5-2017

Data and Geometry; Model Building at Calthorpe Analytics

Samuel M. Upton
Clark University, supton@clarku.edu

Data and Geometry

Model Building at Calthorpe Analytics

Samuel Upton

Degree will be conferred May 2017

A GISDE final project paper
submitted to the faculty of Clark University, Worcester, Massachusetts,

in partial fulfillment of the requirements for the degree of

Masters of Science in Geographic Information Sciences for Development and Environment

in the Department of International Development, Community, and Environment

Accepted on the recommendation of

Yelena Ogneva-Himmelberger, Project Advisor

Abstract

Data and Geometry; Model Building at Calthorpe Analytics

Samuel Upton

This report documents my Summer 2016 internship with Calthorpe Analytics, a Berkeley CA-based urban planning firm. Calthorpe Analytics specializes in scenario development for planning, modeling, and plan evaluation for government and municipal clients. The primary responsibility of my internship was model development and refinement using advanced spatial analytics working in the Python programming language. The internship was extremely successful: it gave me a great opportunity to strengthen my open source GIS skillset, deepen my understanding of data science, and vastly improve my geospatial programming skillset. It also gave me a chance to apply advanced geospatial modeling and spatial statistics in practice. The firm provided a great working environment and a very supportive culture in which to learn and test new ideas and techniques. The following paper will expand on the work of Calthorpe Analytics, their culture and organization, my contributions to their workflow, and reflect on the personal and professional impact of the internship.

Yelena Ogneva-Himmelberger, Project Advisor

Academic History

Name: Samuel Upton

Baccalaureate Degree: B.A., History and Religion

Source: Oberlin College, Oberlin OH

Date: May, 2000

Dedication

I would like to dedicate this paper to my wife, without whom none of this would have been possible.

Acknowledgement

A number of people had a hand in the success of my internship. I would like to thank Nick Wilson for connecting me with Calthorpe Analytics. I would also like to thank Nick and his wife Robin for taking me in, and for showing me the Bay area. I would like to thank Joe DiStefano for giving me a chance to join his team, and the freedom to find my own path through his systems. I would like to thank Erika Lew, Kelan Stoy, Koshy Thomas, and Brock Hicks for their help, teaching and enthusiasm. I would like to thank Jamie Alessio, Alec Flec, and the whole Development team for their advice and technical help. I would finally like to thank my professors and peers in the GISDE program, whose ongoing teaching, support, and encouragement has given me the skillset and drive to succeed in the face of any and all challenges.

Table of Contents

	Page
Introduction	1
Organization	2
Mission	3
Work	4
Spatial Analysis and Mapping	4
Organizational Structure and Culture	5
Strengths	5
Internship Responsibilities	7
Assessing my Internship	12
Conclusion	14
Figures	15
Sources	16
Appendix	17

Introduction

My internship this past summer took me to North Berkeley, and a firm called Calthorpe Analytics. This firm performs scenario-based growth projection and outcome modeling for municipalities. I joined their Analytics team, a group of young engineers, planners, and data scientists who perform client analysis and develop new methodologies to increase the value of the data products. I made the connection with the organization through a friend and GISDE alumnus, who was a long-time employee of Calthorpe Analytics and had created the core of their spatial analytical processes before moving on. My work centered on Python programming, advanced spatial analytics, and data science. I was primarily focused on the research and development aspect, in that I was not there long enough to take on primary longitudinal responsibility for a client project. I thoroughly immersed myself in the workplace culture, a free-wheeling intellectual environment that was extremely open to innovation, paired with weekly bike rides and weekend adventures. My work drove my programming skill and technical GIS toolset to a much higher plane, teaching me powerful new methods, tools, and approaches that have become integral to my GIScience workflows.

