land, causing maximum damage. Table 1 summarizes the five categories of storms using the Saffir-Simpson scale, the method used to categorize storms by their intensity.

Category	Wind Speed (miles-hour⁻¹)	Effects	Surge
1	74-95	No real damage to building structures. Damage primarily to unanchored mobile homes, shrubbery, and trees. Also, some coastal flooding and minor pier damage.	4-5 feet
2	96-110	Some roofing material, door, and window damage. Considerable damage to vegetation, mobile homes, etc. Flooding damages piers and small craft in unprotected moorings may break their moorings.	6-8 feet
3	111-130	Some structural damage to small residences and utility buildings, with a minor amount of curtainwall failures. Mobile homes are destroyed. Flooding near the coast destroys smaller structures with larger structures damaged by floating debris. Terrain may be flooded well inland.	9-12 feet
4	131-155	More extensive curtainwall failures with some complete roof structure failure on small residences. Major erosion of beach areas. Terrain may be flooded well inland.	13-18 feet
5	155+	Complete roof failure on many residences and industrial buildings. Some complete building failures with small utility buildings blown over or away. Flooding causes major damage to lower floors of all structures to the shoreline. Massive evacuation of residential areas may be required.	18+ feet

Table 1 The Saffir-Simpson Scale (Louisiana Department of Homeland Security and Emergency Preparedness, 2012)

Perhaps the worst New England hurricane in recent times was the New England Hurricane of 1938. Also named “The Long Island Express,” the storm, characterized by rainfall of 35 cm (14 inches) of precipitation and winds exceeding $124 \text{ miles-hour}^{-1}$, affected Long Island, then central Connecticut and Massachusetts before moving into Vermont and later the Canadian province of Quebec (Foster and Boose, 1992). This storm caused a 5-meter (17 foot) storm surge in Rhode Island with 15-meter (50 foot) waves at Gloucester, MA and led to an estimated 600 New England deaths (Francis, 1998).

One particularly significant hurricane that affected New England in recent years was Hurricane Gloria which struck in 1985. It was a Category 3 storm with maximum sustained winds of $145 \text{ miles-hour}^{-1}$ at its peak strength while over the Atlantic Ocean; wind readings in Bridgeport, CT measured $74 \text{ miles-hour}^{-1}$ while Boston, MA saw winds around $81 \text{ miles-hour}^{-1}$ (Case, 1986). In all, this storm caused \$900 million in damage while taking eight lives.

While New England hurricanes are rarer than those that affect the states along the Gulf Coast, for example, it remains very important for this area to maintain hurricane mitigation plans that reflect current demographic trends and infrastructure. The current State of Connecticut’s hurricane evacuation plan is based largely on research completed over twenty years ago. This alarming fact may leave some unaccounted-for residents at risk from the damage a future hurricane can bring. A coastal portion of the State of Connecticut was chosen as the study area for this research due to the outdated plan currently in place. If rare storms with Saffir-Simpson readings of 3 or above would hit

this part of New England, one would expect it to cause widespread destruction. If this scenario becomes an actuality, it will require intensive planning to mitigate the problems brought on by such a storm. The State of Connecticut must have a comprehensive plan to protect its residents from the harmful impacts of strong hurricanes.

The most recent significant storm was Hurricane Sandy. This unusually strong storm battered much of the east coast of the United States, particularly in New Jersey and New York. Coastal as well as inland New England saw a powerful storm surge, unrelenting rain, and significant wind damage. The storm hit the northern Atlantic coast of the United States in late October, 2012. Hurricane Sandy ended up being the second-costliest hurricane in United States history, only to be topped by Hurricane Katrina seven years earlier.

2.3 State of Connecticut's Current Evacuation Plan

In order to evaluate its optimality, it is instructive to understand the current condition of Connecticut's coastal evacuation plan in response to a tropical storm event. If not updated periodically to account for demographic changes, an evacuation plan will not be effective, putting lives at risk. This information will provide the baseline for this research and will guide the decisions made when attempting to evaluate and to make improvements.

There are two sections of Connecticut's evacuation plan: the procedures followed by the upper echelons of government when issuing an evacuation and the actual evacuation plan detailing the logistics of where to move a threatened population.

During situations where a hurricane may strike Connecticut, a thorough procedure is in place that outlines how the decision to evacuate is made (CT DEMHS, 2006). The first portion of this plan is to monitor the threat using the latest information from the National Hurricane Center and the local National Weather Service. If the storm appears to be a threat to Connecticut, the Department of Emergency Management and Homeland Security will begin to notify town officials of the possibility of a state-issued evacuation. If an evacuation is proclaimed, town officials are instructed to begin preparing the local shelters for evacuees. The emergency manager official of each threatened town selects shelters ensuring they will provide a safe location for residents to wait out the impending storm. If the weather conditions continue to deteriorate, either the Governor or the Department of Emergency Management and Homeland Security Commissioner will announce publicly either a recommended or a mandatory evacuation depending on the severity of the storm. This requires a designated 'Hurricane Warning' along parts of Connecticut's coastline as deemed by the National Hurricane Center. Local media outlets will also be notified in order to disseminate the evacuation information to the public. At this time, all shelters should be prepared to begin accepting evacuees. After the evacuation proclamation is made, state officials must notify the appropriate federal, state, and private agencies of the evacuation. At this point, all involved state and local authorities should begin evacuation procedures they deem necessary.

The other portion of the current evacuation plan includes how an evacuation should take place. This plan is used to perform the actual evacuation. According to the Connecticut Department of Emergency Management and Homeland Security Region 4 Coordinator, a

detailed evacuation routing plan does not exist, although local municipalities may designate specific roads and label them as such (Anthony Scalora, personal communication, August 15, 2007). A similar response was given by a Connecticut Department of Transportation representative (Jim Mona, personal communication, September 10, 2007).

Attempts were made to reach each town in the study area to solicit their evacuation plans. Not all towns responded to this request. The information that was obtained, however, indicates that there is no systematic evacuation plan that works in concert from town to town. The town of Groton, for example, assumes that residents are familiar with the road network and trust they will make the proper route choices during a coastal evacuation (Joe Sastre, personal communication, January 3, 2007). The evacuation signs and predetermined evacuation routes the town of Groton does use is designed for tourists. Mr. Sastre also noted that some residents rallied against designating evacuation routes since it was feared that it may adversely affect land values to the designated routes.

The 2006 Natural Disaster Plan also provides some background information on behavior and other ancillary data. According to the document, the US Army Corps of Engineers concluded in a 1988 study (USACE, 1988) that there is no significant difference between a category 1 and 2 storm as they would produce similar effects on Connecticut's coast such as flooding and wind damage; these storm categories are collectively be labeled as a "weak" storm. A category 3 and 4 storm also has similar characteristics and is categorized as a "strong" storm. Category 5 storms are considered to be theoretical impossibilities in this part of the country. The study also estimated that it would take the

average person 7 hours from the time of finding out about the evacuation to arrive at their destination. This includes making all personal arrangements, packing, and driving. In addition to the 7-hour evacuation time, 2 additional hours are needed to disseminate the evacuation proclamation. It was therefore estimated that it would take an approximation of 9 hours for the average person to move to safety from the initial alert.

The US Army Corps of Engineers hurricane study of 1988 also delineated coastal hurricane evacuation zones. These zones are intended to manage small portions of a population during an evacuation event. They were created using inundation data and other identifiable geographic features such as Census Block boundaries that compartmentalized the population. If a threat of flooding was perceived, zones that have the potential to flood would be evacuated. The zones were categorized into two groups – those who would be threatened by a Category 1 or 2 hurricane and those who would be threatened by a Category 3 or 4 hurricane.

The state and local municipalities maintain a list of shelters – each one with written directions, location, and capacity. The list is not available to the public at this time due to legal and homeland security issues; shelter location is given to the public only after an evacuation proclamation. When it is deemed necessary, the shelter locations are disseminated to the public using television and radio broadcasts.

A set of maps published in 1993 by the US Army Corps of Engineers showed a town-by-town view of the road network and potential flood zones (Anthony Scalora, personal communication, August 15, 2007). These maps indicate that the flood zones were derived from the Sea, Lake and Overland Surges from Hurricanes (SLOSH) inundation

models, a computer-driven flood prediction tool used to measure coastal tidal surge during a hurricane (NHC, 2007b).

When dealing with citizens that cannot provide evacuation for themselves, it is up to the municipalities to provide assistance. Some residents may not have the ability to evacuate on their own due to a lack of owning a personal vehicle; towns are responsible for making arrangements for such transportation. Providing special assistance may also include transporting sick or physically-challenged persons. To determine those who qualify, the town may decide to canvas its citizens to ascertain the needs, but this can sometimes be problematic due to HIPAA privacy laws (Anthony Scalora, personal communication, August 15, 2007).

Studies have shown that in order to fully model a storm evacuation, three components must be in place. First, the number of residents to be moved must be estimated. These residents will likely evacuate from their home and will drive to a predetermined safe destination. These safe locations must be identified. The final part of this process must define a method used to route residents to their destination.

2.4 Evacuation Modeling

Creating hurricane evacuation zones is a common approach in the literature. Wilmot and Meduri (2005) developed a method that delineates these zones using physical parameters of the affected land and of the hurricane. In their study, the study area was defined using SLOSH MEOW (Maximum Envelopes Of Water) and MOM (Maximums of the Maximum) maps that estimate the height of water for a given storm's physical

parameters such as forward speed, wind speed, and direction. The affected land was then divided by zip code, then by major roads that serve each zip code. Land use data were used to remove the uninhabited areas. Elevation data were used to merge adjacent zones based on similar elevation. Using an overlay feature in the GIS software, the authors were able to use Census data to estimate the population of the resulting zones. These zones can be used to pinpoint evacuation areas if a storm was looming.

An important concept with the Wilmot and Meduri study is to use delineation data that the general public is familiar with. This is why zip code boundaries and major roads are used in the study; it is safe to assume most residents will recognize the locations of these features. Having hurricane evacuation zones based on these familiar landmarks allows for easy communication between emergency management officials and the public.

Previous research indicates that evacuations from threatened hurricane evacuation zones should be modeled using the number of evacuating vehicles and not the population of evacuees (Lindell and Prater, 2007). One such formulation of this philosophy developed by Chen et al. (2006) is expressed as:

$$N_v = N_u * N_{vu} * R_p * R_o * P_{vu} \quad (1)$$

Where:

N_v = the number of evacuating vehicles

N_u = the number of housing units

N_{vu} = the number of vehicles per housing unit

R_p = percentage of people choosing to evacuate

R_o = the occupancy rate of housing units

P_{vu} = the percentage of vehicle usage

Chen et al. (2006) conclude that calculating a value for N_v would be useful in modeling the movement of evacuees on the road network.

The 1988 US ACE report relies on a survey given to different areas of the United States in order to calculate how many vehicles would evacuate. One of the areas surveyed was Groton, CT. The results of the survey showed that approximately 75% of vehicles available to evacuees will be used during an evacuation. The survey was given after Hurricane Gloria threatened the local area.

Network flow models are used to conceptualize and quantify cost of moving goods from one location to another in an optimized manner. These same methods can be applied to evacuation planning.

One such model is called the Transportation Problem (Hitchcock 1941). A solution of this problem provides the least cost route for a set of origins to service a set of destinations (Caliper Corporation, 2005). The cost of travel can be considered time traveled, distance traveled, fuel costs, or any other method of measuring cost. The origins and destinations each have quantity constraints that must be considered. This model can be used to represent evacuation scenarios if one considers the people in the affected area the commodity to be moved. The model assumes that all components of the

road network are equally accessible. This assumption prevents its use from an evacuation analysis since it may designate too many evacuees onto capacity-limited roads.

A modified version of the Transportation Problem addresses the network constraint issues. The Minimum Cost Flow Problem incorporates capacity constraints on a network and produces a solution that is applicable in an evacuation event (Dunn and Newton, 1992). Cova and Johnson (2003) note that this method is quite effective for long-term evacuation planning. The model will produce a result that achieves three important goals; it will allocate affected citizens to their proper destination, reduce overall travel cost (time), and provide an effective routing scheme. One problem associated with using this method concerns estimating origin flow volumes that are largely dependent on the time of day. Another obstacle that surfaces while using the Minimum Cost Flow approach is the driver's choice to freely select a route to the destination.

Some models such as MASS eVACuation (MASSVAC) were developed specifically to deal with hurricane situations (Hobeika and Jamei, 1985). MASSVAC is an algorithm that simulates large evacuations. This model takes a trial-and-error approach to define 'good' and 'bad' evacuation plans, but is time intensive and can produce suboptimal solutions.

Highly sophisticated dynamic traffic flow models have been developed that can aid in evacuation planning. These models produce optimal results using linear programming techniques. They investigate how vehicles enter a transportation network (de Palma and Marchal, 2000). Dynamic Traffic Analysis (DTA) proposed by Peeta and Ziliaskopoulos (2001) is one such model that can be used in an analytical method to estimate link travel

times. DTA can be successfully used to plan for mass evacuations even though it was not originally created for this purpose. Chiu et al. (2007) have developed such a model that not only incorporates DTA methods, but also addresses evacuation-specific parameters. According to this paper,

“[the model] proposes a network transformation and demand modeling technique that allows the optimal evacuation destination, traffic assignment and evacuation departure schedule decisions to be formulated into a unified optimal traffic flow optimization model by solving these decisions simultaneously.”

The goal of the model is to minimize the number of vehicles in a given location at a given time and produces an optimized evacuation plan for the study area. Once the optimized evacuation plan is defined, it is tested using the elevation data to ensure no evacuation routes run the risk of being flooded.

The US ACE study (US ACE, 1988) employed the use of a proprietary method of calculating routing concerns for coastal Connecticut called NETVAC2 and was used to model how an evacuation would unfold. It uses a link and node system with other variables such as road capacity and widths of intersections (Southworth, 1991). NETVAC2 requires route preference information as well as the relative volumes

of traffic at the time of evacuation. The stochastic approach taken by NETVAC2 is useful in determining evacuation timing and route selection by evacuees.

2.5 Flood Modeling

A successful evacuation requires a detailed plan where all significant variables are identified. Some of the variables mentioned earlier such as population estimates, population locations and storm intensity play major roles in planning for and executing an evacuation. Another important variable to consider is storm surge and its affect on certain structures, especially roadways, bridges, and shelters. As a storm moves toward land, the pileup of water along the banks causes the sea to rise and consequently floods the area.

The Federal Emergency Management Agency (FEMA) typically uses the SLOSH model to predict floods caused by a hurricane's storm surge (Greenwood and Hatheway, 1996). Modeling hurricane storm surge with SLOSH incorporates storm parameters such as pressure, size, forward speed, track, and wind speeds and are applied to the physical characteristics of the affected land such as bay and river configurations, bridges, roads, and land elevation (NHC, 2007b). SLOSH, a product of the National Hurricane Center (NHC), is the method used by most emergency managers, including those in Connecticut (CT DEP, 2004), to determine which areas to evacuate (NHC, 2007c).

GIS modeling techniques have been employed to define inundated segments of roads during a flooding event. One such study involves a two-step approach: defining the flood extent and identifying flooded road segments (Cai et al., 2007). The first portion of this

study uses hydrology network data coupled with high-resolution digital elevation model (DEM) imagery to predict where water will spread during a flood. Once the areas of flooding are designated, a road inundation analysis can be performed. Each line segment representing a road is subdivided into smaller portions that match the resolution of the DEM. Each segment is matched to the corresponding elevation on the DEM and is assigned a z-value. These data, when paired with the flood extent, will detail which roads are passable. Since it will define small portions of road that are underwater, this method is particularly practical for evacuation analysis. For example, if only a 10-meter stretch of a 100-meter road is inundated, the whole road will not be considered flooded – only the flooded 10-meter portion would be designated underwater, however, this implies that the entire road is likely impassable by road vehicles. The information provided in this type of analysis is best used after a flood event has occurred when emergency personnel must avoid flooded roads while responding to the needs of the public.

CHAPTER 3: STUDY AREA

3.1 Introduction

This study uses two different study areas. The tools created for this research required different types of study areas due to the functions they perform. Both areas are located in southeast Connecticut, an area that is susceptible to the effects of a hurricane.

3.2 Study Area for the Road Intersection Flood Analysis Tool

Figure 1 shows the Connecticut towns selected to test the Road Intersection Flood Analysis Tool. Included were the towns of Old Lyme, East Lyme, Waterford, New London, Groton, and Stonington. These towns were selected due to their adjacent location to Long Island Sound and their susceptibility to receiving a storm surge from a hurricane.

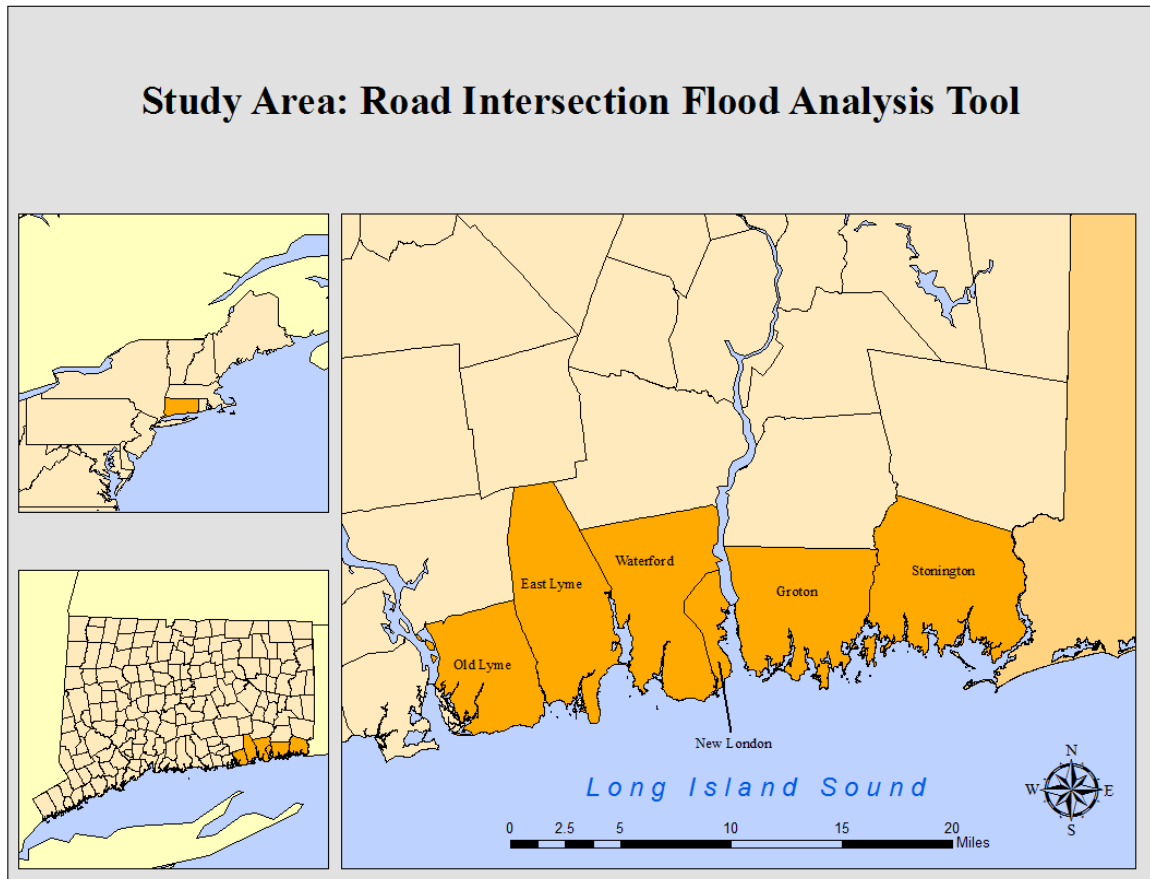


Figure 1 Study Area: Road Intersection Flood Analysis Tool

3.3 Study Area for the Hurricane Evacuation Zone Tool

The study area for the Hurricane Evacuation Zone Tool includes the twelve towns of East Lyme, Groton, Ledyard, Lyme, Montville, New London, North Stonington, Norwich, Old Lyme, Preston, Stonington, and Waterford (Figure 2). The Hurricane Evacuation Zone Tool was tested using this area. This region is located wholly within New London County, Connecticut, USA. These towns encompass approximately 315 square miles (201,780 acres). 209,205 residents live within this region (United States Census Bureau 2009).

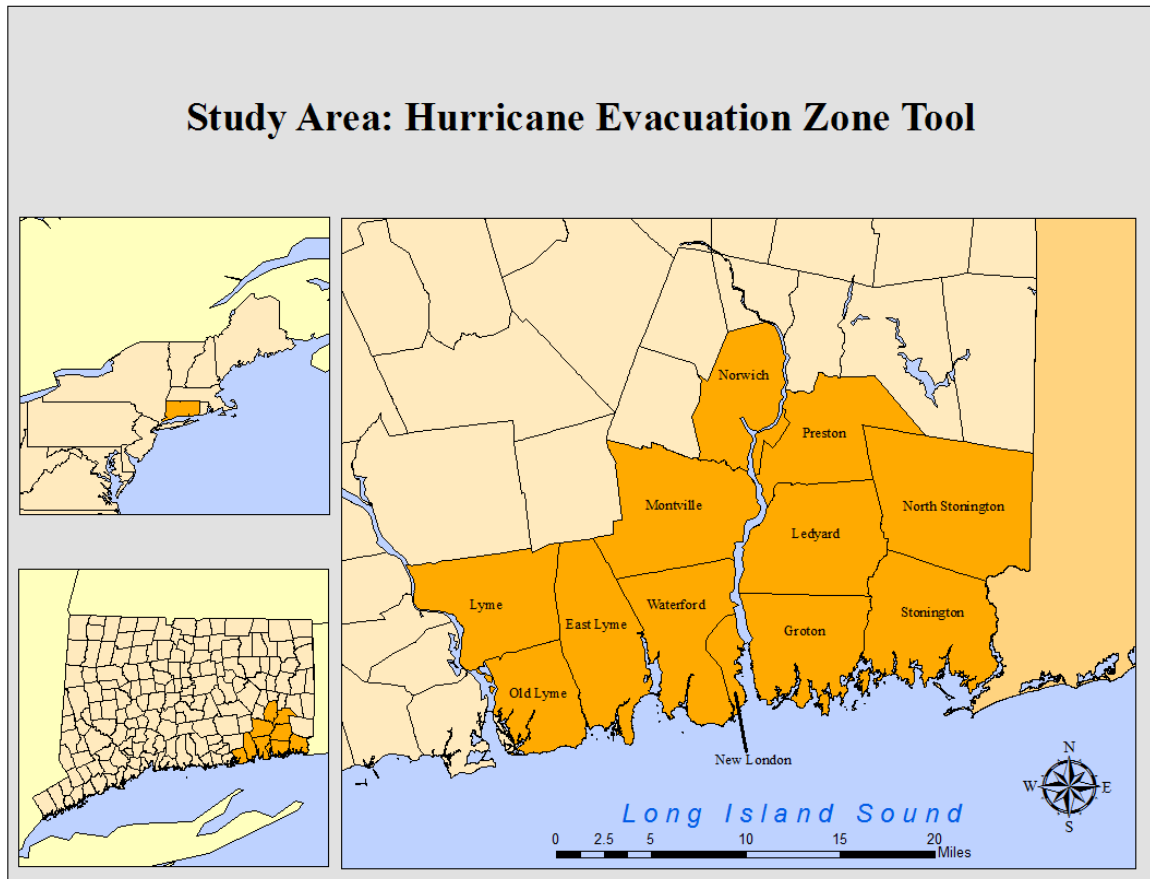


Figure 2 Study Area: Hurricane Evacuation Zone Tool

These study areas were selected for several reasons. Its shores are adjacent to Long Island Sound which opens into the Atlantic Ocean. Between the cities of New London and Groton, the Thames River empties into Long Island Sound. The Thames River could act as a conduit for rising water from a storm surge and place the area along its banks at a greater risk of flooding. The city of Norwich sits at the northern-most point of the Thames River at the confluence of the Yantic and Shetucket Rivers (Figure 3). Historically, there have been many tropical weather systems that have threatened and directly impacted the study area. Since 1851, there have been 47 tropical storms or

hurricanes that have passed within 50 miles of this study area (Figure 4). The towns of North Stonington and Lyme, while not contiguous with either Long Island Sound or the Thames River, were included.

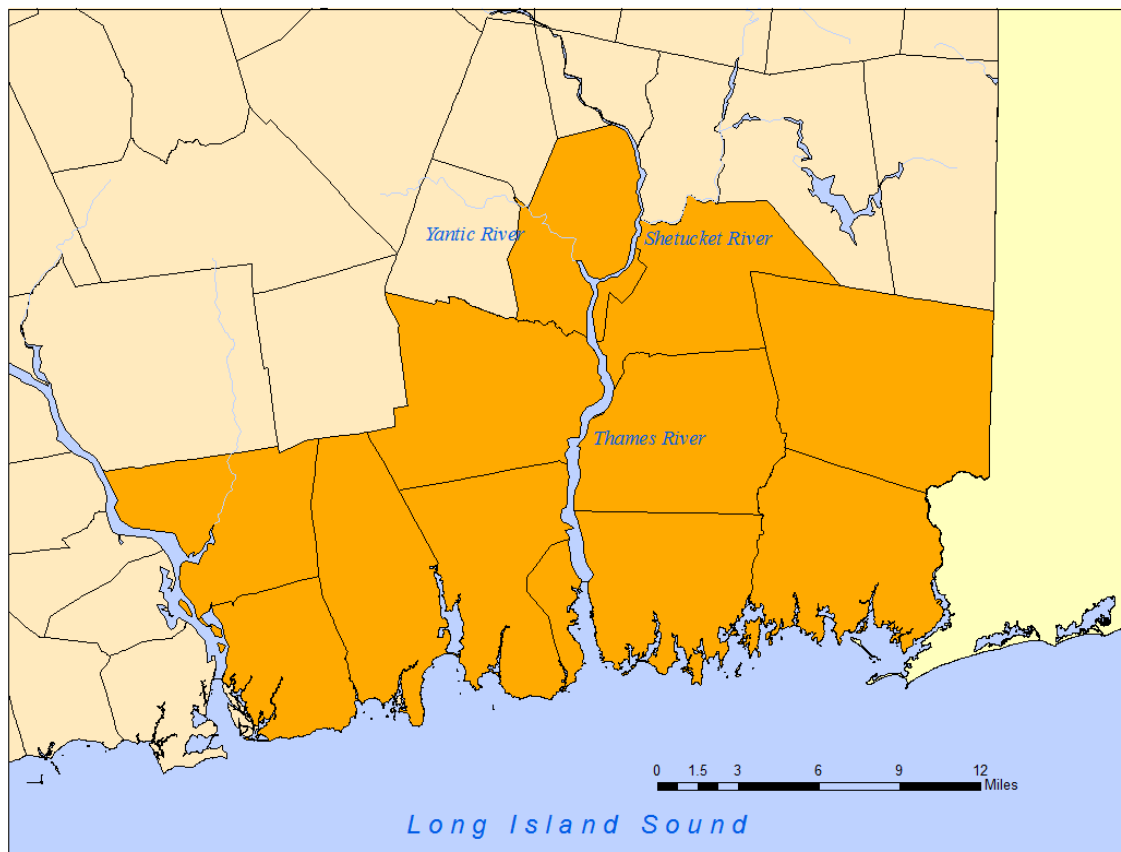


Figure 3 Major Rivers within Study Area

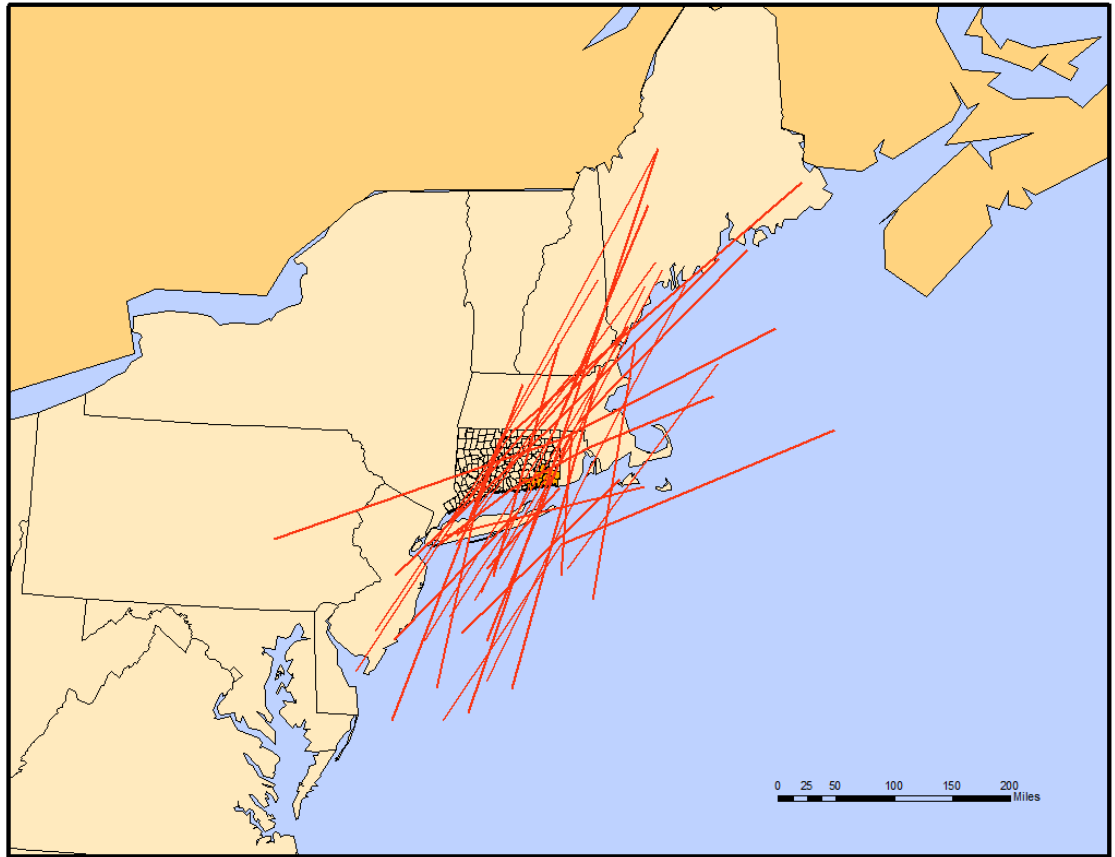


Figure 4 History of Tropical Storm Activity within 50 Miles of Study Area Since 1851

The selected study area also has many important facilities that play major roles in the economy, infrastructure, and military. Examples of these facilities include Millstone Nuclear Power Station, US Naval Submarine Base, and two large casinos. Figure 14 illustrates a few noteworthy facilities within the study area. Table 2 provides the estimated number of people at each facility noted in Figure 5.

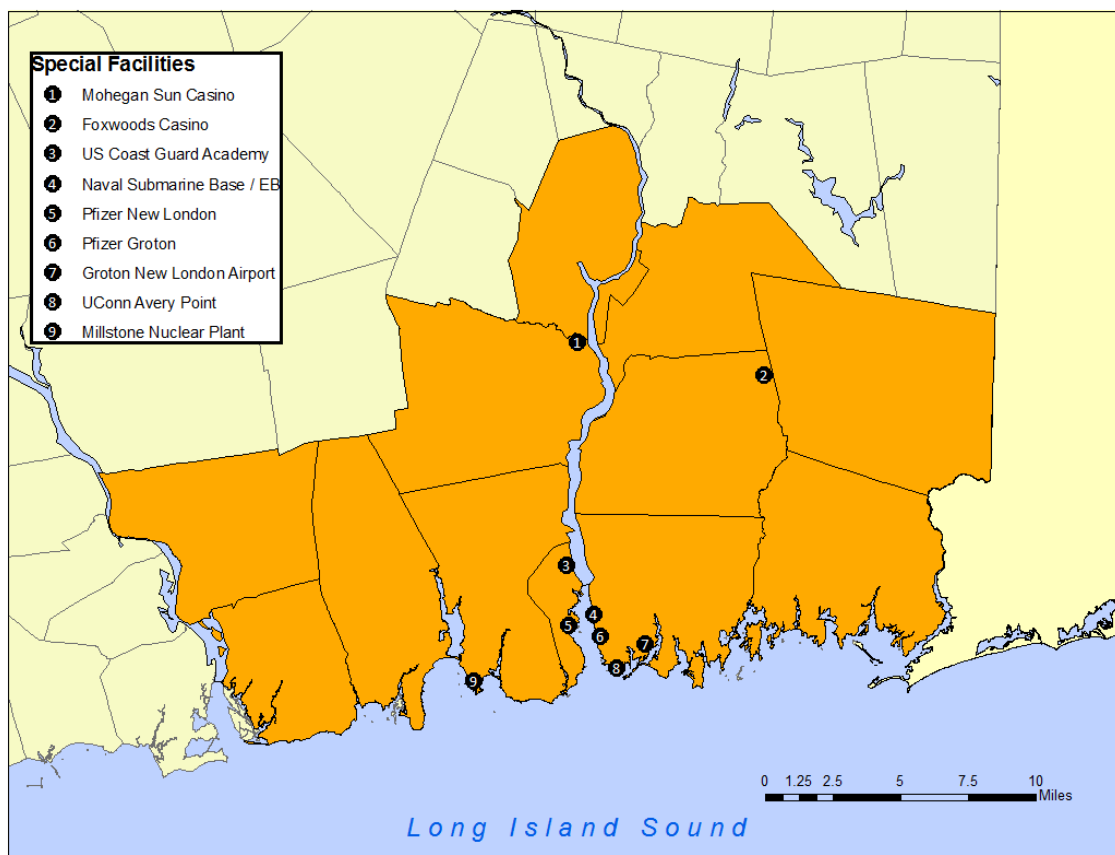


Figure 5 Special Facilities within Study Area

Map Label	Facility	Facility Population
1	Mohegan Sun Casino	10,000 Employees (LinkedIn, 2012)
2	Foxwoods Casino	8,000 Employees (Foxwoods, 2012)
3	US Coast Guard Academy	973 Students, 122 Faculty (United States Coast Guard Academy, 2011)
4	Naval Sub Base/EB	7500 Active Duty Sailors, 2500 Civilians (MilitaryNewcomers.com, 2012)
5,6	Pfizer (both locations)	4000 Employees (Pfizer, 2012)
7	Groton/New London Airport	500 Employees (GrotonNewLondonAirport.com, 2012)
8	UConn Avery Point	748 Students (BrainTrack.com, 2012)
9	Millstone Nuclear Plant	1285 Employees (Courant.com, 2012)

Table 2 Estimated populations at special facilities within study area

Chapter 4: Data and Methodology

4.1 Introduction

Implementing a GIS frequently involves using advanced techniques and customizations that allow for highly specialized analyses. These advanced techniques may include running multiple geoprocessing tools on data, examining results using statistics, or processing data in an iterative fashion until specific closing criteria are met. Emergency management in relation to hurricane preparation and mitigation has its own specialized tool set including proprietary models. Users generally have little control over the internal workings of such models. The advantages of these tools include easy dissemination of methods between users as well as sometimes providing this ability for a user to customize the script for a particular purpose. The goal of this research was to develop a set of tools that will advance hurricane preparedness using the Python scripting language. This chapter includes a detailed description of the study area, the required data, and the methods used to create new tools used for studying the effects of hurricanes.

4.2 Methodology

The methodology used in this research is described in this section. Two new GIS tools developed for this research are discussed individually. The first section contains the Flooded Intersections Tool which can be used to automate the creation of a dataset representing flooded road intersections during a flood event. The second section discusses the Hurricane Evacuation Zone Tool. This tool was developed to design effective hurricane evacuation zones used in evacuation planning. The final portion of

this section contains a summary of these tools and reiterates their utility in emergency management protocol.

4.3 Python

Python is a free, object oriented programming language used across multiple computer platforms (Python Software Foundation, 2009). Its initial development began in the late 1980s by Guido van Rossum, a computer programmer at the National Research Institute for Mathematics and Computer Science in the Netherlands (Artima, Inc., 2009). The programming language is open source.

Since its inception, Python has become a popular choice among programmers for both its powerful tools and the easy-to-use syntax. Special implementations of the original Python language has allowed users to write programs and call up methods of many other common programming languages such as Java and Microsoft's .NET architecture (Python Software Foundation, 2009). With these advantages, Python has earned an audience that employs its flexibility in many different applications.

Python is used as an academic research tool in many fields. O'Boyle and Hutchison (2008) report that they have developed a Python-based set of tools called Cinfony which incorporates three popular but incompatible open source cheminformatics toolkits in one interface. Decision support system (DSS) research has also benefited from the use of Python (Dowhań et al., 2009). Python has been used to model a DSS by calculating multi objective mathematical functions and using artificial intelligence. The biosciences also benefit from Python by using its graphical toolset to create three dimensional

representations of plant structures and even entire plant communities (Pradal et al., 2009). These recreations allow researchers to model spatial interactions between competing plant species.

The idea of using custom scripts to process geospatial data has been an important goal for Environmental Systems Research Institute (ESRI). As outlined by Butler (2005), ESRI first introduced this ability by first releasing ARC Macro Language (AML), then Avenue. These two languages were devised specifically for completing geoprocessing tasks within ESRI software. When ArcGIS 8 was released at the end of 1999, the ability to use common scripting languages such as C++ and Microsoft's Visual Basic was implemented. While this ability was a welcomed improvement over the two original proprietary languages, the process of using these was cumbersome and did not incorporate task automation. ArcGIS 9 was released in mid 2004 and allowed new scripting languages to take full advantage of ArcObjects, the foundation of the ArcGIS platform. The three popular scripting languages now supported in ArcGIS 9 are VBScript, JScript, and Python. Given Python's advantages over the other scripting languages and its popularity amongst those performing automated geoprocessing functions, ESRI has selected Python as its favored scripting language and has fully incorporated it into ArcGIS10.1, the latest version of ESRI's GIS software (ESRI, 2009; ESRI, 2012).

The geoprocessor object in ArcObjects acts as an interpreter between the tools within ArcObjects and the scripting commands of a Python script. In order to use the geoprocessor, a Python script must incorporate lines of code that call up the

geoprocessing object and converts it into a Python object. Figure 6 shows a schematic of how a Python script accesses the geoprocessing tools using the ArcGIS geoprocessor. Additional lines of code are used to define which license level is being used on the computer workstation, the geoprocessing environment including the path of data sources and output locations, and even check out which extensions are required for the type of analysis being performed. Once the geoprocessor is properly defined, the Python script can begin calling up any of the geoprocessing tools within the GIS software. Figure 7 below shows an example of a typical Python command that would execute an intersect command; a function identifies the geometric intersections of the input data. The 'gp' is the geoprocessor object and the 'Intersect_analysis' method is evoking the intersect command from within the Analysis toolbox. The parentheses contain the required and optional parameters which allow the intersect command to properly execute. These parameters can be hard coded values or even variables defined earlier in the script. All of the geoprocessing commands written within Python have a similar syntax.

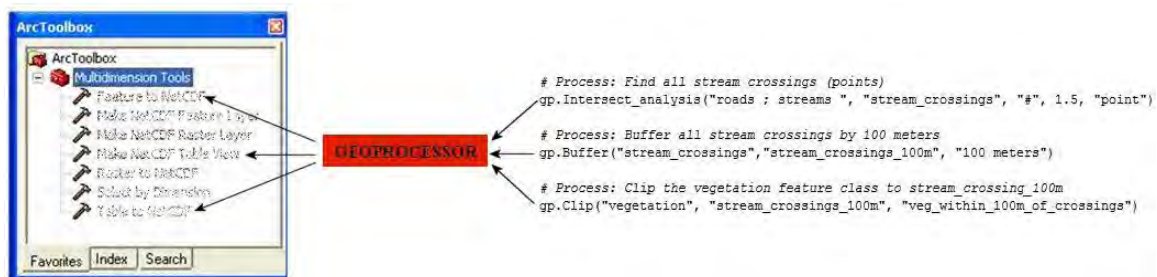


Figure 6 Schematic of how Python accesses geoprocessing tools in ArcGIS

```
gp.Intersect_analysis("roads ; streams ", "stream_crossings", "#", 1.5, "point")
```

Figure 7 Example of a Python geoprocessing command

4.4 The Road Intersection Flood Analysis Tool

The Road Intersection Flood Analysis Tool was developed in order to quickly ascertain which road intersections would be under water during a flood event such as a hurricane. The output of this script can be used to route emergency responders in such a way that avoids flooded intersections. The user has the option to supply real-time flood data to produce an up-to-the-minute representation of the flood event as it happens. Avoiding these impassable intersections will allow emergency personnel to develop routes for emergency vehicles to efficiently provide emergency public assistance including police, fire, and EMS services during a hurricane. Intersections were used in this study to represent connections along the road network; emergency vehicles should be routed to a destination in such a manner that avoids flooded intersections. This tool included a portion of code that creates a point shapefile using only the intersections of lines, a capability only found in expensive ArcGIS add-on modules. The script could easily be modified to identify flooded road segments. The complete Road Intersection Flood Analysis tool script can be found in Appendix A.

4.5 Required Data for The Road Intersection Flood Analysis Tool

The road intersection flood analysis tool required three data types. The user must provide a boundary file that will represent the extent of the study area. The road network must also be provided. It is not necessary for these data to be topologically accurate; nodes at road intersections are not necessary. The final set of data needed is a polygon file representing the extent of rising water. The following sections show how the script

determines which nodes (road intersections) are passable and which are not.

4.6 Script Implementation for The Road Intersection Flood Analysis Tool

The ArcGIS software package allows the user to convert a Python script into an ArcToolbox tool; the Road Intersection Flood Analysis Tool was developed as an ArcGIS tool that provides GIS personnel an easy to use frontend interface. The script can now be executed from ArcToolbox in a similar fashion to other tools. No programming experience with Python and very little GIS knowledge is needed to run this tool.

Figure 8 shows the ArcGIS tool developed for The Road Intersection Flood Analysis Tool. The tool had six parameters that the user customized for their needs. The first parameter allowed the user to supply the town boundary data that outlines the study area. The next required parameter prompted the user for the road network data that was used to generate all potential intersections located inside of the study area. The third parameter was used to specify the location of the SLOSH flood data that determined which intersections were below the flood water during a specific event. To give the user even more control of the output of this tool, the fourth parameter allowed for the input of a SQL statement that chose a subset of all of the available towns defined in the first parameter. The fifth parameter gave the user the ability to select the hurricane strength when the tool was executed. This value was used in conjunction with the SLOSH data to identify the intersections below the flood water. The last parameter let the user select a location on the computer to save the output of the tool. Once these parameters were populated, the tool was executed and it produced the output as specified by the user.

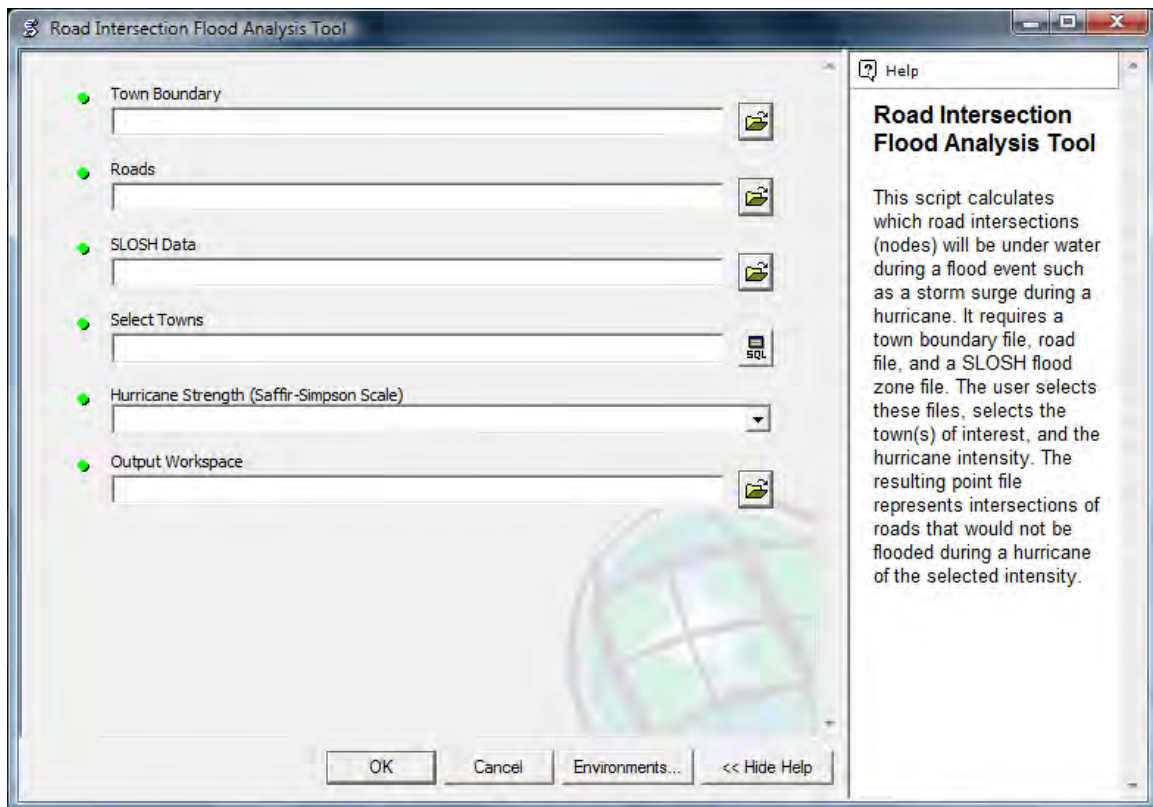


Figure 8 ArcToolbox Interface for Road Intersection Flood Analysis Tool

4.7 The Road Intersection Flood Analysis Tool Test Application

The Road Intersection Flood Analysis tool script consisted of a program written in the Python programming language. The test application for this script involved only the immediate coastal towns of the larger study area defined in section 3.2, including the towns of Old Lyme, East Lyme, Waterford, New London, Groton, and Stonington. Figure 1 shows this subset study area. The flood polygons used in the test application were obtained from the SLOSH inundation data which defines the areas expected to flood from storm surge waters during a hurricane. Appendix C contains a very detailed account of exactly how the script operates.

Sample data representing a portion of the southeast Connecticut coast were used to demonstrate this tool. These data were a subset of the study area described in section 3.3. The tool can be used on large datasets, but for the purposes of illustrating how it works, a small dataset was tested. Figure 9 shows the small dataset which is to the east of the Thames River in the Groton area.

While either empirical or observed flood data can be used, the SLOSH polygon file created by the National Hurricane Center was selected to represent the inundated locations. Figure 10 shows the coverage of the SLOSH polygons for the selected area. Note that for the SLOSH data provided for Connecticut, Saffir-Simpson hurricane categories 1 and 2 are combined into one category since NOAA has determined that there is little difference between the two categories in this location.

The first product of the script calculated all possible road intersections in the study area. The study area has 638 road intersections. Figure 11 shows all intersections in the dataset. The road intersection data were stored in a shapefile and are used later in the script.