Organization

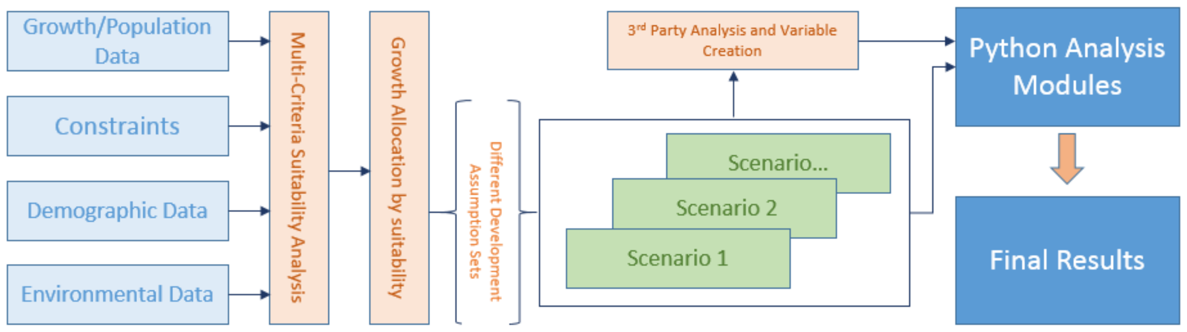
Calthorpe Analytics is an urban planning and software development company that specializes in analyzing and modeling the long-term outcomes of policy decisions. Their particular focus is the impact of the built environment and land-use on financial, environmental, health, and transportation outcomes. The leadership of the organization brings tremendous experience in urban planning to bear on the technical specificity of modeling. It is a sister firm to Calthorpe Associates, an urban design, planning and architecture firm founded by Peter Calthorpe, a leader in sustainability and smart urbanism. The client base is municipalities, municipal associations, national policy groups, environmental organizations and state and national governments.

Calthorpe Analytics has two major teams: Analytics, which performs scenario analysis for clients, and through this process develops and improves the firm's analytical methods and models, and Development, a software development team which is creating a planning-specific GIS, *Urban Footprint*, that incorporates the firm's core analytic structures and models, and will deliver that functionality directly to customers.

The firm's three major analytical packages are *Urban Footprint*, a web-based planning-specific GIS, *RapidFire*, a non-spatially informed Excel spreadsheet-based planning package, and the core analytical package that informs client projects, which is a mix of spatial and non-spatial models implemented in Python and PostgreSQL. All three packages are built in-house, or based on house-developed models.

The basic structure of the analysis performed by Calthorpe focuses on suitability analysis to model different future scenarios based on client-supplied policy assumptions. This creates a set of scenarios representing different projections of the spatial distribution of growth within a municipality's extent. The suitability analysis looks at series of metrics, unit normalized, and weights them, creating a single 0-1 overall suitability score, with development barriers masked out. Growth is then assigned by suitability rank until the growth has been allocated. Once the scenarios are created, the tabular results are used, along with limited third-party transportation analysis, to run non-spatial, multi-tiered multivariate and log models, producing a series of topic-specific outputs for a series of metrics, including environmental, transportation, health, and economic outcomes.

Model Flow Chart:


Growth/Population Data
Constraints
Demographic Data
Environmental Data
Multi-Criteria Suitability Analysis
Growth Allocation by suitability
Different Development Assumption Sets
3rd Party Analysis and Variable Creation
Scenario...
Scenario 2
Scenario 1
Python Analysis Modules
Final Results

Mission

The firm's mission is to provide analytically rigorous future scenarios that support the goals of sustainable urban development, healthy urban environments, expanded mass and rapid transit options, and environmental protection.[1] The analytic models that are at the core of the firm's work emphasize long term outcomes, highlighting the hidden costs of rapid and poorly conceived development compared with more sustainable practices. The Analytics team goals are to provide targeted, context-informed products to clients while constantly working to increase the accuracy and integrity of the models, keeping Calthorpe at the forefront of the field. Towards this goal, there is a constant emphasis on research and development projects, and keeping abreast of technical developments in urban planning and spatial computing. The Development team's goal is to create a technologically advanced, stable, scalable, and usable piece of software that will find a place in the planning data ecosystem[2], and well as provide a platform for routine internal analysis.

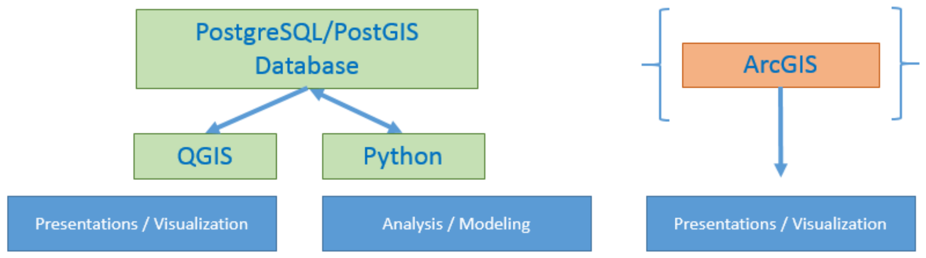
Work

Calthorpe Analytics has worked all over the country and is expanding its international presence. It has completed major projects in Utah, Ohio, Hawaii, Colorado, Wisconsin, Texas. In California they helped create Vision California, the first state-wide smart growth plan, and the long-term plan for SCAG (Southern California Association of Governments) the largest metropolitan planning organization in the country, encompassing over 18 million residents[3]. The firm has worked in Mexico, and is exploring opportunities in both India and China.