Figure 9 Area of interest for Road Intersection Flood Analysis Tool

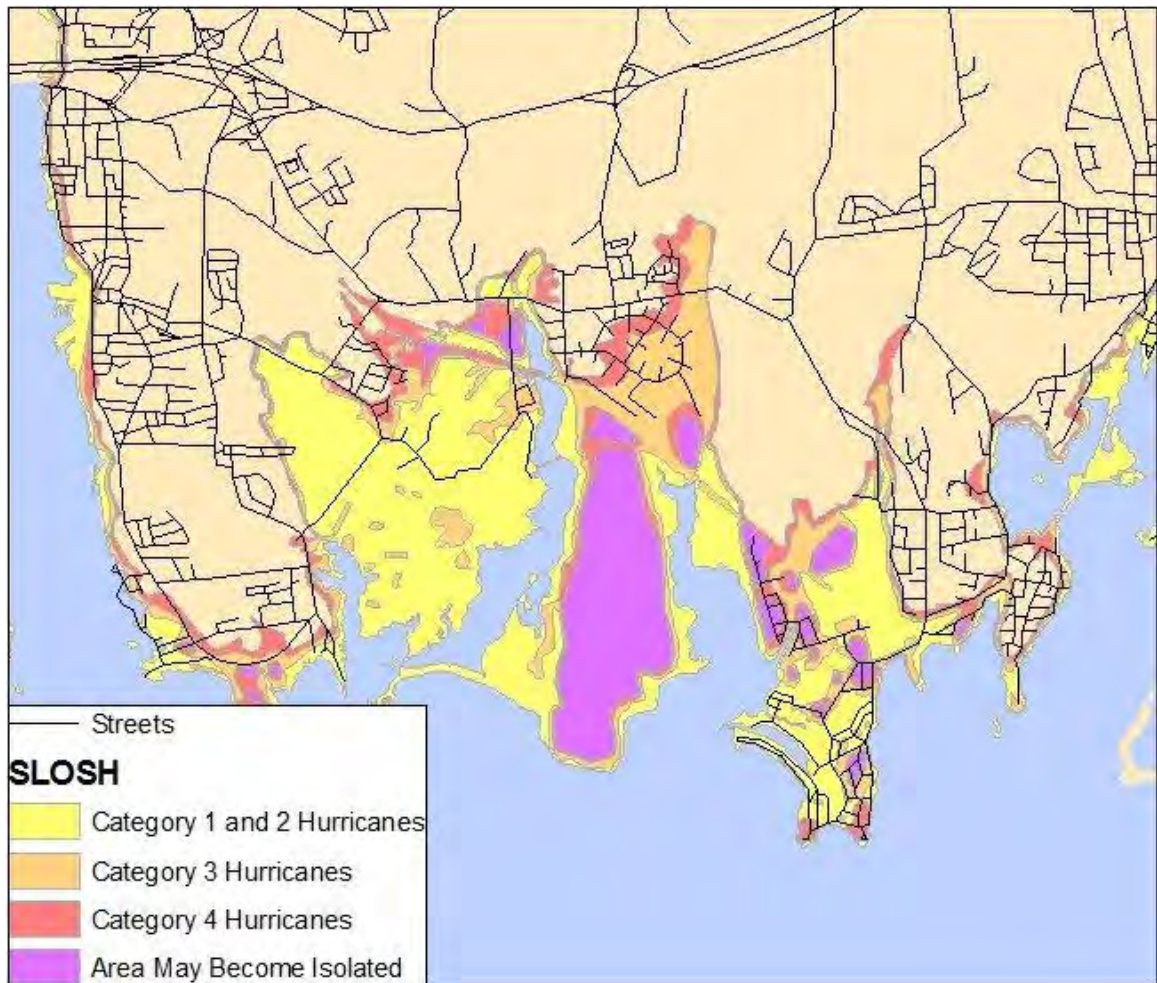


Figure 10 SLOSH Polygons on Area of Interest for Road Intersection Flood Analysis Tool

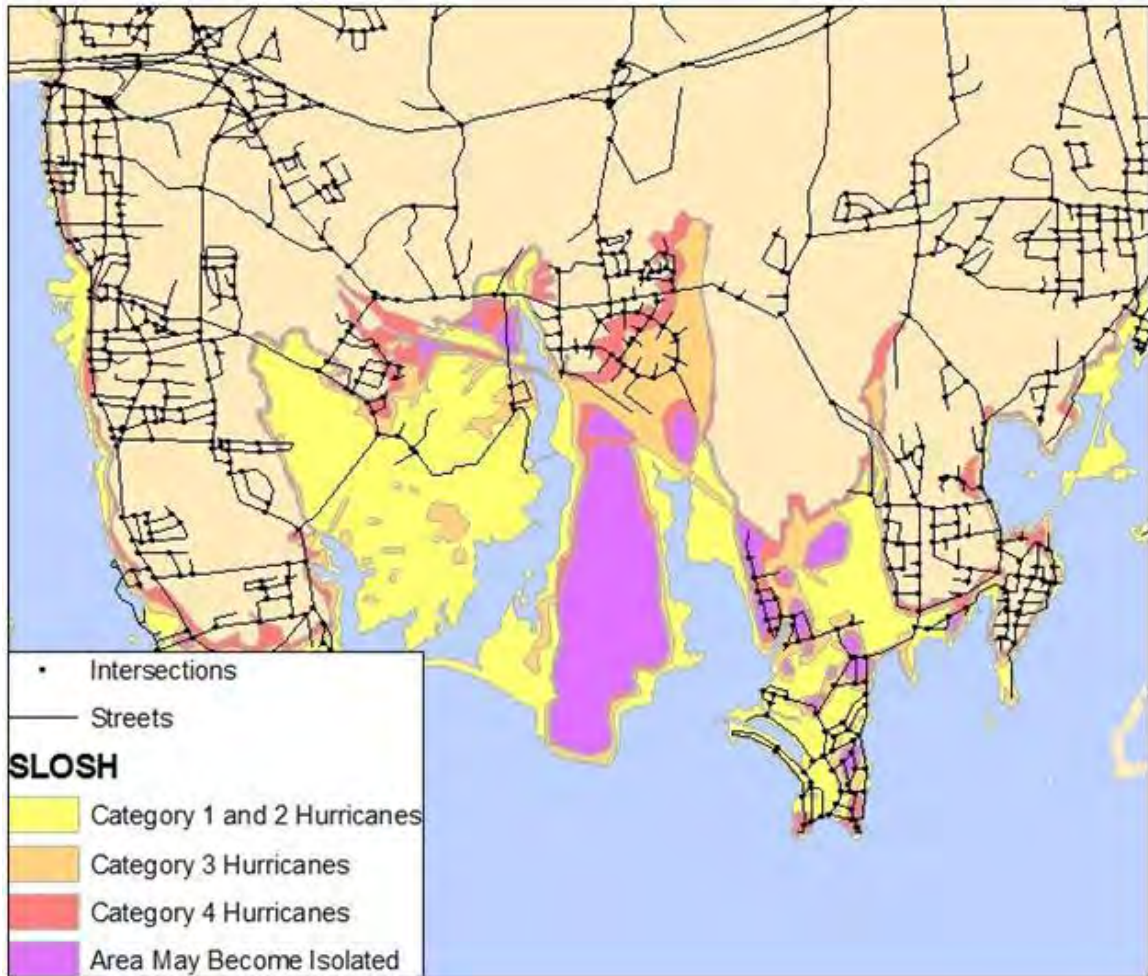


Figure 11 All road intersections created by Road Intersection Flood Analysis Tool

4.8 The Hurricane Evacuation Zone Tool

The Hurricane Evacuation Zone Tool was largely based on work performed by Wilmot and Meduri (2005). Their research attempted to define hurricane evacuation zones based on specific criteria. This was accomplished by using Caliper Script within TransCAD software. The resulting script has limited utility; it requires the user to possess a TransCAD license and to learn a new programming language that is only used within TransCAD.

The approach taken in this research attempted to perform the same analysis, but using Python in conjunction with the ArcGIS system. This allows anyone with access to ArcGIS the ability to obtain the free Python software and perform this analysis. TransCAD is a niche product used for transportation GIS and is not as widespread as ESRI's ArcGIS. This limited the Wilmot and Meduri method of hurricane evacuation zone delineation.

4.9 Required Data for the Hurricane Evacuation Zone Tool

The hurricane evacuation zone tool required the use of three unique datasets. A digital elevation model (DEM) was used to represent the elevation across the study area. Shapefiles representing zip code boundaries and major roads and highways were needed. These files were used to establish the beginning set of possible hurricane evacuation zone polygons. This was done by performing a union on all of these polygon data, combining them into one polygon dataset. The script began merging these polygons that had similar hypsographic characteristics.

4.10 Script Implementation for the Hurricane Evacuation Zone Tool

Like the previous script developed in section 4.2.4, this script was also converted to a tool in ArcToolbox. This allowed the user to run the script without having to learn Python and reduced the chances of accidental changes to the script syntax.

The Hurricane Evacuation Zone Tool interface is shown in Figure 12; it required four parameters to operate properly. The first parameter set a temporary workspace where intermediate files were stored and processed. The polygon data created in previously was

selected in the second parameter. The DEM used to merge the polygons was specified in the third parameter. The final parameter gave the user the option to select the output workspace which contained all iterations of The Hurricane Evacuation Zone Tool.

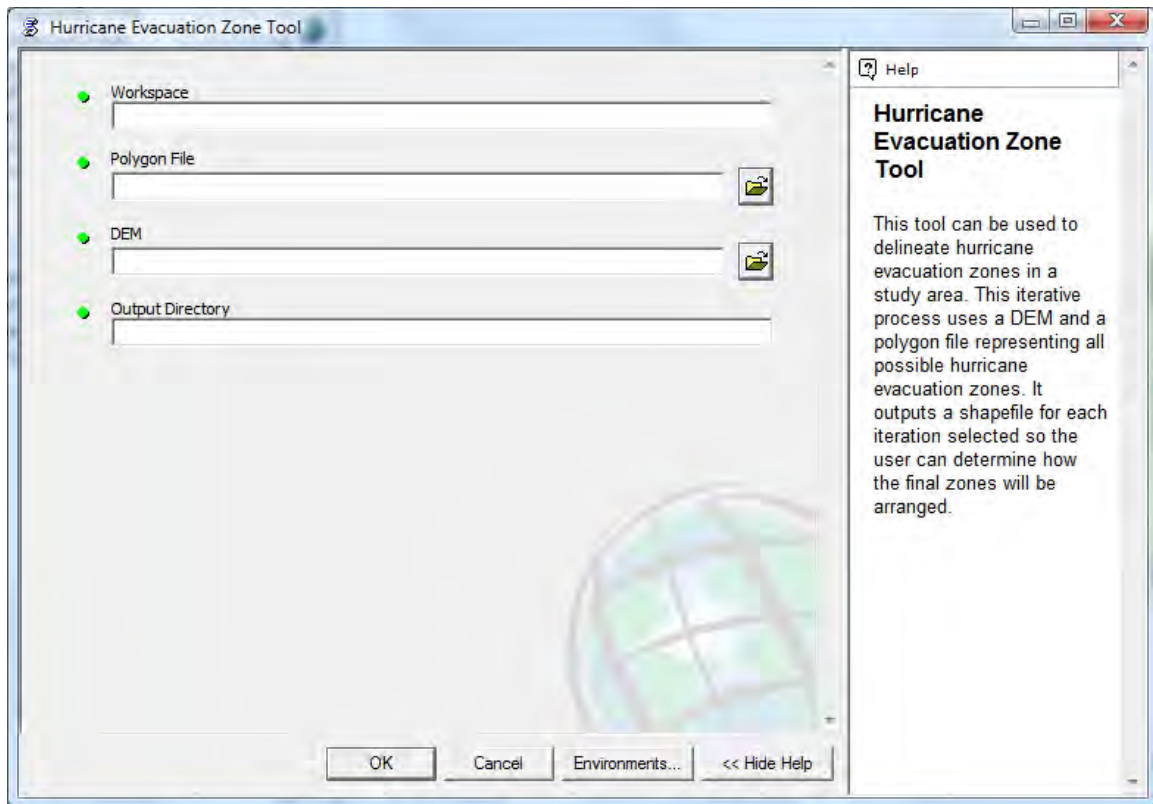


Figure 12 ArcToolbox Interface for Hurricane Evacuation Zone Tool

4.11 The Hurricane Evacuation Zone Test Application

The Hurricane Evacuation Zone script contained 466 lines of code. This script used many advanced features of Python which allows it to run more efficiently than the Wilmot and Meduri script. The entire study area described in section 4.4 was used in this test application. Once executed, the script iteratively processed a polygon file that represented all possible hurricane evacuation zones within the study area. For each iteration, two

adjacent candidate zones were merged into one based on similar characteristics. This process continued until either the number of predetermined iterations was met or until specific closing criteria were obtained. The script presented in this research terminated after 250 iterations are performed.

Before the script was executed, the user must create a polygon shapefile that represents all possible hurricane evacuation zones. Per the method used by Wilmot and Meduri (2005), this data was created using both zip code polygons and a line file representing major roads. The data corresponding to the selected study area were collected and processed by using the Union (Analysis) tool in ArcToolbox. This function took the zip code boundaries and the major roads and processed them into one polygon layer. This procedure produced 300 polygons (Figure 13).

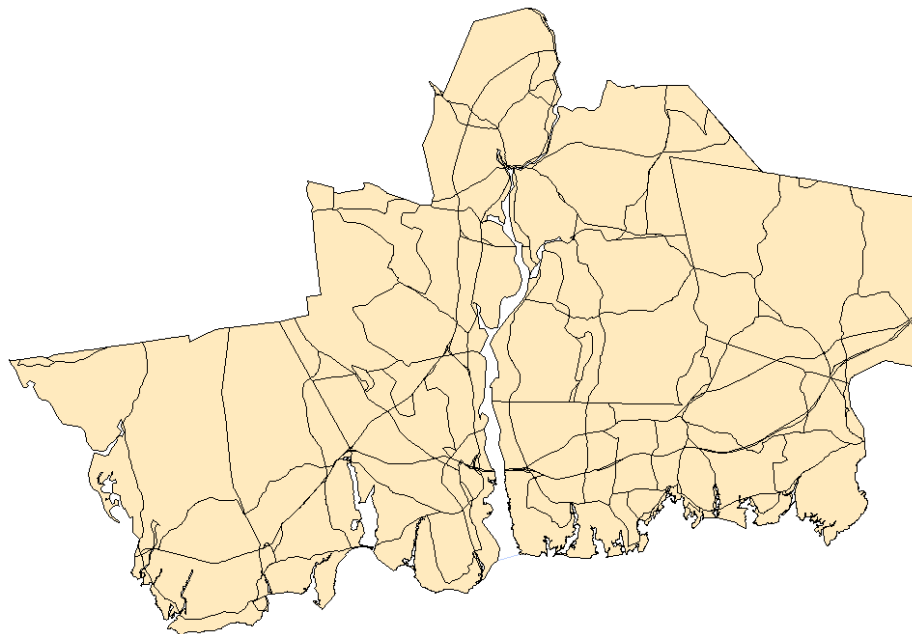


Figure 13 Candidate hurricane evacuation zones

4.12 Software Standards

The two scripts provided in this research were developed using the same software packages. The GIS software used ESRI's ArcGIS software package, version 9.2.0.1324. The software architecture of Python version 2.4.1 running on Microsoft Windows was used to process the scripts. For a friendlier user experience, PythonWin32 (build 210) was implemented which allows for easier script production and troubleshooting. Figure 14 shows the user interface of PythonWin32. Both Python and PythonWin32 are provided on the ArcGIS install DVD. Using the supplied versions of these software packages ensured compatibility between Python and the geoprocessing tools of ArcGIS.

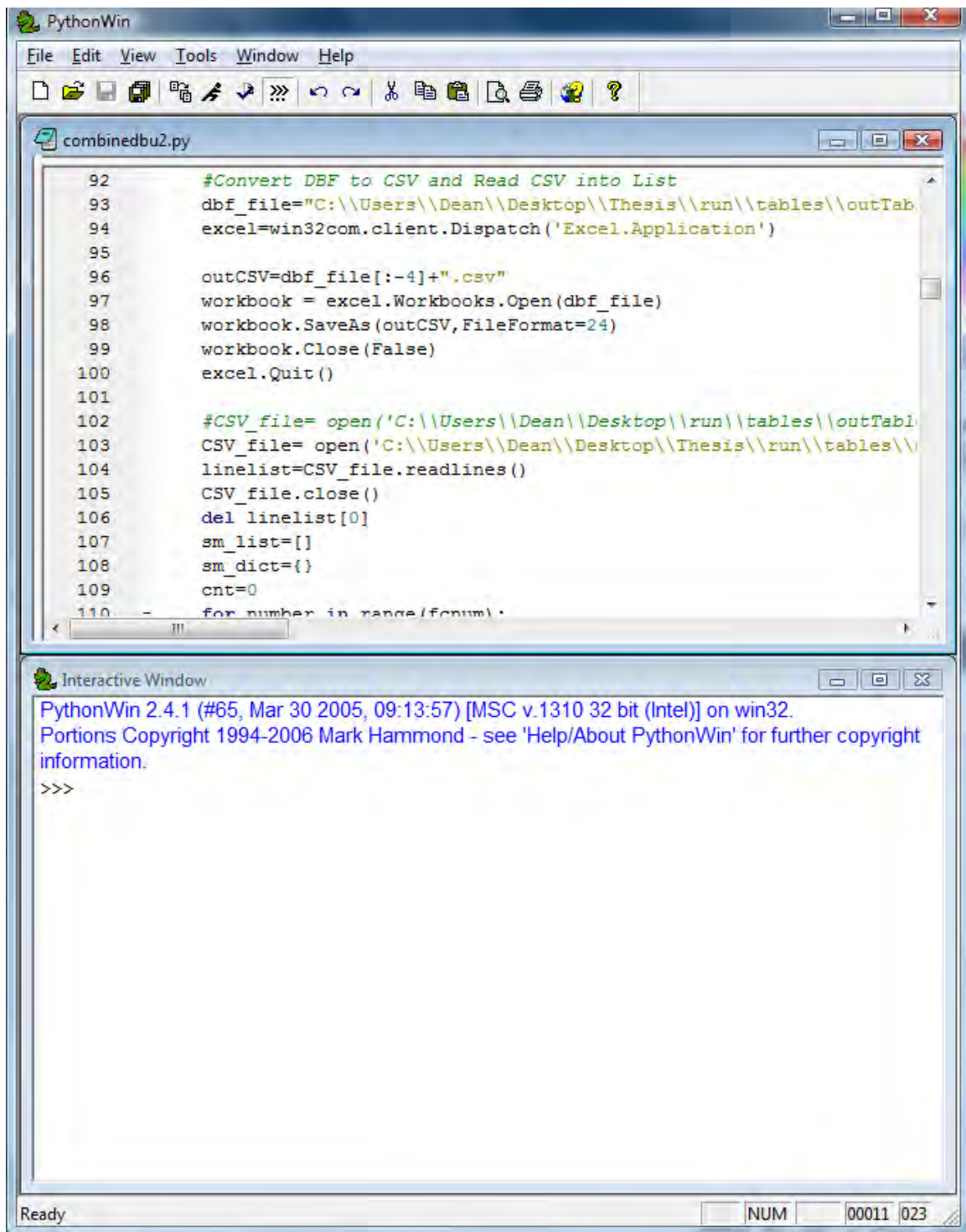


Figure 14 PythonWin User Interface

4.13 Road Network Data

TIGER road network data were obtained from the Connecticut Department of Environmental Protection website (CTDEP, 2008). These data are created by the US Census Bureau in order to facilitate their study of the US population. The CTDEP freely distributes these data which has been modified to represent all roadway segments in the State of Connecticut. The data represents the road network at a scale of 1:100,000. Within the study area, there are 15,140 road segments representing approximately 1,845 miles. Figure 15 illustrates the spatial characteristics of these data.

Data representing the major roads within the study area were required and were obtained from the Connecticut Department of Environmental Protection website. This data represented all major roadways including interstate highways, state highways and most primary roadways. This dataset is less detailed than the TIGER data since it was digitized at a smaller scale (1:250,000) and is not intended to illustrate all roads in the study area.

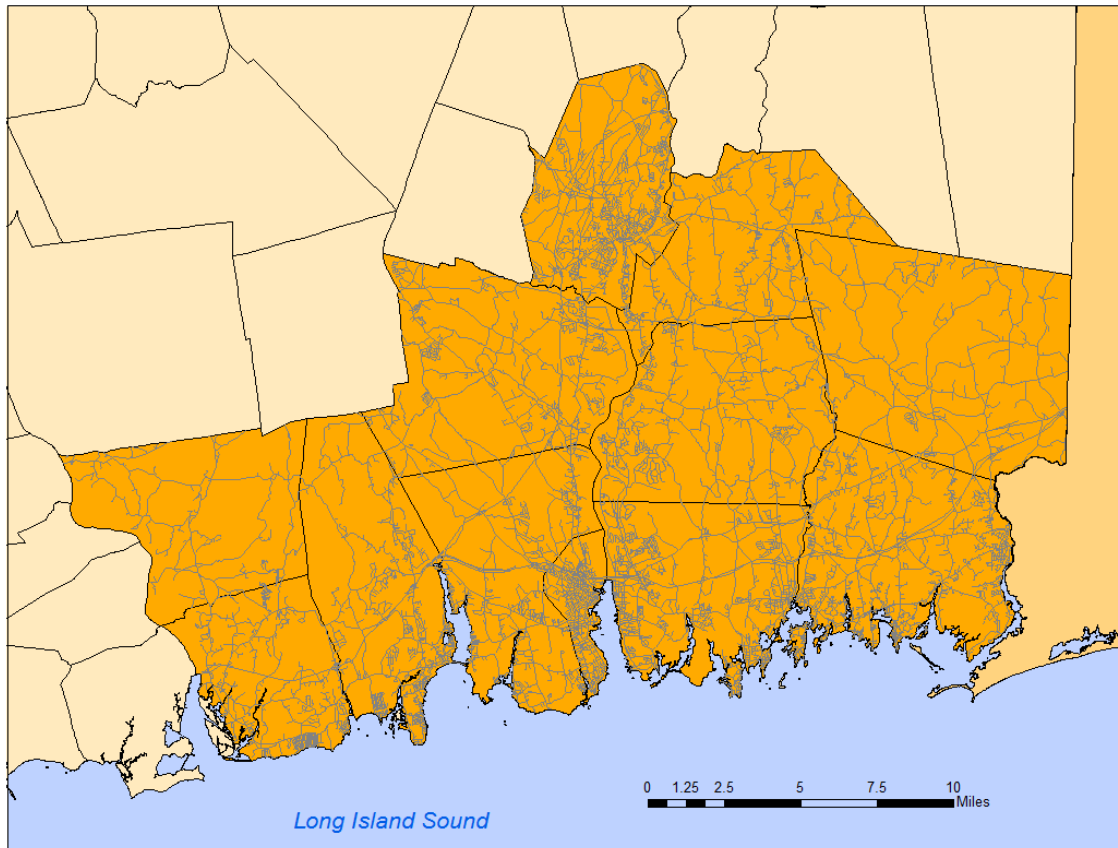


Figure 15 Road Network within Study Area

4.14 SLOSH Flood Data

This research required the use of data representing the threat caused by flooding from tropical cyclone events. SLOSH flood data were incorporated and used in the analyses.

The SLOSH data are hosted by the CTDEP and is supplied in shapefile format (CTDEP, 2008). The data were created by the US Army Corps of Engineers and the Federal Emergency Management Agency for use by the State of Connecticut (CTDEP, 2008).

The polygons of the SLOSH data represent areas that are likely to flood during a storm of a given intensity. Figure 16 shows the locations of these flood-prone areas.

While SLOSH data is usually considered the most well-known and widely used flood dataset in the United States, it is worth pointing out a few of its flaws. The SLOSH polygons are notoriously coarse and do not take into account minute undulations in the terrain. These features can be easily seen with modern, high resolution elevation data, but are not represented in the SLOSH data. Another drawback of SLOSH is the lack of information regarding the degree of the flood within each zone. For example, an area considered to be flooded for a Category 3 storm may have varying degrees of water depth. One location may be covered by only one inch of water, while another within the same SLOSH polygon has 12 inches of water. These limitations play no role in this study and are not detrimental to the results of the Road Intersection Flood Analysis Tool.

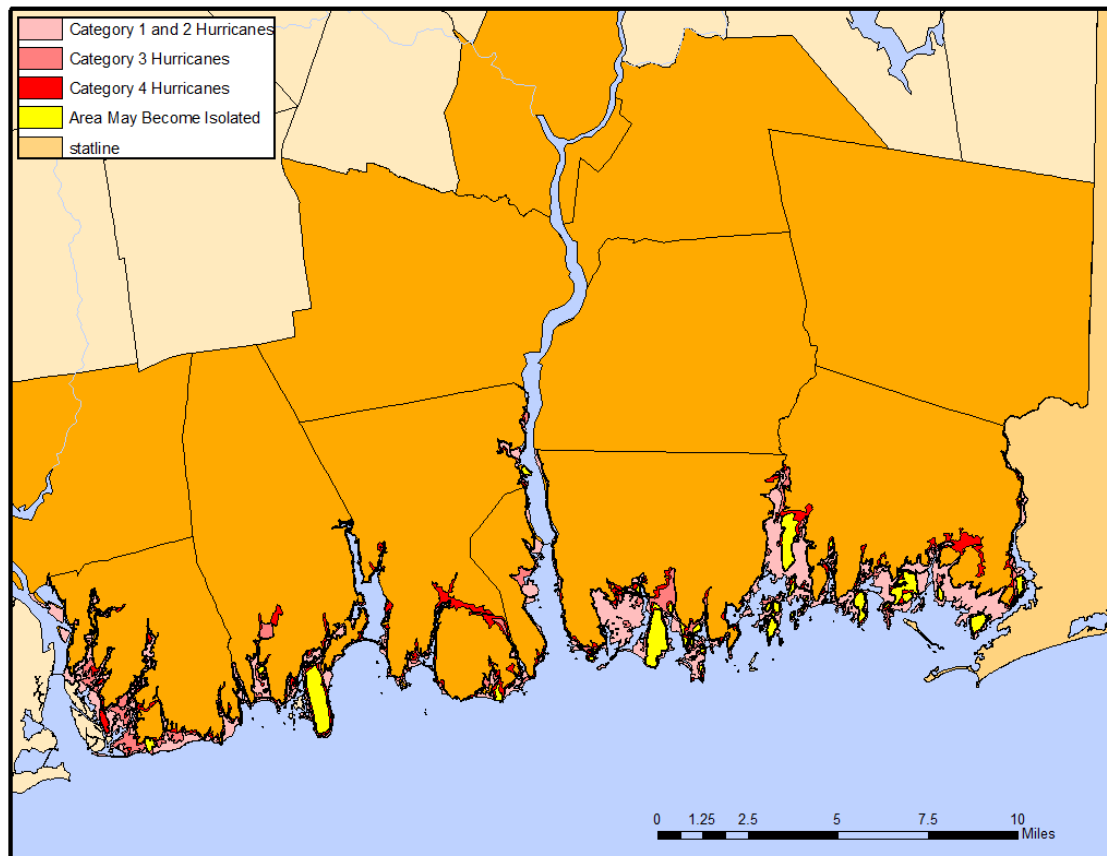


Figure 16 SLOSH Data within Study Area

4.15 LIDAR Elevation Data

LIDAR elevation data were obtained from the University of Connecticut's Center for Land Use Education and Research (CLEAR). This DEM has elevation measurements at 20-foot postings, with a vertical accuracy of 1 foot. The dataset was created for the entire state of Connecticut, but was trimmed to conform to the study area. With the stated resolution, the study area contained approximately 112.8 million data points. The elevation of the study area ranges from 0 feet representing the elevation at sea level to 608 feet (Figure 17).

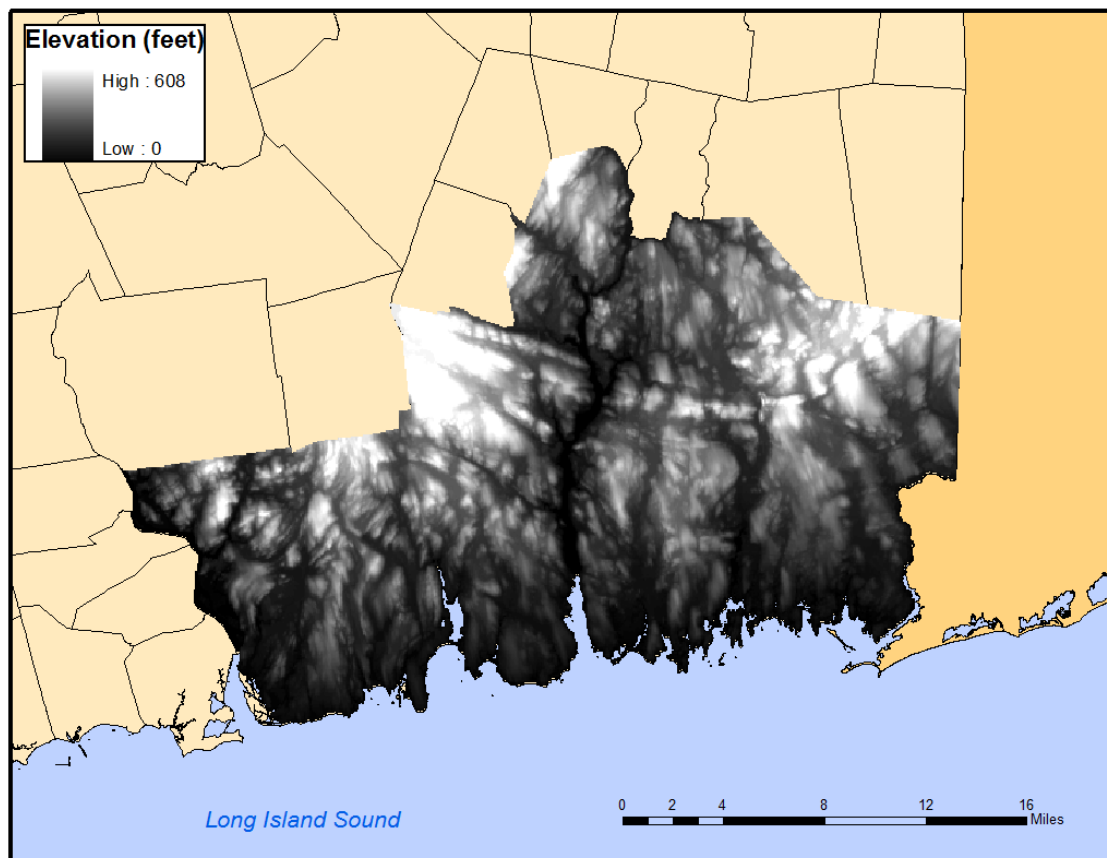


Figure 17 LIDAR Elevation Data within Study Area

4.16 Geographic Base Files

This research required various types of geographic base files. The Connecticut town boundary data were obtained from the Connecticut Department of Environmental Protection. US Zip Code polygons were downloaded from the US Census Bureau's website. This data were trimmed to represent the zip codes within the study area. While not directly used in the analyses, other geographic base files such as a US state dataset and a world country dataset were used to create visual products such as maps. The data were obtained from a CD supplied with Mastering ArcGIS (Price, 2003).

4.17 Summary

Two programming scripts using the open source Python computer language were created. These scripts both used the ability of the Python language to interface with the tools in ESRI's ArcGIS software package. The scripts used the tools in ArcGIS to complete geoprocessing tasks related to emergency management planning regarding hurricanes. The study area for this research consisted of twelve southeastern towns of Connecticut.

The first script determined which road intersections were impassable during a flood event, including a hurricane. Data representing the road network, flood polygons, and other geographic base files were used in this process. The script used the road network shapefile and created nodes at each road intersection which were saved in a point shapefile. SLOSH shapefiles which represent the extent of coastal flooding during a hurricane were used to determine which nodes were affected by floodwaters. This knowledge can improve the effectiveness of emergency responders during a storm.

The second Python script presented in this study attempted to improve on methods defined by Wilmot and Meduri (2005). In their study, hurricane evacuation zones were delineated using Caliper's TransCAD software and its custom scripting language. The research presented in this report performed the same analysis, but using free, open source Python programming and tools in ArcGIS. The product of this script can be used by emergency management personnel to determine where hurricane evacuation zones should be located.

Chapter 5: Discussion

5.1 Introduction

The goal of this study was to explore new methods of geospatial analysis through the development of two automated tools which take advantage of the geoprocessor engine of the ESRI ArcGIS software package. Each of the tools drew from the more than 500 available tools in the ArcGIS geoprocessor to produce new ways of processing spatial data.

The tools developed in the previous section can make hurricane mitigation management easier. Both were written in a common scripting language and executed within the most common GIS software package. The road intersection flood tool successfully identified road intersections that are not underwater during a hurricane which would prevent the passage of motor vehicles. The hurricane evacuation zone tool methodology was translated from a niche scripting language into Python which will grant many more users the access to the method. This tool successfully delineated recommended hurricane evacuation zones based on specific criteria.

5.2 The Road Intersection Flood Tool Case Study Results

The road intersection flood tool was created to identify which road intersections were not underwater during a tropical storm or hurricane. This data can be used on its own or in other analyses, particularly network studies that attempt to route emergency personnel through an area containing floodwaters.

In order to test the tool, a sample dataset was defined. The two key components of the dataset consisted of road data and hypothetical flood data. The resulting shapefile successfully identified which road intersections were not underwater during the hypothetical flood caused by a hurricane.

Figure 18 represents the road intersections that were not considered underwater during a category 1 or 2 hurricane. Under these conditions, 575 of the road intersections were expected to be above water. In this situation, there were 63 fewer intersections that were underwater, or almost 10%.

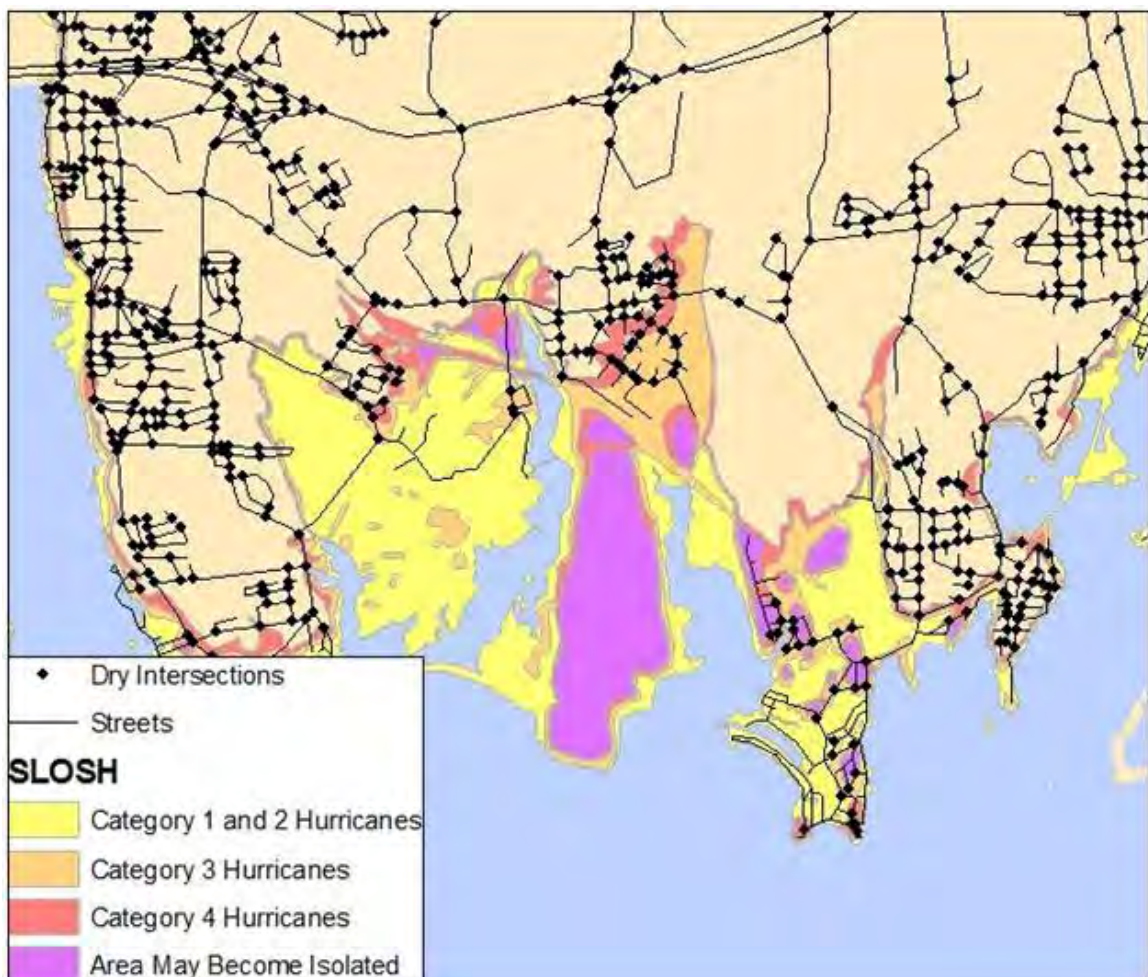


Figure 18 Intersections Above Water After Category 1 or Category Hurricane

Selecting a category 3 hurricane from the tool's parameters produced a smaller set of above water intersections (Figure 19). From the original 638 possible road intersections, only 528 were above water, a difference of 110 intersections (approximately 17%). The reduction in above water intersections is evident in the figure. Only the intersections not covered by the SLOSH polygons and the intersections in the category 4 SLOSH polygons are seen.

Finally, the tool was used to calculate which road intersections would be above water during a category 4 hurricane. This storm event would only allow for 476 dry intersections in the study area (Figure 20). This is 162 underwater intersections, or about 25% of the total number.

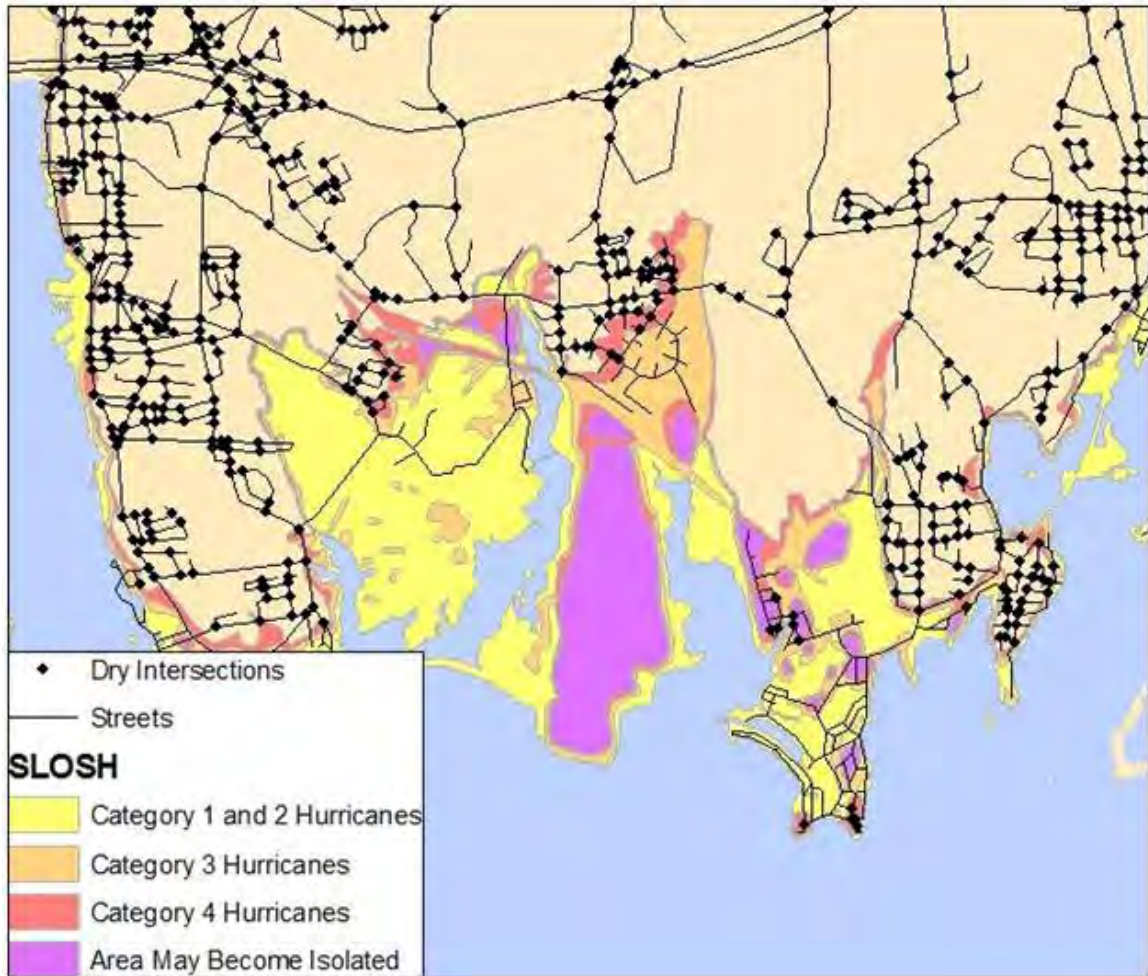


Figure 19 Intersections Above Water After Category 3 Hurricane

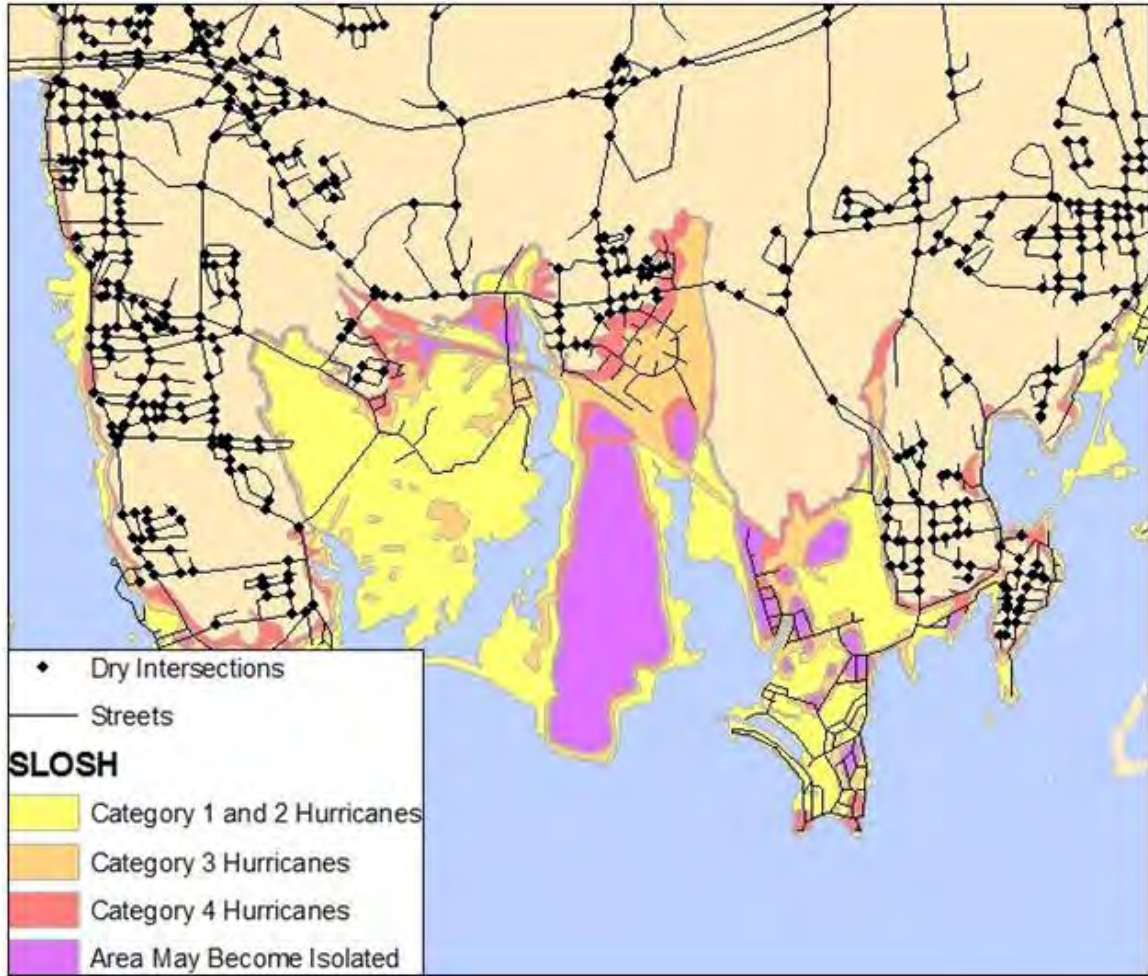


Figure 20 Dry Intersections After Category 4 Hurricane

Figure 21 shows all intersections in this subset of the area of interest. The intersections are color coded to represent how they would be affected by floodwaters during a flood caused by a hurricane.