Spatial Analysis and Mapping

Geospatial information is one of the core building blocks of the company's products, and as such is pervasive in the organization. Basic fluency in Geographic Information Systems (GIS) structure and methodology, as well as strong programming skill are common at every tier of the organization. Higher level Geographic Information Science (GISci), spatial method and algorithm development, and spatial modeling happen with the Analytics team. During my internship the team as made up of 4 members with advanced degrees in engineering, transportation engineering, and urban planning. The most common GISci projects and tasks are centered on model refinement and associated variable creation. Mapping is generally undertaken for client presentations, but on the whole cartography is a very minor part of the workflow on the organization. The majority of GISci tasks are done outside of commercial GIS packages. The modeling is implemented in Python spatial libraries (Fiona, Shapely, Geopandas) and in the larger ecosystem of Python data science libraries.

GIS workflow:


PostgreSQL/PostGIS Database
QGIS
Python
Presentations / Visualization
Analysis / Modeling
ArcGIS
Presentations / Visualization

Organizational Structure and Culture

As a small company Calthorpe Analytics has a very open and level organization. The co-founder, Joe DiStefano, heads both the Planning and Software teams, and is the lead on client communication and new client acquisition. Erika Lew is lead planner and manages the Analytics team. The rest of the team will act as lead on various projects, from small technical tasks to longitudinal multi-year client projects. These tasks are assigned on both availability, and on the specific technical requirements of the job with each team member representing a specific set of strong technical/operational skills. The Development team duplicates this overall structure. Both Calthorpe Analytics and Associates are diverse in terms of gender, ethnicity and nationality at all organizational levels. The overall office is informal in both dress and demeanor, with a dedicated, self-motivated group. There are daily check-in meetings to manage work flow, weekly meetings for strategic goals, and weekly lunchtime bike rides up into the Berkeley hills.

Strengths

Calthorpe Analytics' greatest strengths are its relentless focus on innovation and improvement, and its openness to new points of view. Even the most routine structures of the analytic process are under constant review and scrutiny. Employees are encouraged to bring their particular interests and expertise to the table and will get a hearing of their ideas, often at the weekly demonstration period held every Friday. It is a chance for the whole team to sit down and show progress, describe interesting new ideas, new methods, to ask for group help on problems, and to poll the group and leadership on the feasibility/desirability of new tools. The downside of this approach is that it can be hard to keep all the many pieces running together, keep errors out of code and data, and keep the whole team up to date on the current state of the art. Going forward the company is switching to a more formal ‘release version’ of its analytic package (i.e. v1.0, v1.1, v1.2), mirroring the structure of the software development project. This would allow for more comprehensive testing and stability, as well as more ease in reporting methods to clients.

Internship Responsibilities

The core of my internship was the creation of advanced spatial analytic tools in the Python programming language, integrating the open-source Pandas and Geopandas data science libraries. My internship duties and responsibilities fell into several main categories: direct spatial and data analysis for client projects, research and development of spatial methods for future projects, integrating new data structures into existing in-house modeling software, and limited cartography for client-facing communication. My individual projects were either direct requests from within the department for a specific analysis or functionality, or self-directed exploration around expanding the overall quality or accuracy of the larger modeling environment.