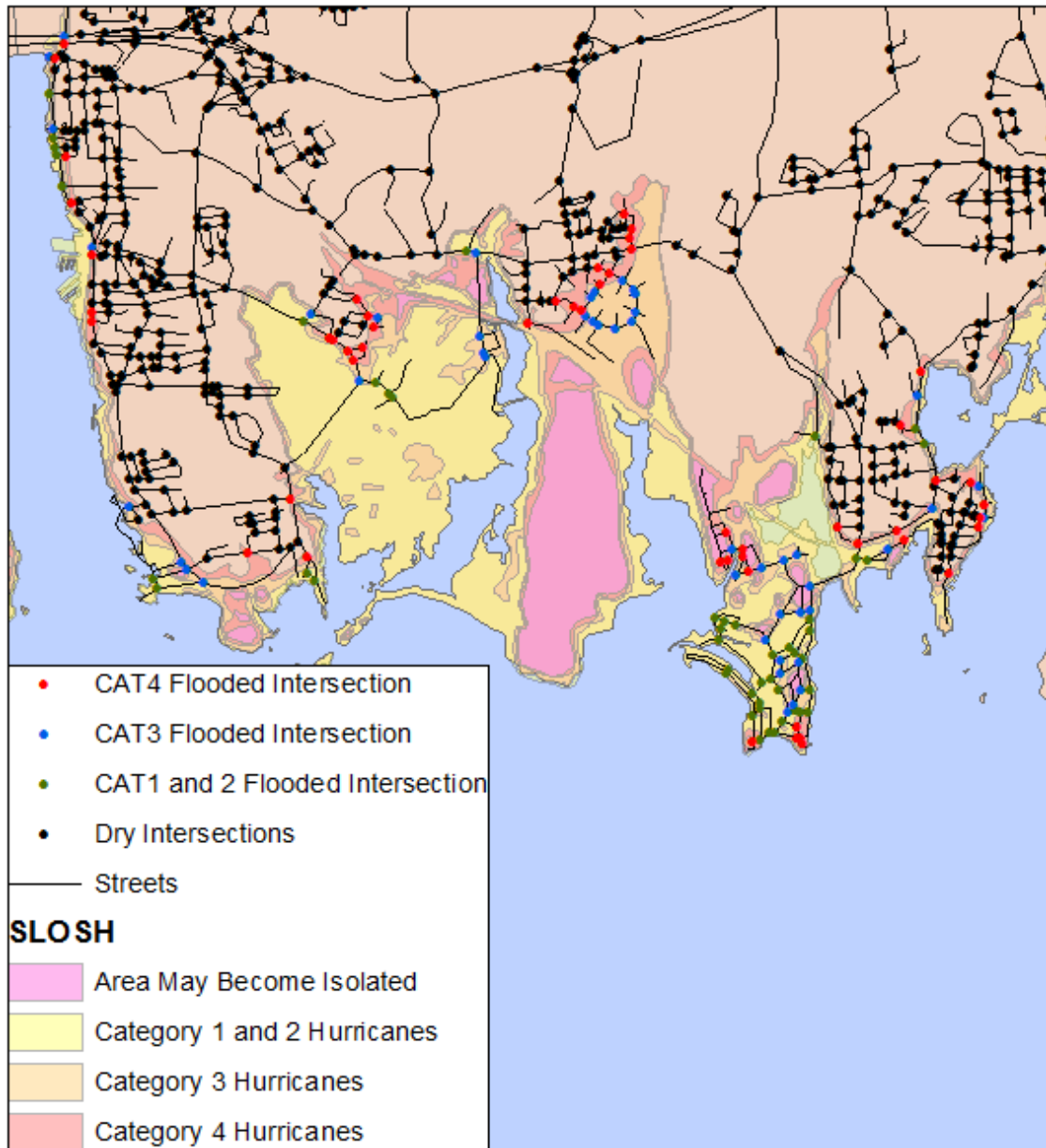


Figure 21 Composite results of the Road Intersection Flood Analysis Tool

5.3 The Hurricane Evacuation Zone Tool Case Study Results

The hurricane evacuation zone script was designed to perform the same analysis outlined by Wilmot and Meduri (2005). The hurricane evacuation zone script presented in this paper successfully employed the scripting capabilities of Python and the tools in the ArcGIS geoprocessor to quickly and accurately delineate hurricane evacuation zones. Refer to Appendix B to view the final hurricane evacuation zone tool script.

Testing the utility of this script was performed using data within the study area outlined in section 4.4.1. The user must supply a polygon file that represents all possible hurricane evacuation zones. As mentioned previously, this is created by combining the town boundary, zip code, and major road data into one polygon dataset. This yielded 300 possible hurricane evacuation zones (Figure 22).

The script iterated 250 times, combining two adjacent evacuation zones with each iteration. After 250 iterations, there were only 50 potential hurricane evacuation zones remaining. Figures 23 – 32 show the progression from all possible hurricane evacuation zones (Figure 22) to only 50 remaining hurricane evacuation zones (Figure 32) in 25 iteration increments. The boundaries of the dissolved zones are represented in red. As the number of iterations increased, the degree of aggregation also increased, which diminished the utility of the data. Since the script is written to create a statistics log for each iteration, the user can view the log to determine exactly how many iterations were needed to achieve an acceptable result. Table 3 shows sample entries from the statistics log. While the statistics log for this script contained only the elevation's average joint standard deviation between the two polygons to be merged, any joint statistic between the

two merging polygons could be determined and logged for further investigation. A plot of the elevation's average joint standard deviation values for all 250 iterations is shown in figure 32. This script calculated the same statistic used as a closing criterion in the Wilmot and Meduri study, but for an arbitrarily high number of iterations. This allowed the investigator to look at all possible iterations once they are processed and determine how many are acceptable.

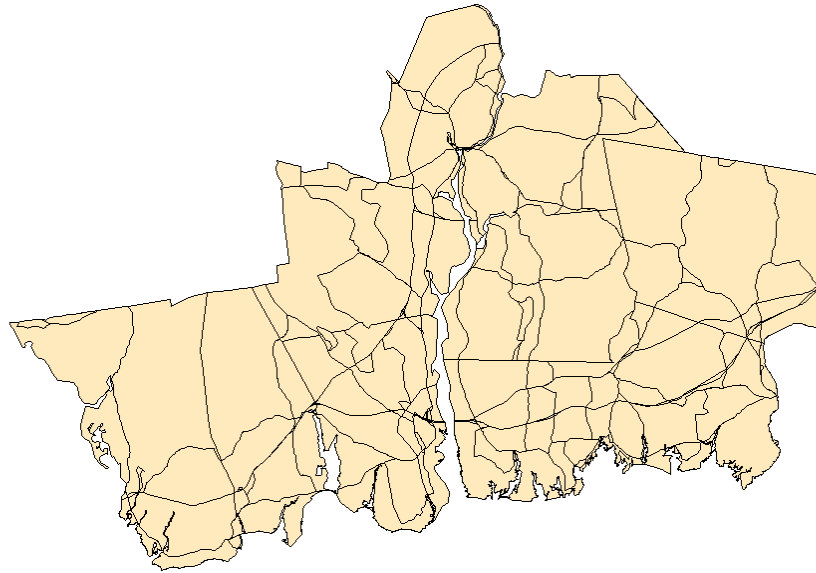


Figure 22 All 300 Possible Hurricane Evacuation Zones

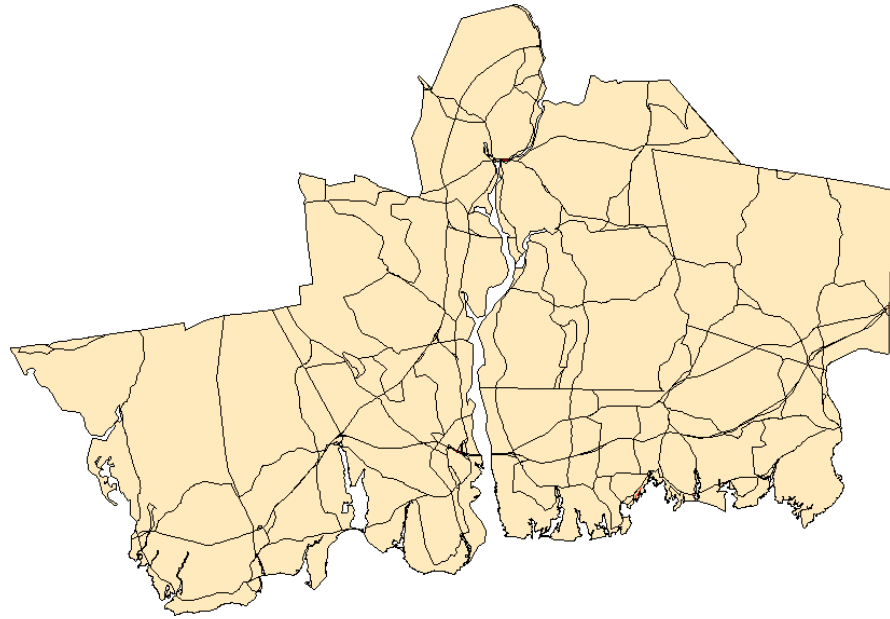


Figure 23 Potential Hurricane Evacuation Zones After 25 Iterations

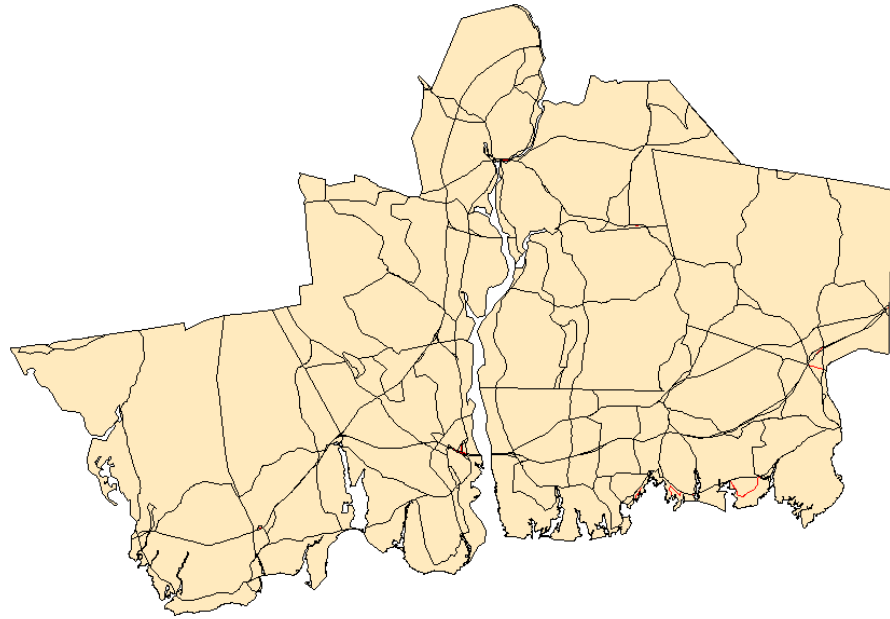


Figure 24 Potential Hurricane Evacuation Zones After 50 Iterations

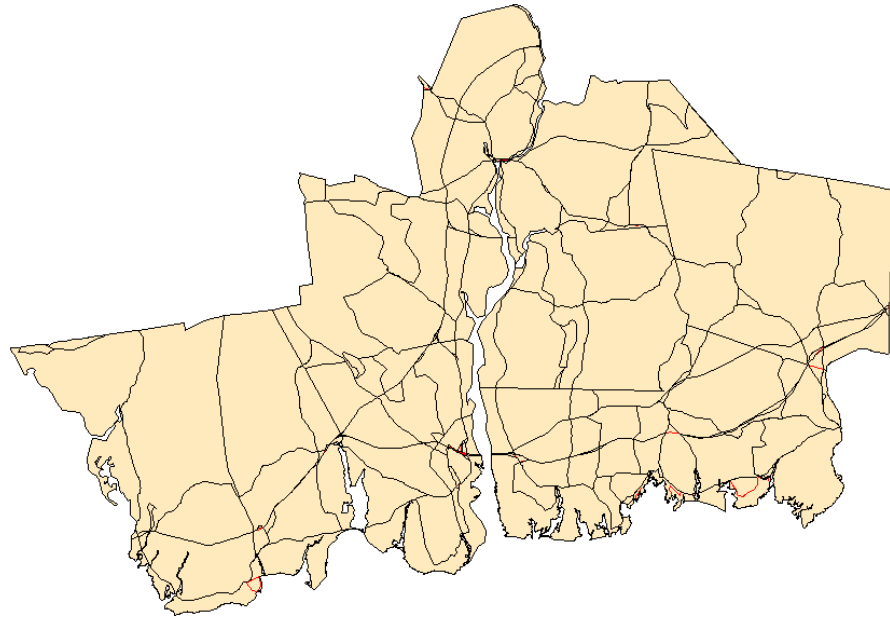


Figure 25 Potential Hurricane Evacuation Zones After 75 Iterations

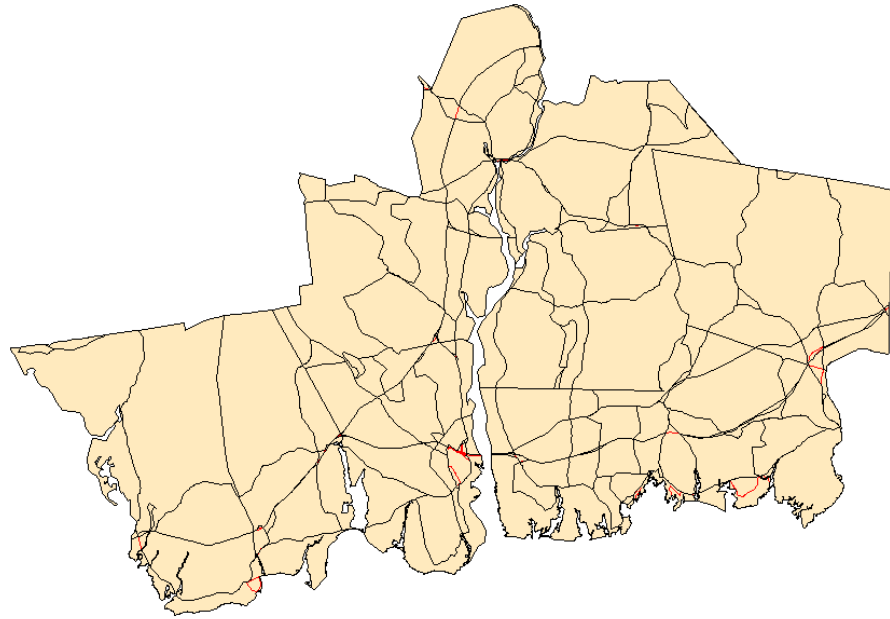


Figure 26 Potential Hurricane Evacuation Zones After 100 Iterations

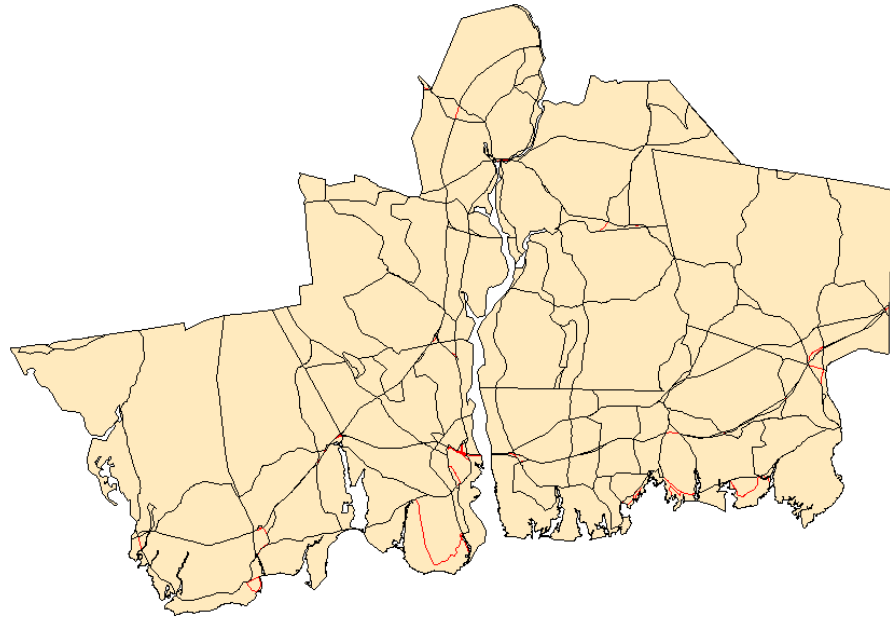


Figure 27 Potential Hurricane Evacuation Zones After 125 Iterations

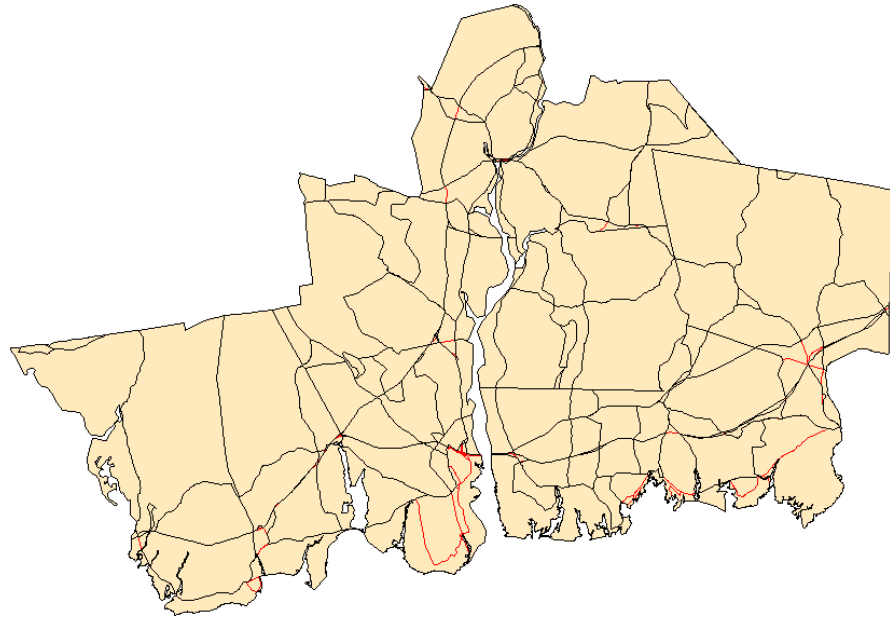


Figure 28 Potential Hurricane Evacuation Zones After 150 Iterations

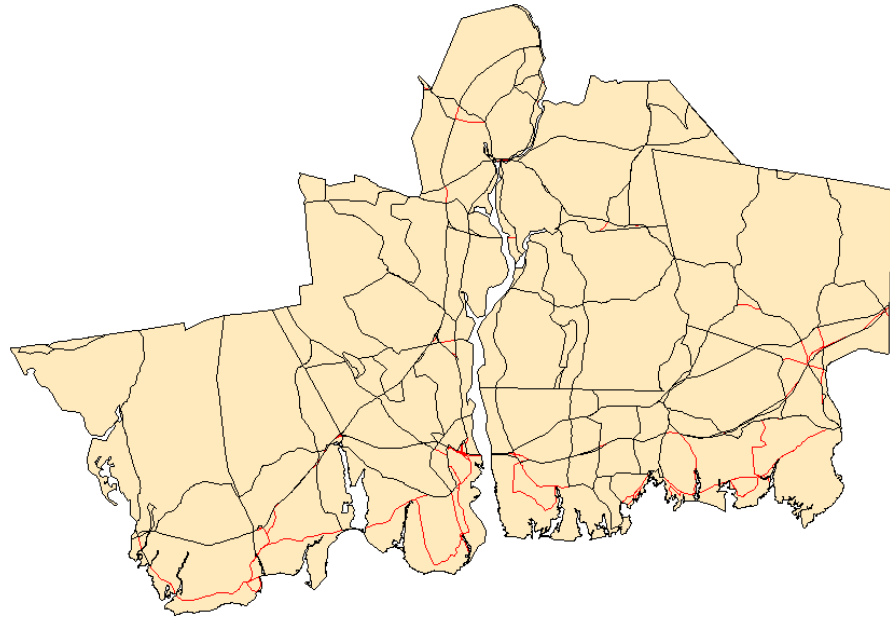


Figure 29 Potential Hurricane Evacuation Zones After 175 Iterations

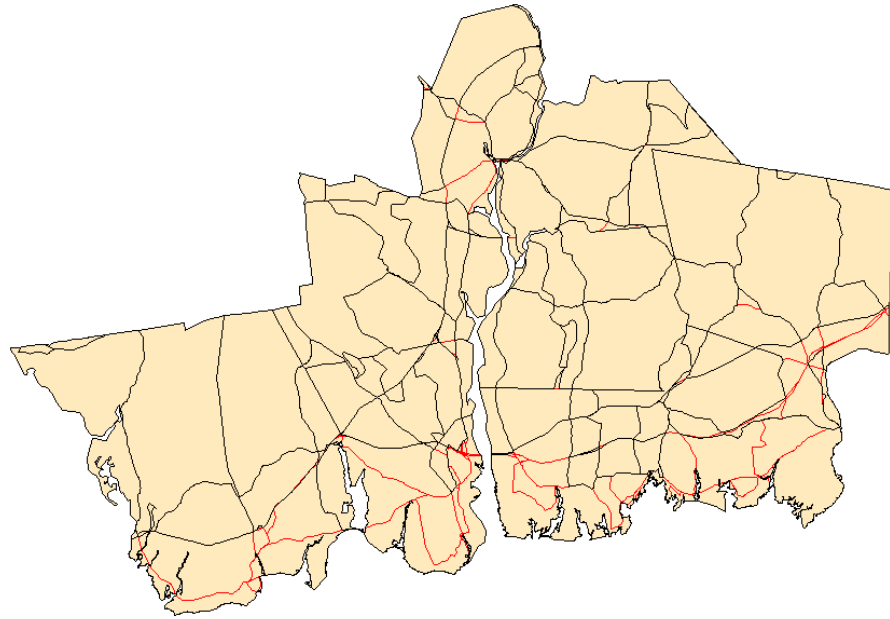


Figure 30 Potential Hurricane Evacuation Zones After 200 Iterations

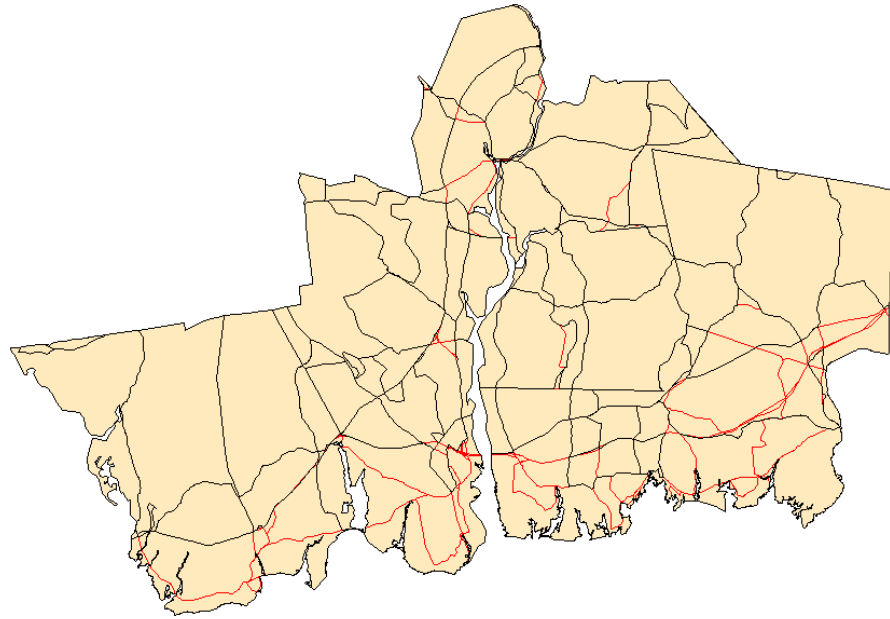


Figure 31 Potential Hurricane Evacuation Zones After 225 Iterations

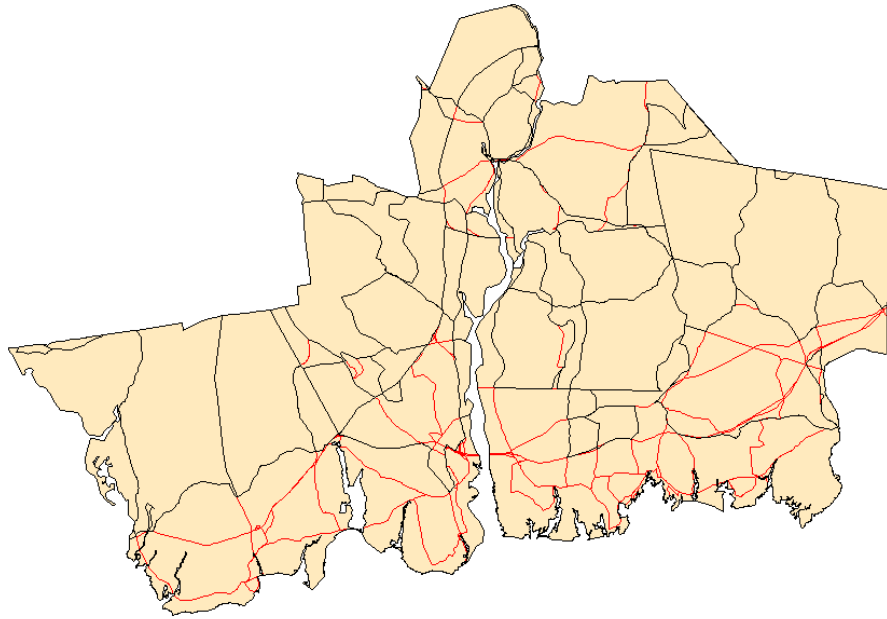


Figure 32 Potential Hurricane Evacuation Zones After 250 Iterations

Iteration	Polygon A ID	Polygon B ID	Elevation Average Joint Standard Deviation
1	96	97	6.327935
2	120	124	5.0653
3	128	133	4.85853
4	79	80	1.8821515
5	83	81	1.156874
6	9996	78	1.957605
7	208	206	7.75147
8	179	177	2.022335
9	9995	9994	1.95
10	181	9992	1.72793

Table 3 Sample Statistics From Output Log

The elevation's average joint standard deviation for the two polygons to be merged was logged in this script for each iteration. The contents of this log were imported into Microsoft Excel where the chosen statistic was plotted. This plot is shown in figure 33; an increasing linear trend is seen in this statistic. This trend shows increasing variability of average elevation values for each of the merging pair of polygons as the number of iterations increase.

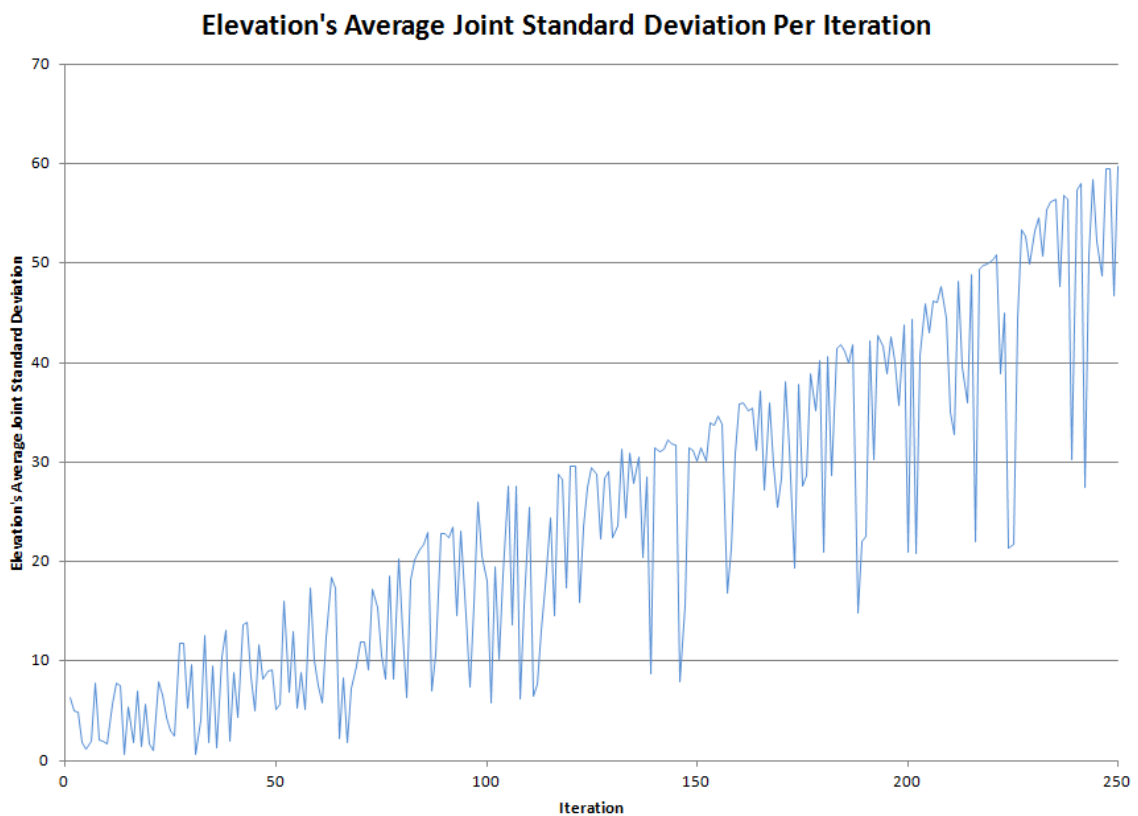


Figure 33 Elevation's Average Joint Standard Deviation Per Iteration for Average Elevation

The investigator must decide how much variability is acceptable and set a threshold of maximum allowable variability. For example, a predetermined maximum allowable joint

standard deviation can be chosen that represents the highest amount of joint standard deviation between evacuation zone polygons that the script decides to be merged. Once the threshold is determined, the number of corresponding iterations can be identified using the statistics log and its shapefile can be used to represent the recommended hurricane evacuation zones for the study area.

Alternatively, the script could be altered slightly to automatically stop processing the hurricane evacuation zones when the threshold (closing criterion) is met. The selected closing criterion must be accurate, otherwise any changes in this value will require the data to be processed with the script again, consuming large amounts of processing time. This method illustrates the benefit of producing a statistics log; all of the data are processed once and the determination of an acceptable closing criterion can be chosen after the data has been processed. This saves the investigator a significant amount of time.

This script was executed on a PC-based computer running Windows Vista. This machine used a Core 2 Duo dual core processor and had 2 gigabytes of RAM. The script required a total of 11 hours and 39 minutes to complete on this computer, or an average of 2 minutes 47 seconds per iteration. While a direct comparison cannot be made due to significant differences in computer hardware and choice of study area, the average time per iteration in the Wilmot and Meduri study is 22 minutes 6 seconds. Figure 34 is a plot of the time taken for each iteration of the script.

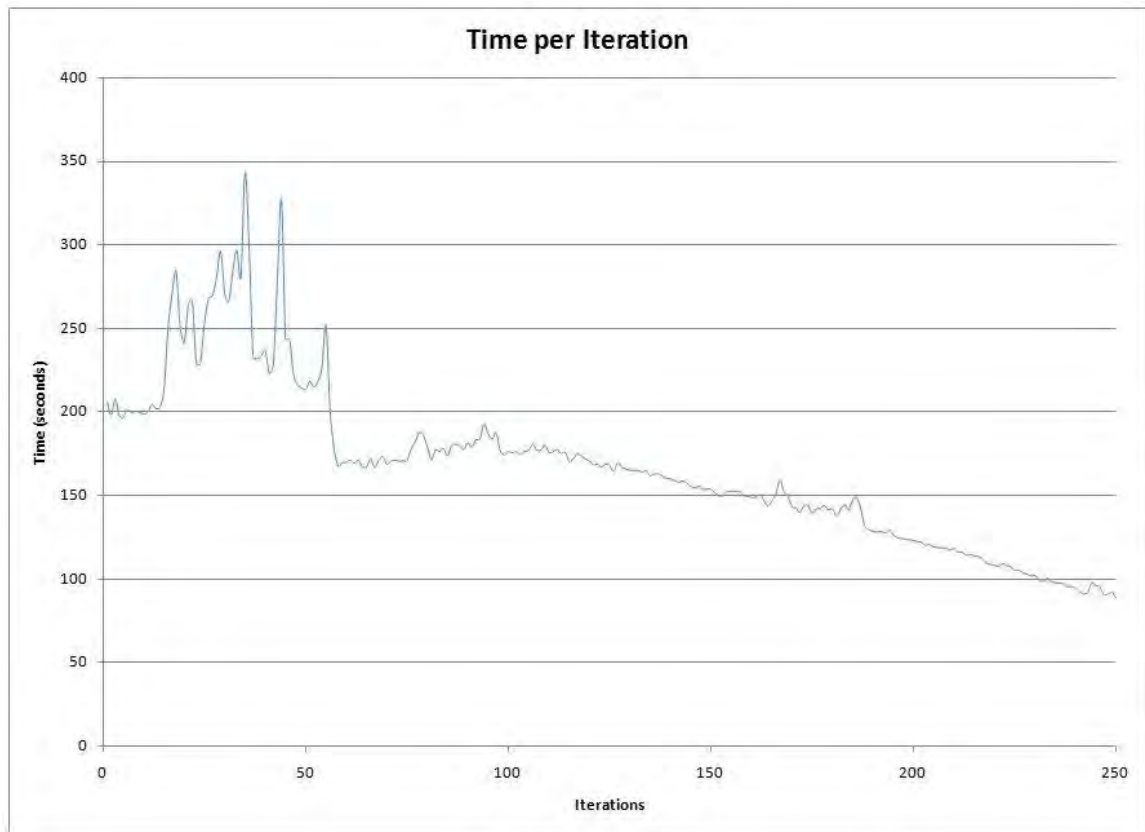


Figure 34 Time Per Iteration

5.4 Summary

The two tools presented here successfully met all research expectations as outlined in the research goals.

The road intersection flood tool determined which road intersections would be inundated during a hurricane. Using road and flood data, the flooded intersections in the study area were successfully identified

The hurricane evacuation zone tool successfully delineated the boundaries of hurricane evacuation zones. The tool uses the road network, zip code areas, elevation data, and

town boundaries to determine where these zones should be located. The script iteratively processed the data and merged two adjacent potential hurricane evacuation zones together based on similarities in their elevation. The script was based on similar research published by Wilmot and Meduri (2005). The research presented here improved on their research by implementing the geoprocessing into a free, open source language based on widely used GIS software. The investigator can sift through the output of this script and determine how many iterations are necessary to arrive at an acceptable outcome.

Chapter 6: Conclusion

6.1 Review of Study

The goal of this study was to develop new automated tools that use the functions of the ArcGIS geoprocessor and are applied to hurricane mitigation. The Road Intersection Flood Tool identified which road intersections were underwater during a hurricane event. The Hurricane Evacuation Tool was originally written in a previous study using proprietary code in the TransCAD scripting language; in this research, it was completely rewritten in Python and is based on the geoprocessor of the ArcGIS platform.

6.2 The Road Intersection Flood Tool

The Road Intersection Flood Tool was able to successfully determine which road intersections were underwater during a hurricane. Emergency managers should find this tool very helpful in decision-making situations where some roads are underwater and emergency personnel needs to be routed around these obstacles.

As the tool was developed, it became evident that its importance as a stand-alone tool would be useful in different ways. The entire script, as previously described, derived the non flooded road intersections during a flood caused by a hurricane. Embedded within this script is a block of code which created a point shapefile using the line intersections of a line shapefile. This ability is not found in any tool within ArcToolbox. ESRI has an add-on module for the ArcGIS software package called Production Line Tool Set, or PLTS. Within the additional tools of PLTS, there is one tool that is similar, but not as flexible as the one presented here. The ESRI tool is called “PLTS Create Points at

Intersections” and it can only calculate the point intersection of two selected line features. It is not able to process multiple sets of intersections at one time. The ability of the script presented here to process an entire dataset could be a great benefit to some users such as transportation geographers wanting to create a point shapefile representing road intersections, rail intersections, or intersections of waterways. One use of the new intersection data would be in a network routing analysis.

The script presented in this research can serve as a foundation for other users to add their own customizations. For example, if the road intersection points need to have populated attributes which represent characteristics such as number of lanes or street name information, this could be an easy update! The flexibility of the script allows it to be tweaked to suit the needs of the user and the specific application.

6.3 Hurricane Evacuation Zone Tool

The purpose of the hurricane evacuation tool was to delineate suggested hurricane evacuation zones within a study area based on differences in elevation. This study proved that the methodology used by the original investigators can be successfully ported to the ubiquitous ArcGIS geoprocessor. The method can now be used by many more people. It also demonstrated that other geoprocessing tools written in different languages can be rewritten in Python, the preferred scripting language of ArcGIS.

The hurricane evacuation zone tool demonstrated the benefit of converting older geoprocessing scripts into the Python language. ESRI is publically touting Python as their preferred scripting language and has ensured that Python is completely compatible

with the ArcGIS geoprocessor. Because of this partnership between the two, users can produce efficient geoprocessing scripts using the easy to learn Python language. This is evident in the hurricane evacuation zone script; the average time per iteration was much less using this script than the one Wilmot and Meduri created using the TransCAD scripting language. This script and its corresponding ArcToolbox-compatible toolbox file can be easily used by others.

Instead of running iterations of this script until a specific closing criterion was attained, this script ran a fixed, arbitrarily high number of iterations. This provided closing criteria data for many iterations which was evaluated after the script completes. The Wilmot and Meduri script is programmed to stop iterations when a particular closing criterion is achieved. Unfortunately if the investigator determines the closing criterion was either too high or too low, the whole script has to be run again. The script in this study benefits the user in that the script only needs to be run once and the resulting closing criterion statistic can be evaluated afterwards which can save large amounts of time since the script took many hours to complete.

6.4 Future Studies

As technology advances, demanding computational processes similar to what are demonstrated in this study will be easier to produce and execute. The Python scripting language, ESRI's ArcGIS software, and computer hardware are products of modern technology. As the fields of computer science and geography advance, improvements in both software packages will lead to more efficient processing and new tools at the user's disposal.

One trend seen today is the use of graphical processing units (GPUs) in lieu of a computer's central processing unit (CPU). The architecture of the modern GPU is very different of a CPU and enables advanced floating point calculations previously attainable only on supercomputers (Folding@home, 2009a). These advances in hardware may result in faster processing times for geospatial data and enable the user to use highly complex geoprocessing tools. GPUs are already being used to process certain types of data. The Folding@Home initiative at Stanford University uses a distributed computing network to take advantage of the participants' GPUs to calculate the complex folding patterns of proteins (Folding@home, 2009b). Other researchers have used the GPU power in modern video game consoles to partially crack the previously untouched WPA wireless encryption scheme (Slashdot, 2009), crack secure passwords (Computerworld, 2007), and even circumvent the method used online that certifies that a website is secure (AntiVirusConnection, 2007). If the power from these GPUs is harnessed to process complex geospatial data, the limits of current geoprocessing techniques will dissolve. These advances will make complex processing tasks such as the ones found in this study a trivial matter.

Both scripts utilized a custom toolbox within the ArcToolbox, the interface used in the ArcGIS software which enables the use of geoprocessing tools. The custom toolboxes allows a user of ArcGIS who is not versed in Python scripting to process data with the Python script using an graphical user interface (Figures 3 and 7). Like the scripts themselves, the toolbox interface for the scripts are entirely customizable. It is worth noting if any future changes are made within the script related to the variables used in the corresponding toolbox, the toolbox itself will need to be updated to reflect such changes.

The user must also be mindful of balancing the number of options within the toolbox interface of the scripts. If the toolbox contains too few options, the user will have less of an opportunity to provide pertinent data and the scripts utility will be diminished. If too many options are given in the toolbox, the user may be overwhelmed and the original purpose of providing the easy to use interface to the Python script is negated.

6.5 Future Studies: Road Intersection Flood Analysis Tool

The road intersection flood script can serve as a foundation tool where others can incorporate custom input data or determine what the output will be. For example, a user could tweak the code in such a manner that allows multiple sets of input data to be combined before the intersections are calculated. The user may opt to only output the flooded intersections. These and other changes can be incorporated by using their own customizations to the script.

As it is currently written, the tool is stand-alone and will process data as described. If needed, this tool could be converted into a Python module. Python modules are a way to compartmentalize complex tasks into simple-to-use commands. The new module could be imported into other scripts where the flooded intersections need to be determined. The portion of the current tool which creates a point shapefile of the line intersections can be pulled out and converted to a module if the user wishes only to calculate all of the road intersections. The flexibility of the Python language gives the user practically limitless options when creating tools such as these.

The current script only reviews road intersections. An additional study may be warranted to look at the road segments themselves to ascertain which ones may be affected by floodwaters. Situations may arise where you have one road segment solely serving many nodes. The nodes themselves may be above the flood threshold, but this important connecting segment may not be. In this case, the nodes served by the flooded road segment should be considered unserviceable.

6.6 Future Studies: Hurricane Evacuation Zone Tool

The hurricane evacuation zone tool contained many parts that work together to achieve the research goals in this study. It is based on a method by Wilmot and Meduri (2005) in which geospatial data are used in order to calculate where hurricane evacuation zones should be located. If future users decide to alter the mathematical formulas used in this script which determine how the adjacent polygons are merged, they have the ability to change something as simple as the percentile used or as complex as retooling the entire mathematics section of the script to meet their needs.

The script is currently able to record one closing criterion statistic per iteration. This statistic is the one used by Wilmot and Meduri (2005). If a user determines another closing criterion is advantageous or decides that multiple closing criteria are necessary, these calculations can be incorporated into the current script and saved in the output text file. The output text file can be imported into a spreadsheet or even processed by another script to view and interpret the closing criteria statistics.

Each iteration of the script outputs a polygon file of the resulting hurricane evacuation

zones. This method may consume too much hard disk space or the output may not be necessary. A user may decide to disable the output as it is written. Another option would be to only create the polygon shapefile on every fifth or tenth iteration. The user has the flexibility to decide what output will suit their needs.

Literature Cited

- Antivirus Connection. 2007. <http://www.antivirusconnection.com/index.php/avc-news/200-sony-ps3-consoles-undertaken-to-crack-secure-site-certification> Accessed 09/21/2008.
- Artima, Inc. 2009. <http://www.artima.com/intv/pythonP.html> Accessed 08/05/2009.
- Assilzadeh, H., and Y. Gao. 2009. "Oil Spill Emergency Response Mapping for Coastal Area Using SAR Imagery and GIS." *Oceans* September 2008, pp. 1-6.
- Bettinger, Pete, Krista Merry, and Jeff Hepinstall. 2009. "Average Tropical Cyclone Intensity Along the Georgia, Alabama, Mississippi, and North Florida Coasts." *Southeastern Geographer* 49(1), pp. 50-66.
- Boose, Emery R., Kristen E. Chamberlin, and David R. Foster. 2001. "Landscape and Regional Impacts of Hurricanes in New England." *Ecological Monographs* 71(1), pp. 27-48.
- BrainTrack.com. 2012. <http://www.braintrack.com/college/u/university-of-connecticut-avery-point> Accessed 01/14/2012.
- Butler, Howard. 2005. <http://www.esri.com/news/arcuser/0405/files/python.pdf> Accessed 08/29/2009.
- Cai, Hubo, William Rasdorf, and Chris Tilley. 2007. "Approach to Determine Extent and Depth of Highway Flooding." *Journal of Infrastructure Systems* 13(2), pp. 157-167.
- Caliper Corporation. 2005. "Routing and Logistics with TransCAD 4.8." Reference manual.
- Case, Robert A. 1986. "Annual Summary: Atlantic Hurricane Season of 1985." *Monthly Weather Review* 114, pp. 1390-1405.
- Chen, Xuwei, John W. Meaker, and Benjamin Zhan. 2006. "Agent-Based Modeling and Analysis of Hurricane Evacuation Procedures for the Florida Keys." *Natural Hazards* 38, pp. 321-338.
- Chiu, Yi-Chang, Hong Zheng, Jorge Villalobos, and Bikash Gautam. 2007. "Modeling no-notice mass evacuation using a dynamic traffic flow optimization model." *IIE Transactions* 39, pp. 83-94.
- Computerworld. 2007. <http://computerworld.co.nz/news.nsf/scrt/C50D36EFEC2B482ACC2573A00070EF83> Accessed 09/07/2009.

- Connecticut Department of Emergency Management and Homeland Security (CT DEMHS). 2006. "State of Connecticut Natural Disaster Plan 2006."
- Connecticut Department of Environmental Protection (CT DEP). 2004. "Natural Hazard Mitigation Plan For 2004 – 2007."
- Connecticut Department of Environmental Protection (CT DEP). 2008.
<http://www.ct.gov/dep/cwp/view.asp?a=2698&q=322898> Accessed November 23, 2008.
- Courant.com. 2012. <http://www.courant.com/business/hc-millstone.artfeb10,0,1131539.story> Accessed 01/14/2012.
- Cova, Thomas J. and Justin P. Johnson. 2003. "A Network Flow Model for Lane-Based Evacuation Routing." *Transportation Research Part A* 37, pp. 579-604.
- Dowhań, L., Wymysłowski, A., & Dudek, R. 2009. Multi-objective decision support system in numerical reliability optimization of modern electronic packaging. *Microsystem Technologies*, 1-7.
- Dunn, Christine E. and David Newton. 1992. "Optimal Routes in GIS and Emergency Planning Applications." *Area* 24(3), pp. 259-267.
- Environmental Systems Research Institute, Inc. (ESRI). 2009.
http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=About_getting_started_with_writing_geoprocessing_scripts Accessed August 24, 2009.
- Environmental Systems Research Institute, Inc. (ESRI). 2012.
<http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//002z00000001000000> Accessed June 30, 2012.
- Folding@home. 2009a. <http://folding.stanford.edu/English/FAQ-ATI#ntoc2> Accessed 07/07/2009.
- Folding@home. 2009b. <http://folding.stanford.edu/English/Main> Accessed 07/07/2009.
- Foster, David R., and Emery R. Boose. 1992. "Patterns of Forest Damage Resulting From Catastrophic Wind in Central New England, USA." *Journal of Ecology* 80 (1), pp. 79-98.
- Foxwoods. 2012. <http://www.foxwoods.com/Careers.aspx> Accessed 01/14/2012.
- Francis, Arthur A. 1998. "Remembering the Great New England Hurricane of 1938." <http://www2.sunysuffolk.edu/mandias/38hurricane> Accessed March 27, 2007.

- Greenwood, David J and Darryl J Hatheway. 1996. "Assessing Opal's Impact." *Civil Engineering* 66(1), pp. 40-43.
- GrotonNewLondonAirport.com. 2012.
http://www.grotonnewlondonairport.com/ECONOMIC_IMPACT.pdf Accessed 01/14/2012.
- Hitchcock, F. 1941. "The Distribution of a Product from Several Sources to Numerous Localities." *Journal of Mathematics and Physics* 20, pp. 334-230.
- Hobeika, Antoine G. and Bahram Jamei. 1985. "MASSVAC: A Model for Calculating Evacuation Times Under Natural Disasters." *Simulation Series* 15(1), pp. 23-28.
- Jarmin, Ron S. and Javier Miranda. 2009. "The Impact of Hurricanes Katrina, Rita and Wilma on Business Establishments." *Journal Business Valuation and Economic Loss Analysis* 4(2), Article 7.
- Lindell, Michael K. and Carla S. Prater. 2007. "Critical Behavioral Assumptions in Evacuation Time Estimate Analysis for Private Vehicles: Examples from Hurricane Research and Planning." *Journal of Urban Planning and Development* 133 (1), pp. 18-29.
- LinkedIn. 2012. <http://www.linkedin.com/companies/mohegan-sun> Accessed 01/14/2012.
- Louisiana Department of Homeland Security and Emergency Preparedness. 2012.
<http://gohsep.la.gov/hurricanerelated/hurricanecategories.htm> Accessed 05/29/2012.
- MilitaryNewcomers.com. 2012.
<http://www.militarynewcomers.com/NEWLONDON/Guide.html> Accessed 01/14/2012
- National Climatic Data Center (NCDC). 2006. "Climate of 2005: Atlantic Hurricane Season." <http://www.ncdc.noaa.gov/oa/climate/research/2005/hurricanes05.html> Accessed: March 25, 2007.
- National Hurricane Center (NHC). 2007a. "Tropical Weather Summary – 2005 Web Final."
http://www.nhc.noaa.gov/archive/2005/tws/MIATWSAT_nov_final.shtml?
 Accessed: March 25, 2007.
- National Hurricane Center (NHC). 2007b. "SLOSH Model."
<http://www.nhc.noaa.gov/HAW2/english/surge/slosh.shtml> Accessed: December 19, 2007.