The first phase of my responsibilities was learning to work in a new software environment. The Analytics department runs on Windows with Ubuntu Linux operating systems running on virtual machines for full-scale model runs. The data is kept on several large PostgreSQL databases using the PostGIS spatial extension. All the analysis is conducted in Python. I entered my internship with a good Python skillset, but without much experience with large datasets or complex spatial or data analysis. The start of my internship coincided with a new push to replace the traditional Python approach to tabular data - nested dictionaries - with a new and more powerful data-science library Pandas (PANel DAta, a reference to the three-dimensional data structure) and its spatial extension Geopandas. The fundamental goal of the Pandas library development is to bring the data-structures and analytical power of the statistical programming language R to Python. It recreates R's basic vector/matrix scheme with its attendant efficiencies of item-wise vector math, allowing for computational efficiency over large datasets. The library sits on top of the NumPy library, a core toolset for Python. At the start of my time at Calthorpe, I had more knowledge of R than my teammates and no client-specific responsibilities, so it was agreed that I would spearhead the push to integrate the new methods. This meant a lot of reading along with a set of practical problems that allowed me to find my way in the new structures. For the rest of internship, along with my other responsibilities, I was the resident reference for Pandas functionality, and often contributed code-snippets of specific operations in colleague's work. This nested naturally with learning and exploiting the power the of spatial extension of the data structure, Geopandas, and the still somewhat experimental network analysis toolset Pandana (Panda Network Analyst). Geopandas utilizes the Shapely library to create and manipulate vector spatial structures and produce Well Known Binary (WKB) format output for visualization in QGIS. It maintains a consistent projection using either EPSG or Proj4 definitions.

To further describe my internship, I will more fully describe several projects that I worked on or completed. The first is a script to draw out the per-unit average value of different types of structures across four large-scale spatial regimes (city, town, village) each with three potential land development density categories (standard, compact, urban). The script starts with multiple SQL queries to gather the requisite information from multiple tables. These are created as Pandas data frames, the basic 2d structure in the library, where they are joined on the unique geography id of the parcels, and duplicate records are dropped. The early part of every client project includes extensive data ingestion and cleaning, so I was able to assume a very high level of data integrity in my work, thus throughout my code there are relatively few cleaning and null-value testing routines. The value data in the merged table is then normalized by developed square footage. The table is then grouped by the two development categories and summed by assemblage (code in appendix, example 1).

The second example is allocating input data from base geometries to a network model that assigns all data to nodes. We needed to assign a proportion of the data in a polygon to the appropriate node, not double count data, with a node potentially capturing data from multiple polygons. The Pandana tool assigns data to nodes where all data from a polygon is set to the node closest to the polygon centroid. Our end-product analysis from this tool was the availability of employment and population within walkable distances. The native functionality was not fine-grained enough for our needs – it would be possible to traverse the network around a large polygon for a distance longer than our search distance without encountering the node associated with that polygon's data. This essentially 'hides' a polygon holding significant data from the analysis, creating an under count of the variable across the network. My strategy was to create a voronoi diagram of the network nodes and intersect it with the base geometries of the polygon set. From this I created a numerical table of the identities and relative proportions of the area of the polygons that intersect each node's voronoi polygon. The data of each polygon is then multiplied by the proportion and assigned to the original voronoi polygons. This data is then given to the network tool, which finds data in a polygon with a centroid coincident with the node, and data at every node. This allows the network solving tool to accurately find data at every node (partial code in appendix, example 2).

The third and final example is a tool for creating new network connectivity in greenfield development areas without the analyst or user having to manually add new network in areas of projected future growth. Previous analysis had used a proxy variable, number of intersections, as a metric for walkability. We replaced these with metrics of actual available resources within the walkable 'neighborhood' using the network tool above. This means however that in positing development in a polygon it needs to be integrated with the network to be part of the solver's output. The network is never visualized for the user, so there was no need to make the result 'look' like standard development, only to make it behave topologically like a road network. In this context this means that edges can only intersect at a node, and edges must connect the most efficient local set of nodes, and the edges under most circumstances must not intersect the boundary of the underlying polygon. A graphical representation of the process can be found below in figure 1. I chose Delaunay triangulation to create the new network; it is the inverse set of voronoi polygons, where points are connected if their polygons share an edge in the voronoi set. To make the nodes the tool creates a bounding box of the geometry of the parcel, populates it with 10,000 random points, clips those points to the original geometry, and performs a k-means clustering with k equal to the number of desired nodes. The resulting points are the intersections. The Delaunay set is created on those points, and the resulting edges are tested against the underlying polygon, with edges that intersect the polygon being removed, such that no node is left without connection to the rest the set, in which case the shortest valid edge to that node is retained. To find the distance to the existing network, a geometric object of radiating lines is set on each intersection, and the ids of the existing network edges it intersects are recorded along with the (x,y) of the closest point of intersection. This table is sorted, and the user-specified number of closest points are used to connect the new network to the existing network. The existing network is then edited to re-create the edges that contain the new nodes (code in appendix, example 3).

Beyond tool creation and data analysis, I did a limited amount of basic cartography, largely for client presentations. I used the functionality of the QGIS Composer Manager to make repetitive map creation more streamlined. ArcGIS was also used for some cartographic purposes, mainly for consistently symbolizing land-use classes, a case where the time and work of recreating the layer file for QGIS were considered overly onerous. Before my tenure, the whole office had migrated extremely effectively to open source tools, leading to the company no longer maintaining any advanced Esri licenses or extensions. Unfortunately, the maps I created represent client data, which precludes their inclusion in this report.

The final type of work I participated in was integrating these new structures into the existing code base, or the existing version of UrbanFootprint (UF), the in-house GIS product. A coworker recreated the Vehicle Miles Traveled (VMT) model in Pandas and then we plugged it into UF. This was a time consuming and frustrating task, trying to understand a very complicated product that had been developed over a long time with varying degrees of code comments and a very opaque structure. It became a Frankenstein's monster of suturing our code into places it was never intended to go, but it was successful for the task at hand.

I did the majority of my work on solo projects, working ad hoc with other analysts for individual tasks. We worked in an open workspace and we often used the whiteboard for thinking through algorithms, code examples, or mathematical calculations. We used the Development team as a technical resource, often using their deep coding experience to help us find our way through difficulties. Both Analytics and Development met together weekly to present work, progress, and questions for the collective. These meetings were often used to walk through ideas and get feedback.

Our collective work in Analytics represented the most visible part of the overall mission of Calthorpe Analytics. The company's 'product' is the rigor of the process and the integrity of the results. The leadership encourages, and spends a large amount of intellectual and actual capital on research and development of new and more sophisticated capabilities. My work spanned the whole of the Analytics team's responsibilities and as such gave me a thorough look at the whole process.

Assessing my Internship

My internship with Calthorpe Analytics was an amazing and fruitful part of my education in GIS. The expectations of excellence and creativity in spatial analysis and Python programming, along with the time and encouragement to work on difficult and speculative projects drove me to vastly expand my technical proficiency and understanding of the field. My first year at Clark gave me the technical skills to work productively in their system. It also sent me out with high level knowledge of spatial analytical methodology, allowing me to bring new ideas and new approaches to Calthorpe's dataset. This internship undoubtedly changed my trajectory in the field, pushing me fully towards tool and algorithm development.

It is hard to catalogue all the things I learned this summer at Calthorpe. This experience completely recreated my GIS analytical workflow, bringing the vast majority of vector analysis into Python and PostgreSQL. It taught me the core of object oriented programming, allowing me to improve the clarity and efficiency of my code. It improved my overall quantitative mental toolset, showing example of a wide variety of analytical models and structures. It taught me how to find and assimilate new technical skills and tools in a very short time. It taught me how to work in a collaborative technical context using GitHub for version control. In a purely Python context I learned the NumPy, Pandas, Geopandas, Fiona, Shapely, and PySal (The spatial-statistics package GeoDa's functionality in Python) libraries allowing me to create, clean, manipulate and analyze a wide range of data. I created machine learning tools out of the powerful SciKit-Learn family of libraries, which formed the core of my current independent study with Professor John Rogan. I learned to work with a PostgreSQL database, which I also now utilize on my desktop. I also learned significantly from the various skills and backgrounds of my coworkers, who were a constant source of interesting and useful experience.

The most important skills that I brought with me from Clark were those I learned in *Advanced Vector GIS*, and *Python/Computer Programming for GIS*. As I described in chapter 2, the entire organization is very skilled using both commercial (ArcGIS) and open source (QGIS) geographic information systems. My value to the organization began at the point where my skills graduated from GISystems to GIScience; they were looking for someone to bring in novel and appropriate ways to look at their data, and explain the meaning of the various outputs. In this regard my exposure to machine learning and advanced biophysical remote sensing from Professor John Rogan's *Advanced Remote Sensing* also provided me with important contributions. Obviously a high level of Python programming was essential, but Professor Jie Tien's set of classes also gave me the programmatic and theoretical knowledge of the inner workings of the standard geometric model, allowing me to intuitively engage with complicated new systems, building, decomposing and editing complicated geometries. Professor Ylli Kellici's *Spatial Statistics with R* and *Spatial Database Development in Practice* were both very helpful, and my knowledge of the R data model was instrumental in my success with Pandas.