- National Hurricane Center (NHC). 2007c. "Storm Surge."
http://www.nhc.noaa.gov/HAW2/english/storm_surge.shtml Accessed:
 December 19, 2007.
- National Oceanic and Atmospheric Administration (NOAA). 2006. "NOAA Reviews
 Record-Setting 2005 Atlantic Hurricane Season: Active Hurricane Era Likely to
 Continue." <http://www.noaanews.noaa.gov/stories2005/s2540.htm> Accessed:
 March 25, 2007.
- de Palma, A. and F. Marchal. 2000. "Dynamic traffic analysis with static data: some
 guidelines with an application to Paris." *Transportation Research Record: Journal
 of the Transportation Research Board.* 1752(2001), pp. 76-83.
- Peeta, S. and A. K. Ziliaskopoulos. 2001. "Foundations of dynamic traffic assignment:
 the past, the present and the future." *Networks and Spatial Economics*, 1(3/4), pp.
 233–265.
- Pfizer. 2012. http://www.pfizer.com/research/rd_works/rd_locations.jsp Accessed
 01/14/2012.
- Pradal, C., Boudon, F., Nougier, C., Chopard, J., & Godin, C. 2009. PlantGL: A python-
 based geometric library for 3D plant modelling at different scales. *Graphical
 Models*, 71(1), 1-21.
- Price Maribeth. 2003. Mastering ArcGIS. New York, NY: McGraw-Hill.
- Python Software Foundation. 2009. <http://www.python.org> Accessed 08/05/2009.
- O'Boyle, N. M., & Hutchison, G. R. 2008. Cinfony - combining open source
 cheminformatics toolkits behind a common interface. *Chemistry Central Journal*,
 2(1).
- Slashdot. 2009. <http://it.slashdot.org/article.pl?sid=08/11/06/1546245> Accessed
 09/12/2009.
- Southworth, Frank. 1991. "Regional Evacuation Modeling: A State-of-the-Art Review."
Center for Transportation Analysis. Department of Energy, Oak Ridge National
 Laboratory. March 1991.
- United States Coast Guard Academy. 2011. <http://www.cga.edu/display1.aspx?id=340>
 Accessed 06/18/2011.
- United States Army Corps of Engineers (USACE), New England Division. 1988.
 Connecticut Hurricane Evacuation Study.
- United States Census Bureau. 2009. <http://www.census.gov> Accessed 06/14/2009.

- Wilmot, Chester and Nandagopal Meduri. 2005. "Methodology to Establish Hurricane Evacuation Zones." *Transportation Research Record* 1922, pp. 129-137.
- World Meteorological Organization (WMO). 2006. "Global Guide to Tropical Cyclone Forecasting." http://www.bom.gov.au/bmrc/pubs/tcguide/global_guide_intro.htm
Accessed: March 29, 2007.

Appendix A: Road Intersection Flood Analysis Tool Python Script

```
1  ##Initiate Geoprocessor
2  #Import ArcGIS and sys module
3  import arcgisscripting, sys
4  #Create geoprocessor
5  gp = arcgisscripting.create()
6
7  #Allow overwriting of output data
8  gp.OverwriteOutput = 1
9
10 #Create Script Arguments
11 TBFile = sys.argv[1] #User-Defined variable 1
12 roads = sys.argv[2] #User-Defined variable 2
13 slosh = sys.argv[3] #User-Defined variable 3
14 TownLst = sys.argv[4] #User-Defined variable 4
15 SloshVal = sys.argv[5] #User-Defined variable 5
16 wksp = sys.argv[6] #User-Defined variable 6
17
18 # environment settings...
19 gp.workspace = wksp #Sets active workspace to the location specified in the variable wksp
20 gp.OutputCoordinateSystem = TBFile #Sets coordinate system to CT Town Boundary File
21
22 #Add necessary toolboxes
23 gp.AddToolbox("C:/Program Files/ArcGIS/ArcToolbox/Toolboxes/Data Management Tools.tbx")
24
25 ##User Selects CT Town Boundary
26 gp.MakeFeatureLayer_management (TBFile, "TBLayer") #Converts town boundary file to a layer
27 gp.SelectLayerByAttribute_management ("TBLayer", "NEW_SELECTION", TownLst) #Selects
   user-defined town boundaries
28
29 ##Selected Towns Clip Road Data
30 gp.AddMessage("Clipping Roads. . .") #Outputs message when running as a script in ArcToolbox
31 gp.Clip_analysis (roads, "TBLayer", "roads_clip.shp") #Clip roads using selected town
   boundaries
32 gp.MakeFeatureLayer_management ("roads_clip.shp", "roads_clip_lyr") #Converts town
   boundary file to a layer
33 gp.AddMessage("Clip Complete!") #Outputs message when running as a script in ArcToolbox
34
35 ##Clipped Roads are Converted to Network (Point Data)
36 outputName = "intersections.shp"
37
38 # dissolve roads into single-part lines...
39 gp.AddMessage("Dissolving roads. . .") #Outputs message when running as a script in
   ArcToolbox
40 gp.Dissolve_management ("roads_clip_lyr", "roads_dis.shp", "", "", "SINGLE_PART")
   #Dissolves roads into a single part feature set
41 gp.AddMessage("Dissolve Complete!") #Outputs message when running as a script in ArcToolbox
42
43 # create dictionary for line vertex coordinates...
44 ptDct = {}
45
46 # create cursor and get row for dissolved roads...
47 cur = gp.SearchCursor("roads_dis.shp")
48 row = cur.next()
49
50 # set counter (for point ID value)...
51 cnt = 0
52
53 # for each line feature...
54 while row: #Begin while statement
55
56     geom = row.GetValue('Shape') #sets 'geom' variable equal to the geometry in the
   'Shape' field
57
58     cnt +=1 #Advances counter by 1
59     firstPt = geom.FirstPoint.split(" ") #Splits X and Y values
60     X = float(firstPt[0]) #sets X equal to the X coordinate
```

```

61     Y = float(firstPt[1]) #sets Y equal to the Y coordinate
62     ptDct[cnt] = [X,Y] #Saves X and Y values in the ptDct dictionary using the counter
    value as a key
63
64     cnt +=1 #Advances counter by 1
65     lastPt = geom.LastPoint.split(" ") #Splits X and Y values
66     X = float(lastPt[0]) #sets X equal to the X coordinate
67     Y = float(lastPt[1]) #sets Y equal to the Y coordinate
68     ptDct[cnt] = [X,Y] #Saves X and Y values in the ptDct dictionary using the counter
    value as a key
69
70     # get next line feature...
71     row = cur.next()
72
73 del cur,row #Deletes the cursor and row variables
74
75 # point ID list...
76 keys = ptDct.keys() #Extracts keys from ptDct and saves them to a list called keys
77
78 # list for nodes...
79 nodeList = [] #Creates empty list called nodeList
80
81 # for each point ID...
82 for key1 in keys: #Begin for loop
83     # get X and Y coordinates...
84     X1 = ptDct[key1][0] #Extracts X from first key in ptDct dictionary
85     Y1 = ptDct[key1][1] #Extracts Y from first key in ptDct dictionary
86
87     # for each point ID...
88     for key2 in keys: #Begin nested for loop
89         # if points do not have same ID...
90         if key1 <> key2: #Conditional statement true if the two keys are different
91             # get X and Y coordinates...
92             X2 = ptDct[key2][0] #Extracts X from second key in ptDct dictionary
93             Y2 = ptDct[key2][1] #Extracts Y from second key in ptDct dictionary
94
95             # if X and Y coordinates are the same for both points
96             # then point is a node and point is not already in nodeList...
97             if X1==X2 and Y1==Y2 and key2 not in nodeList: #Conditional statement true if
the X values are equal, the Y values are equal, and key2 is not located in nodeList
98
99                 # add ID to nodeList...
100                 nodeList.append(key2) #Adds key2 to nodeList list
101
102                 # CHECK IF POINT IS ALREADY IN NODELIST
103                 # for each point in nodeList...
104                 for key3 in nodeList: #Begin for loop
105                     # if points do not have same ID...
106                     if key2 != key3: #Conditional statement true if key2 is not equal to key3
107                         # get XY coordinates of point...
108                         X3 = ptDct[key3][0] #Extracts X from key3 inptDct dictionary
109                         Y3 = ptDct[key3][1] #Extracts Y from key3 inptDct dictionary
110
111                         # if XY coordinates are the same...
112                         if X2==X3 and Y2==Y3: #Conditional statement true if X values are
the same and Y values are the same
113                             # remove point from nodeList...
114                             nodeList.remove(key2) #Removes value of key2 from nodeList list
115                             # break out of loop...
116                             break
117
118
119 # create new point feature class...
120 gp.CreateFeatureClass(wksp, outputName,"point")
121
122 # create insert cursor for new point feature class...

```

```

123 inCur = gp.InsertCursor(outputName)
124
125 # for each node...
126 for node in nodeList: #Begins for loop
127     # get XY coordinates...
128     X = ptDct[node][0] #Extracts X value of 'node' in inptDct dictionary
129     Y = ptDct[node][1] #Extracts Y value of 'node' in inptDct dictionary
130
131     # create new point object and set coordinates...
132     pnt = gp.CreateObject("point") #Creates point object
133     pnt.X = X #Sets X coordinate
134     pnt.Y = Y #Sets Y coordinate
135
136     # create new row and set 'Shape' value...
137     newRow = inCur.NewRow() #Creates new row
138     newRow.SetValue('Shape',pnt) #Sets shape field
139     newRow.SetValue('ID',node) #Sets ID field
140
141     # insert new row...
142     inCur.InsertRow(newRow) #Creates new row
143
144     del pnt,X,Y,newRow #Deletes variables pnt, X, Y, and newRow
145
146 del inCur,nodeList,ptDct #Deletes variables inCur, NodeList, and ptDct
147
148 gp.MakeFeatureLayer_management (outputName, "intersections_layer") #Converts feature class
to a layer
149
150 ##User Selects Storm Intensity from SLOSH Data
151 SloshVal = int(SloshVal) #Converts SLOSH argument from a string to a number
152 if SloshVal == 1: #True if Category 1 hurricane selected by user
153     gp.AddMessage("Category 1 Hurricane Selected") #Outputs message when running as a
script in ArcToolbox
154     slosh_expression = "Surge_Area = 'Category 1 and 2 Hurricanes' or Surge_Area = 'Area
May Become Isolated'" #SQL statement used to designate Category 1 hurricane
155 elif SloshVal == 2: #True if Category 2 hurricane selected by user
156     gp.AddMessage("Category 2 Hurricane Selected") #Outputs message when running as a
script in ArcToolbox
157     slosh_expression = "Surge_Area = 'Category 1 and 2 Hurricanes' or Surge_Area = 'Area
May Become Isolated'" #SQL statement used to designate Category 1 hurricane
158 elif SloshVal == 3: #True if Category 3 hurricane selected by user
159     gp.AddMessage("Category 3 Hurricane Selected") #Outputs message when running as a
script in ArcToolbox
160     slosh_expression = "Surge_Area = 'Category 1 and 2 Hurricanes' or Surge_Area = 'Area
May Become Isolated' or Surge_Area = 'Category 3 Hurricanes'" #SQL statement used to
designate Category 1 hurricane
161 elif SloshVal == 4: #True if Category 4 hurricane selected by user
162     gp.AddMessage("Category 4 Hurricane Selected") #Outputs message when running as a
script in ArcToolbox
163     slosh_expression = "Surge_Area = 'Category 1 and 2 Hurricanes' or Surge_Area = 'Area
May Become Isolated' or Surge_Area = 'Category 3 Hurricanes' or Surge_Area = 'Category 4
Hurricanes'" #SQL statement used to designate Category 1 hurricane
164 else:
165     print "Your selection must be 1, 2, 3, or 4" #Prints error message if proper hurricane
category not selected
166
167 gp.MakeFeatureLayer_management (slosh, "slosh_lyr") #Converts SLOSH feature class to a
feature layer
168 gp.SelectLayerByAttribute_management ("slosh_lyr", "NEW_SELECTION", slosh_expression)
#Selects user-defined SLOSH values
169
170 ##Selected SLOSH Data used to define 'dry intersections'
171 gp.SelectLayerByLocation_management ("intersections_layer", "COMPLETELY WITHIN", "slosh_lyr"
, "", "NEW_SELECTION") #Selects all nodes within selected SLOSH flood area
172 gp.SelectLayerByLocation_management ("intersections_layer", "", "", "", "SWITCH_SELECTION")
#Swaps selection made in previous line; now all non-flooded nodes are selected

```

```
173 gp.CopyFeatures_management ("intersections_layer", "dryintersections.shp") #The layer with  
the 'dry intersections' selected is converted to a shapefile
```

Appendix B: Hurricane Evacuation Zone Delineation Tool Python Script

```
1  import arcpy,os,win32com.client,sys,string,glob,csv,math,time,win32file
2
3  ##SETUP GEOPROCESSOR##
4
5  gp = arcpy scripting.create()
6
7  gp.SetProduct('ArcView')
8
9  gp.OverwriteOutput = 1
10
11 gp.CheckOutExtension('spatial')
12
13 gp.AddToolbox("C:\Program Files\ArcGIS\ArcToolbox\Toolboxes\Data Management Tools.tbx")
14 gp.AddToolbox("C:\Program Files\ArcGIS\ArcToolbox\Toolboxes\Analysis Tools.tbx")
15 gp.AddToolbox("C:\Program Files\ArcGIS\ArcToolbox\Toolboxes\Spatial Analyst Tools.tbx")
16
17 datadir=sys.argv[1]#User-Defined variable 1
18 tab_output="C:\\Users\\Dean\\Desktop\\python\\tables\\"
19 gp.workspace=datadir
20 polyfile=sys.argv[2]#User-Defined variable 2
21 rastfile=sys.argv[3]#User-Defined variable 3
22 polyfile_temp=datadir+"\\polyfile_temp.shp"
23 out_dir=sys.argv[4]#User-Defined variable 4
24 outputfile=open("C:\\Users\\Dean\\Desktop\\python\\output\\outfile.txt","w")
25 ccriteria=1
26 iteration=1
27 mergedpoly=[]
28 timetotal=0
29 merge_code=9999
30 ##END GEOPROCESSOR SETUP##
31 ##BEGIN ITERATIONS##
32
33 while ccriteria < 251:
34     print "Iteration "+str(iteration)
35     t1=time.clock()
36     ##BEGIN GEOPROCESSING##
37
38     minilist=[]
39     counter=0
40
41     #Perform Zonal Stats on Area
42
43     gp.ZonalStatisticsAsTable_sa(polyfile, "Id", rastfile, tab_output+"outTable.dbf", "DATA"
44 )##changed from _fid_
45     print "Zonal Stats Complete."
46
47     #Create list of polygons
48
49     cursor=gp.searchcursor(polyfile)
50     row=cursor.next()
51     fcnumlist=[]#blank feature class number list
52     while row <> None:
53         entry=row.getvalue('Id')#was: entry=row.getvalue('id'), ('Id')
54         fcnumlist.append(entry)
55         row=cursor.next()
56     fcnum=len(fcnumlist)
57     del row,cursor
58
59     gp.MakeFeatureLayer(polyfile, "polyfile_lyr")
60     adjdict={}
61     cursor=gp.searchcursor("polyfile_lyr")
62     row=cursor.next()
63     feat_num=len(fcnumlist)
64     for num in fcnumlist :
65         value=row.getvalue('Id')
```

```

66         #Select Polygon and Create Temp Polygon File
67
68
69         expression = "Id = "+str(value)##-----changed from fid
70         gp.SelectLayerByAttribute_management ("polyfile_lyr", "NEW_SELECTION", expression)
71         gp.SelectLayerByLocation_management ("polyfile_lyr", "INTERSECT" , "", "",
"NEW_SELECTION")
72         gp.copyfeatures_management ("polyfile_lyr","polyfile_temp")
73         gp.MakeFeatureLayer(polyfile_temp, "polyfile_temp_lyr")
74
75         #Extract Polygon ID into Dictionary
76         cursor2=gp.searchcursor("polyfile_temp_lyr")
77         cursor2.reset()
78         row2=cursor2.next()
79         while row2:
80             minilst.append(row2.getvalue('Id'))
81             row2=cursor2.next()
82             counter=counter+1
83         adjdict[num]= minilst
84
85         del minilst, row2, cursor2
86         minilst=[]
87         gp.delete("polyfile_temp_lyr")
88
89
90         cursor.next()#Advances to next polygon
91
92         del cursor
93
94         #remove dissolved polygons from adjacency list
95         adjdict_keys=adjdict.keys()
96         for x in mergedpoly:
97             for y in adjdict_keys:
98                 if x == y:
99                     del adjdict[x]
100
101
102         print "Adjacency Dictionary Complete."
103
104         #Convert DBF to CSV Using MS Excel and Read CSV into List
105         dbf_file="C:\\Users\\Dean\\Desktop\\python\\tables\\outTable.dbf"
106         excel=win32com.client.Dispatch('Excel.Application')
107
108         outCSV=dbf_file[:-4]+".csv"
109         workbook = excel.Workbooks.Open(dbf_file)
110         workbook.SaveAs(outCSV,FileFormat=24)
111         workbook.Close(False)
112         excel.Quit()
113
114
115         CSV_file= open('C:\\Users\\Dean\\Desktop\\python\\tables\\outTable.csv', 'r')
116         linelist=CSV_file.readlines()
117         CSV_file.close()
118         del linelist[0]
119         sm_list=[]
120         sm_dict={}
121         cnt=0
122         for number in range(fcnum):
123             line=linelist[cnt].split(",")
124             del line[1]
125             del line[1]
126             del line[1]
127             del line[1]
128             del line[1]
129             del line[3]
130             del line[3]

```

```

131         del line[3]
132         del line[3]
133         del line[3]
134         sm_list.extend(line)
135         key=line[0]
136         sm_dict[key]=line
137         cnt=cnt+1
138
139
140     zstat_dict={}
141     keykey=0
142     elkey=1
143     e2key=2
144     for x in range(fcnum):
145         listkey=int(sm_list[keykey])
146         e1=float(sm_list[elkey])
147         e2=float(sm_list[e2key])
148         zstat_dict[listkey]=[e1,e2]
149         keykey=keykey+3
150         elkey=elkey+3
151         e2key=e2key+3
152
153     ##END GEOPROCESSING##
154     print "Geoprocessing Complete"
155     ##BEGIN STATISTICS##
156
157     zstatdictsd={}
158     zstatdictkeys=zstat_dict.keys()
159     for x in zstatdictkeys:
160         zstatdictsd[x]=zstat_dict[x][1] #creates dictionary with polyID and SD values only
161     zstatdictsd_swap={} #This section swaps SD and polyID values so SD is the key
162     for x in zstatdictkeys:
163         key=x
164         value=zstatdictsd[x]
165         zstatdictsd_swap[value]=key
166     keylist=adjdict.keys()
167     sublist=[]
168     sddict={}
169     for key in keylist: #Swap adjacent polygon and key polygon value for SD values
170         keysd=zstat_dict[key][1]
171         adjlist=adjdict[key]
172         for entry in adjlist:
173             sd=zstat_dict[entry][1]
174             sublist.append(sd)
175         sddict[keysd]=sublist
176         sublist=[]
177     del key, sublist
178
179     sdlistkeys=sddict.keys() #Remove central polygon from calculation and swaps polyID with SD
180     calcdict={}
181     calclist=[]
182     for key in sdlistkeys:
183         adjsdlist=sddict[key]
184         for sd in adjsdlist:
185             if key != sd:
186                 calclist.append(sd)
187         calcdict[key]=calclist
188         calclist=[]
189     del key, sd, calclist
190
191     compdict={} #Creates Joint SD Dictionary
192     complist=[]
193     listallsd=[]
194     calckeys=calcdict.keys()
195     for key in calckeys:
196         calcentry=calcdict[key]

```



```

197         for sdentry in calcentry:
198             jointsd=math.sqrt((key**2)+(sdentry**2))/2#Computes joint SD values for each
polygon
199             complist.append(jointsd)
200             listallsd.append(jointsd) #creates list with all SD values
201             compdict[key]=complist
202             complist=[]
203         del key, complist
204
205         #Calculate Percentile
206         perlist=[]
207         listallsd.sort() #SD values list is sorted to find percentiles
208         length=len(listallsd)
209         percentile=10.0
#10.0-----
210         rank=(int(round((percentile/100)*(length+1))))-1
211         value=listallsd[rank]
212         for x in listallsd:#Create List of all joint SD values under 10th Percentile
213             if x <= value:
214                 perlist.append(x)#Creates list of all JSD below percentile
215         del x
216
217         #Build dictionary with both JSD and PolyID
218         perlist.sort()
219         listlist=[]
220         compdict2=compdict#Allows editing of a copy of original compdict dictionary
221         compdict2keys=compdict2.keys()
222         polyjsddict={}
223         counter=0
224         for x in compdict2keys:
225             entrylist=compdict2[x]
226             for y in entrylist:
227                 poly_id=calcdict[x][counter]
228                 listlist.extend([poly_id,y])
229                 counter=counter+1
230             polyjsddict[x]=listlist
231             counter=0
232             listlist=[]
233
234         prelimlist=[]
235         keylist=polyjsddict.keys()
236         for key in keylist:#This loop removes dictionary entries that are greater than the 10th
percentile
237             length=len(polyjsddict[key])-1
238             #print "key= "+str(key)
239             #print "length= "+str(length)
240             while length >= 0:
241                 prelimlist.extend(polyjsddict[key][length])
242                 if prelimlist[1]>value:
243                     del polyjsddict[key][length]
244                 prelimlist=[]
245                 length=length-1
246         del prelimlist,key,length
247
248
249         keylist=polyjsddict.keys()#This loop removes dictionary entries that are greater than
"value" variable
250         for key in keylist:
251             if len(polyjsddict[key]) == 0:
252                 del polyjsddict[key]
253
254         #Define pair means, calculate difference, and determine smallest difference value
255
256         #Make list of JSD values
257

```

```

258     del x,y
259     listofjsdentries=[]
260
261     polyjsddictkeys=polyjsddict.keys()
262     for x in polyjsddictkeys:
263         length=len(polyjsddict[x])
264         while length>0:
265             polyid=zstatdictsd_swap[x]
266             mean=zstat_dict[polyid][0]#gets mean from zstat_dict
267             jsddictentry=5
268             listofjsdentries.append(jsddictentry)
269             length=length-1
270     finalentryset=set(listofjsdentries)
271     finalentrylist=list(finalentryset)#makes list of all JSD values from a set for each
    polygon remaining from previous steps that remove polygons with JSDs greater than 10th
    percentile
272     finalentrylist.sort()
273     lowestjsd=finalentrylist[0]#derives lowest JSD
274
275     #Identify left over polyID pairs
276     rem_polyid=len(polyjsddict.keys())
277     keys=polyjsddict.keys()
278     pair_dict={}
279     temp_list=[]
280     counter=1
281     for x in range(rem_polyid):#create dictionary with count as a key and a list of the two
    matching, remaining polyID pairs as values
282         entry1=zstatdictsd_swap[keys[x]]
283         len2=len(polyjsddict[keys[x]])
284         for y in range(len2):
285             entry2=zstatdictsd_swap[polyjsddict[keys[x]][y][0]]
286             temp_list.append(entry1)
287             temp_list.append(entry2)
288             pair_dict[counter]=temp_list
289             temp_list=[]
290             counter=counter+1
291         temp_list=[]
292
293     del keys,x,y,temp_list,counter,rem_polyid,entry1,entry2
294
295     ***pair_dict** should now have a counter and two polyID entries for each pair that
    needs to have the difference of means calculated
296
297     #next step will switch candidate polygon pair IDs with means to create
    **polymeanpairs** dictionary
298
299     polymeanpairs={}
300     keys=pair_dict.keys()
301     diff_list=[]
302     pair_dict_diff={}
303     for x in keys:
304         temp_list=[]
305         #temp_list2=[pair_dict[x]]
306         var1=pair_dict[x][0]
307         entry1=zstat_dict[var1][0]
308         var2=pair_dict[x][1]
309         entry2=zstat_dict[var2][0]
310         diff=abs(entry1-entry2)#calculates difference of means between the two polygon
    pairs' means.
311         diff_list.append(diff)#creates list of only differences
312         temp_list.extend([entry1,entry2,diff])
313         #temp_list2=pair_dict[x].extend(diff)
314         pair_dict[x].append(diff)
315         polymeanpairs[x]=temp_list
316
317     del keys,temp_list,var1,entry1,var2,entry2,diff,x

```

```

318
319     diff_list.sort()#sorts diff_list to find smallest difference
320     low_mean=diff_list[0]#identifies which mean difference is lowest
321     print "Lowest difference of means value = "+str(low_mean)+". "
322
323     #finds which polygon IDs from **polymeanpairs** that has lowest difference of means;
creates **merge_poly_list** which contains the two Polygon IDs to be merged.
324
325     merge_poly_list=[]
326     keys=polymeanpairs.keys()
327
328     while len(merge_poly_list) == 0:
329         for key in keys:
330             mean=polymeanpairs[key][2]
331
332             if mean == low_mean:
333                 merge_poly1=pair_dict[key][0]
334                 merge_poly2=pair_dict[key][1]
335                 temp_list=[]
336                 temp_list.extend([merge_poly1,merge_poly2])
337                 merge_poly_list=temp_list
338
339     print "Polygon ID "+str(merge_poly1)+" and Polygon ID "+str(merge_poly2)+" must be
merged."
340
341     ##END STATISTICS##
342     print "Stats Complete"
343     ##BEGIN DISSOLVE##
344
345     polynome="zip_poly"
346
347     #The next two while loops replaces the Id field in the attribute table from the default
value to the merge_code
348
349     cur=gp.updatecursor(polyfile)
350
351     row=cur.next()
352
353     event=0
354     while event==0:#Iterates through attribute table to replace merge_poly1 with merge_code
355         if row.getvalue('Id') == merge_poly1:
356             row.setvalue("Id",str(merge_code))#swaps default dissolve code with merge code
357             cur.updaterow(row)#saves previous row.setvalue commands
358             event=1
359             row=cur.reset()
360             row=cur.next()
361
362         else:
363             row=cur.next()
364
365     del row,cur,event
366
367     cur=gp.updatecursor(polyfile)
368
369     row=cur.next()
370
371     event=0
372     while event==0:#Iterates through attribute table to replace merge_poly2 with merge_code
373         if row.getvalue('Id') == merge_poly2:
374             row.setvalue("Id",str(merge_code))#swaps default dissolve code with merge code
375             cur.updaterow(row)#saves previous row.setvalue commands
376             event=1
377         else:
378             row=cur.next()
379
380     del row,cur

```

```

381         merge_code=merge_code-1
382
383         gp.MakeFeatureLayer(polyfile, "polyfile_layer")#converts data into a layer so it can be
384         dissolved
385
386
387         gp.Dissolve_management("polyfile_layer", "zip_poly2", "Id")#dissolves layer using Id
388
389         del_filelist=glob.glob(datadir+"\\\\"+polynome+".*")#removes all original "zip_poly.shp"
390         file and associated files
391         for x in del_filelist:
392             os.remove(x)
393         del del_filelist
394
395         rename_filelist=glob.glob(datadir+"\\\\"+polynome+"2"+".*")#will rename all files
396         associated with "zip_poly2.shp" file back to original "zip_poly.shp"
397         for x in rename_filelist:
398             newfn=x.replace("zip_poly2","zip_poly")
399             os.rename(x,newfn)
400
401         gp.delete("polyfile_layer")
402
403         ##END DISSOLVE##
404
405         print "Dissolve Complete"
406
407         os.remove("C:\\Users\\Dean\\Desktop\\python\\tables\\outTable.csv")
408         os.remove("C:\\Users\\Dean\\Desktop\\python\\tables\\outTable.dbf")
409         os.remove("C:\\Users\\Dean\\Desktop\\python\\tables\\outTable.dbf.xml")
410
411         del_filelist=glob.glob(datadir+"\\polyfile_temp.*")#removes all original
412         "polyfile_temp.shp" file and associated files
413         for x in del_filelist:
414             os.remove(x)
415         del del_filelist
416
417         #Create list of polygon IDs that are no longer available
418         mergedpoly.append(merge_poly1)
419         mergedpoly.append(merge_poly2)
420
421         t2=time.clock()
422         timespent=t2-t1
423
424         print "Iteration "+str(iteration)+" took "+str(timespent)+" seconds (" +str(float(
425         timespent)/60)+ " minutes)."
426
427         ##CLOSING CRITERIA CALCULATIONS##
428         sd1=zstat_dict[merge_poly1][1]
429         sd2=zstat_dict[merge_poly2][1]
430         avgstdmergepoly=(sd1+sd2)/2
431         print "Average standard deviation between the two merged polygons = "+str(
432         avgstdmergepoly)+". "
433
434         ##Move output polygon file to its own folder
435         dir=datadir+"\\output\\iteration_"+str(iteration)
436         os.mkdir(dir)
437         mov_fl=glob.glob(datadir+"\\zip_poly.*")
438         for x in mov_fl:
439             fname=x.lstrip("C:\\Users\\Dean\\Desktop\\python\\")
440             dir_file1=str(x)
441             dir_file2=str(dir+"\\\\"+str(fname))
442             win32file.CopyFile(dir_file1, dir_file2,0)#Arguments: from file; to file;
443             0=overwrite if file exists, 1=stop script if file exists)
444         del mov_fl,x,dir_file1,dir_file2

```

```

440     ##Save average STDDDev in output file file
441     data=str(iteration)+","+str(merge_poly1)+","+str(merge_poly2)+","+str(avgstdmergepoly)+
"\n"
442     outputfile.write(data)
443     del data
444
445     ccriteria=ccriteria+1
446     iteration=iteration+1
447     timetotal=timetotal+timespent
448     print "Total time so far = "+str(float(timetotal/60))+ " minutes, "+str(float(timetotal/
60/60))+ " hours."
449     print ""
450
451 else:
452
453     outputfile.close()
454
455     del_filelist=glob.glob(datadir+"\\polyfile_temp.*")#removes all original
"polyfile_temp.shp" file and associated files
456     for x in del_filelist:
457         os.remove(x)
458     del del_filelist
459
460     gp.CheckInExtension('spatial')
461     print "Process Complete."
462     print ""
463     print "Total Iterations = "+str(iteration-1)+ "."
464     print "Total Time = "+str(timetotal)+ " seconds (" +str(float(timetotal/60))+ " minutes)."

```

Appendix C: The Road Intersection Flood Analysis Tool Technical Description

This script began like most other geoprocessing Python scripts by defining the geoprocessor and setting the working environment. Line 1 to line 23 accomplished this task. The input variables were defined in line 11 to line 16; they corresponded to the user inputs illustrated in Figure 3. Other settings such as data input and output locations, ArcGIS toolbox locations, and defining the output coordinate system were performed. This script required importing the *arcgisscripting* and *sys* modules. In addition, some commands used in this script required the *Data Management Tools* toolbox; it was initialized in line 23.

The first geoprocessing task performed was to select only the locations where the analysis would be performed (lines 26 and 27). In the test application, the entire Connecticut town boundary shapefile was converted into a layer where a SELECT BY ATTRIBUTE function was used which will select only the towns defined by the user. The SELECT BY ATTRIBUTE function required layer data; many other tools also required layer data and resulted in shapefile-to-layer conversions. As a result, the tool had many shapefile-to-layer conversions which facilitated the SELECT BY ATTRIBUTE function as well as others, including DISSOLVE and CLIP.

Once the towns in the test application were selected, the road shapefile was clipped to the selected towns in the town boundary layer (line 31). Like the town boundary shapefile, the clipped road shapefile was converted to a layer (line 32).

Line 40 takes the clipped road network layer and dissolved the lines into single part

features. Road segments were broken apart in each location where another line crossed it. The dissolve command performed this action whether or not there is a node at each line intersection. This was the reason the road network data does not need to be topologically accurate. The result of the dissolve function was a line shapefile.

The line shapefile created from the previous step was processed in order to determine all road intersections. A blank Python dictionary was defined (line 44) and a geoprocessing search cursor was established (lines 47-48). The code then iterated through the endpoints of each line segment, populating the blank Python dictionary with their X,Y coordinates. A simple counter (line 51) was used to key the dictionary.

After the endpoint coordinate dictionary was complete, the dictionary keys were used (line 76) to iterate through each X,Y pair of endpoint values. Nested *for* loops were used to compare sets of X,Y pairs. A road intersection was identified if two X,Y pairs are identical. Each time a road intersection was defined, its X,Y location was saved to a list (lines 97–100). Line 82 to line 100 evaluated the line shapefile and determined where road intersection nodes should be located.

Since each X,Y pair must go through the iteration process, there were multiple instances of the same X,Y pair in the node list. The node list was processed in lines 104–116 in order to remove duplicate entries. In an iterative fashion using *if* statement and *for* loops, it read the node list and removed all but one instance of each X,Y pair. The final product of the previous two steps resulted in a list of X,Y pairs that represented valid road intersections.

The node list of X,Y pairs produced in the previous step were converted to a point shapefile in order to determine which road intersections will be flooded. Line 120 created a blank point shapefile where the X,Y node data will be converted into point geometry. The *for* loop in lines 126-144 read each X,Y pair and created a point feature using the geoprocessing insert cursor. The loop iterated through the entire node list until all X,Y pairs were added to the new point shapefile. Once the point shapefile was complete, it is converted to a layer (line 148).

The last portion of this script established the proper flood polygon and created the final point shapefile representing only the road intersections that can be traversed without worry of rising floodwaters. Line 151 took the user input SLOSH hurricane intensity and converted it from a string to a number. This number was then used in lines 152-165 to define a SQL (Structured Query Language) statement and was used later to select the proper flood polygons. For example, if the user selects a hurricane with the Saffir-Simpson intensity of a Category 3, the SQL expression representing hurricanes of Categories 1, 2, and 3 is invoked since intersections located in all three category zones would be affected. This SQL expression was used to select the proper polygons in the SLOSH data (line 168).

The SLOSH layer containing the selected flood polygons was used to select out the nodes created earlier in the script. A select layer by location command was performed (line 171) to select the points that were underwater during a flood event. Another select layer by location was executed (line 172) where the points selected in the previous step were inverted so that the non-flooded nodes were now selected. The final line of the script (line

173) contained the copy features command that took the point layer with its non-flooded intersections selected and created a shapefile containing only those selected points.

Appendix D: The Hurricane Evacuation Zone Tool Technical Description

As in the previous script, this script began by initializing the geoprocessing functions used by Python and ArcGIS. This script required the use of the following modules: *arcgisscripting*, *os*, *win32com.client*, *sys*, *string*, *glob*, *csv*, *math*, *time*, and *win32file*. The three toolboxes required to run this script include the *Data Management Tools*, *Analysis Tools*, and *Spatial Analyst Tools*. As Figure 7 indicates, there were four user defined variables used in this script. They were the directory where the data are stored, the full path name of the polygon file, the full path name of the DEM, and the output directory. These initialization steps were contained in lines 1 through 24.

The Wilmot and Meduri (2005) study discussed multiple approaches to determine closing criteria for this iterative process. It was decided in this study that the average standard deviation of the merged zones could indicate an appropriate closing criterion. Per a personal correspondence with Chester Wilmot (2008), the author suggested that any statistic could be used in a study as a closing criterion as it depends on an acceptable limit chosen by the researcher. If the closing criteria are changed, the script would have to be changed and executed again until an acceptable result is reached.

The design of the script presented in this research set an arbitrarily high number of iterations. The noteworthy statistics for each iteration were collected and stored in a text file as the script processes the data. The benefit of such a method allowed the script to be executed just one time instead of multiple times while the researcher attempts to finalize the exact closing criteria. Once this script is executed, the researcher can review the data contained in the output text file that contains the statistics for each iteration and the

closing criteria can be decided.

Before the iterative process begins in line 33, other variables were set that aid in the proper execution of this script. The variables *ccriteria* and *iteration* are later used to tally the number of iterations that are performed and were used to terminate the script. Line 27 created an empty list that must be created before the iterative process. The variable *timetotal* was used to calculate the length of time each iteration requires. Finally, *merge_code* was used later in the script to properly number the polygons selected to merge.

The iterative portion of the script began on line 33 by use of a *while* loop. With each successful iteration, the *ccriteria* variable increased by a value of 1. Since this variable's initial value is 1, the iterations continued until *ccriteria* is equal to 251, for a total of 250 iterations. The researcher could change this threshold; if too many or few iterations are set before the script is executed, the output may contain too many or too few hurricane evacuation zones. Setting the number of iterations to 250 out of 300 possible polygons ensured enough merges will be performed, but did not unnecessarily create very large zones resulting from too many merges.

The first step of processing the data in this study required the calculation of zonal statistics for each polygon. Line 43 performed this task by using the polygon file containing the candidate hurricane evacuation zones and the DEM previously referenced. The output of this procedure was stored in a temporary .DBF file and is used later in the script.

The polygon file containing the candidate hurricane evacuation zones was processed using the search cursor function in lines 48 to 56. This procedure created a Python list of all of the polygons in the file.

Lines 58 through 92 used the list created in the previous step to build a Python dictionary containing entries for each polygon. This dictionary was used in the script to provide adjacency information for each polygon. For any given polygon, this dictionary provided a list of polygons that were adjacent to the initial selected polygon. This adjacency dictionary was created by a series of selections based on attributes and locations. The initial polygon was selected in line 70 using a Select by Attribute function. A Select by Location function was performed (Line 71) using the Intersect method. This selects all adjacent polygons. The selected polygons were output into a temporary shapefile using the Copy Features command (line 72). Line 73 converted this temporary file into a layer, a step required in order to create a search cursor. This search cursor (Line 76) probed the temporary shapefile and extracted the polygon identification information. Once extracted, this information was stored in the variable *adjdict*, or the adjacency dictionary. This entire procedure was repeated for every polygon in the zip code-major roads shapefile; each new entry was appended into the adjacency dictionary.

The adjacency dictionary inherently contained the polygon identifier for each polygon. For example, if you wanted to determine which polygons were adjacent to Polygon 35, one of the entries in the adjacency dictionary for this polygon would also be Polygon 35. This problem was remedied in lines 95 to 99. These lines scanned the adjacency dictionary and removed these redundant entries. Line 102 provided feedback to the user

of the script that the adjacency dictionary for this iteration had been created.

The next portion of the script prepared the output of the zonal statistics to be processed.

The .DBF file created by the Zonal Statistics Tool was converted into a comma separated value file using the *win32com.client* module imported at the beginning of the script. Line 106 to 112 used this capability and invoked the file format conversion tools of Microsoft Excel. Line 110 converted the .DBF file into a .CSV file by using the *FileFormat* command with a value of 24. Once the comma separated value file was created, lines 115 to 151 imported the comma separated file into another Python dictionary (*zstat_dict*). This dictionary was keyed with each polygon's identification number and the entries contained the zonal statistics for those polygons.

The next portions of the script processed the statistics associated with the polygon file.

The statistics segment of the script decided which two adjacent polygons should be combined into one.

Before any calculations are performed, the data in the *zstat_dict* dictionary were reformatted for easy assimilation into the script. Since the main idea of defining the hurricane evacuation zones was primarily based on each polygon's standard deviation, a new dictionary (*zstat_dicts*) was created which had the polygon ID as the keys and only the polygon's standard deviation as its entry (lines 157 to 160). The other statistics created by the Zonal Statistics Tool were removed. An additional step of swapping the keys with the entries was performed in lines 161 to 166; this step was necessary later in the script when joint standard deviations were calculated. At this point, the two dictionaries were *zstat_dict* and *zstat_dicts*, both of which contained the same data,

but the entries and keys were reversed.

The next step in this process was to incorporate the polygon adjacency data in the *adjdict* dictionary with the standard deviation data in the *zstat_dict* dictionary. This procedure was executed in lines 169 to 177 where the adjacency polygon identification data in *adjdict* was switched with the standard deviations of each adjacent polygon. This new data was stored in the *sddict* dictionary. Before this dictionary can be processed any further, redundant entries in the dictionary were removed in lines 179 to 189 where an iterative process created a new dictionary named *calcdict* which only contained the standard deviations for each unique adjacent polygon with no repeats.

calcdict was used to create the joint standard deviation dictionary. The joint standard deviation was defined by Wilmot and Meduri (2005) as the square root of the sum of each adjacent polygon's squared standard deviation divided by two. A new Python dictionary (*compdict*) was created which stores this data. Lines 191 to 203 showed this portion of the script; the joint standard deviation was calculated in line 198. A list was created and appended in line 200 with each new joint standard deviation value. This list, *listallsd*, was used later during the calculation of the percentile.

A percentile calculation was used (lines 206 to 215) to help determine which two hurricane evacuation zones to combine. This procedure used the *listallsd* list created in line 200. This list was sorted from smallest value to largest in line 207. As prescribed by Wilmot and Meduri (2005), a percentile value of 10 was used (line 209) to determine which joint standard deviation value is nearest the tenth percentile. The calculation of the joint standard deviation value that occupies the tenth percentile of the *listallsd* list was

performed in line 210. After this value was calculated, another list was created (*perlist*) which contained only the joint standard deviation values below the tenth percentile.

At this point, it was important to begin merging the joint standard deviation values that were below the tenth percentile with their corresponding polygon identification values.

The *compdict2* dictionary was created directly from the *compdict* dictionary. This copy of the original allowed free editing and manipulation of the data without risking data loss in *compdict*. A dictionary used to contain both the joint standard deviation values along with their corresponding polygon identification values was created (*polyjsddict*). Since the *polyjsddict* dictionary contained polygons that had joint standard deviation values greater than the tenth percentile, it was necessary to remove them. This was performed using a loop in lines 236 to 246. A similar procedure was performed in lines 249 to 252 that removed all dictionary entries in *polyjsddict* where the key had a character length of zero. This problem arose from previously removed polygons that were greater than the tenth percentile. At this point, the *polyjsddict* dictionary contained only keys and entries that represented the adjacent polygon information paired with the joint standard deviation values.

Now that the polygons that have the joint standard deviation values less than the tenth percentile were identified and stored in *polyjsddict*, the mean for each of these polygons were extracted from *zstat_dict* (lines 299 to 315). These lines also calculated the difference of mean elevations between each candidate pair of adjacent polygons. A new list named *diff_list* contained a list of all possible combinations of adjacent polygons along with the corresponding difference of means. This list was sorted from lowest value

to highest (line 319) and the lowest difference of means of the candidate pairs was ascertained in line 320. The lowest difference of means value was stored in the *low_mean* variable.

The last portion of the statistics calculations in this script used the variable *low_mean* to identify which polygon pair should be merged. Lines 325 to 337 processed the entire list of candidate polygon pairs stored in the *polymeanpairs* dictionary until the value stored in the *low_mean* variable was matched. When a matching dictionary entry was found, the two polygon identification values were extracted (lines 333 and 334) and stored in corresponding variables. At this point, the two adjacent polygons that must be merged were identified and the merge can take place.

Now that the two candidate polygons to be merged were defined, the last part of the script can merge these polygons together. This process began on line 349 which created an update cursor, an object in Python used to update data in a shapefile's attribute table. Since the feature identification values for each of the polygons would change as pairs are merged, each iteration would have a different set of values, complicating the process. A field called 'Id' was used to store merge codes. These codes were designed to be very large when compared to the original Id values and prevented two polygons from having the same Id value. The values for this field were initially set to the FID value before this script was executed. The two adjacent polygons that have been identified got new merge codes which indicated to the script which two to merge. Lines 354 to 365 showed the loop process that searches the attribute table for the first polygon to be merged and replaced the Id value with the merge code. The same process was repeated for the second

selected polygon in lines 367 to 380. In order to have a unique merge code for each iteration, the merge code value was decreased by a value of one in line 381.

The final geoprocessing steps were performed in lines 384 to 397. First, the shapefile was converted to a layer in line 384; this is a required step that allows the dissolve tool to function. The next line of the script contained the command that dissolves (merges) the two adjacent polygons. The dissolve was executed based on the Id field prepared earlier.

Once the dissolve was complete, a list was created of all temporary files (line 387) using the glob module and its associated glob command. The glob command created a list of all files in a specified location using specific search criteria. This list was then used to delete all temporary files in lines 388 to 390. The files that were to be kept were renamed using a similar glob command. Both of the glob commands were used in conjunction with the os module. In the first case, the os.remove command was used (line 389) to delete the list of temporary files. In the latter case, os.rename was used (line 395) to rename each of the files in the glob list. This represented the last of the geoprocessing tasks.

The end of the script contained some file management commands that deleted all temporary files and variables for the iteration that had been created. First, the tables created by the zonal statistics tool and its subsequent files were deleted to prepare for the next iteration; this process used the os.remove command (lines 403 to 405) to delete them. Lines 407 through 410 rid the working directory of more temporary shapefiles created earlier in the script.

In order to keep a log of which polygons are merged, a list named *mergedpoly* was used

to store the polygon Id values (lines 413 and 414).

While not necessary for processing data, some lines of script were created which monitors how much time passes for each iteration. Line 416 used the time module's clock method to calculate the time when the script got to this point. It was stored in the variable named *t2*. The variable *t1* was created earlier in line 35. The difference in time from when these two variables were given values by the script is determined in line 417. Feedback was given to the user in line 419 which indicates how long the previous iteration took to complete.

An example of a closing criteria statistic was included in this script in lines 422-424. As stated in Wilmot and Meduri (2005), the closing criterion selected for their particular study area was the average standard deviation between the two merged polygons. This statistic was the one that is calculated and printed in the PythonWin interactive window. Any statistic deemed important in calculating closing criteria could be written into the script at this point. In this particular case, the selected closing criterion was written into a text file using a comma separated format (lines 428 and 429). Each line of this file contained the iteration number, the polygon Id value for the two merged polygons, and the closing criterion statistic.

Using glob and os module commands, lines 433 to 441 made a uniquely-named directory where the final shapefile for the iteration was stored. Once the new directory was created, the shapefile data were copied into the new directory. With each iteration, the directory name was changed incrementally to indicate the iteration number.

Some functions of the script were based on variables that increase by a value of one for each iteration. For example, the *iteration* variable stored the iteration number. At the end of one iteration, the value stored by this variable was increased by one. Lines 443 and 444 contained the two incremental variables used in this script. Line 445 created a *timetotal* variable which tallied the time it took for the entire script to run. Line 446 provided user feedback related to the time information.

At this point, the iteration was complete. Since this script was instructed to run for 250 iterations, it went back to line 33 to begin processing another iteration if the *iteration* variable was less than 251. After 250 iterations, the *iteration* variable equaled 251 and the while loop in line 33 which began the iteration process ceased to continue and skipped to the last few lines of the script.

There were four functions at the end of the script that were run in order to finalize the output file and erase any leftover files from the hard disk. The output file which was opened for editing in line 24 was closed otherwise data corruption of the file may occur. This file was closed in line 451. A glob list was created in line 453 and was used to delete the last of the temporary files created by the script. In line 458, the spatial analyst extension must be released from use by the script. If this is not done, other scripts or even the ArcGIS software will not be able to use the extension since it is locked for use. The last four lines of the script (459 to 462) provided user feedback on the completion of the script, how many iterations were performed, and the total time taken for the script to run.