This internship gave me a strong new focus in my studies at Clark, in that I have really discovered the path through the industry that I want to pursue. My coursework and planning have been shifted to help me become a credible applicant for jobs involving spatial method and algorithm creation. I believe that the experience of this summer will be very important to my post-Clark work, both in the skillsets I now know, and the kind of projects I pursue.

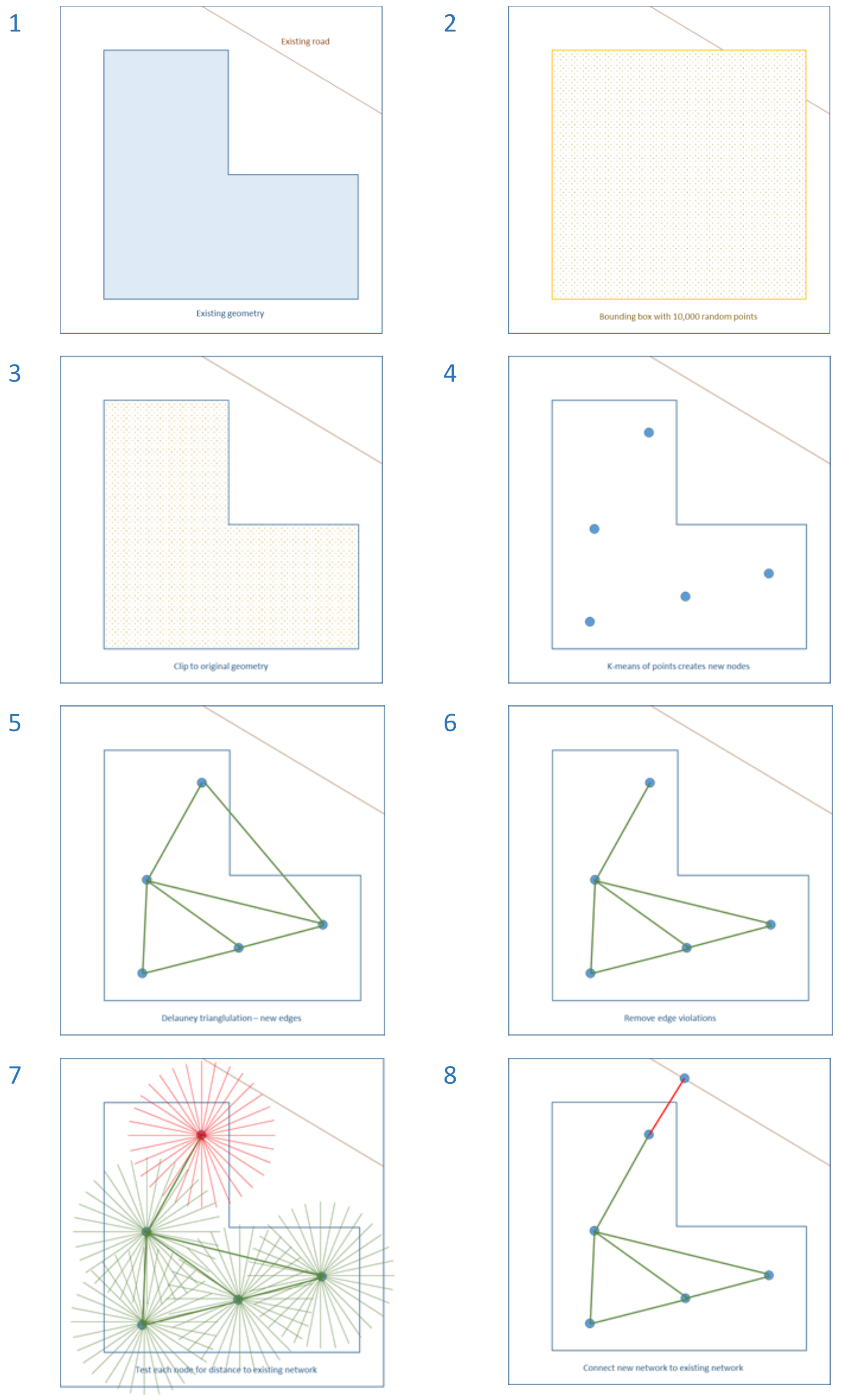
I would encourage students who are very serious about technical GIS to consider being in touch with Calthorpe, although they do not have a regular internship program. My caveat would be that unless the student feels like they can step right into enterprise-level Python coding, they might feel out of their depth in the Analytics office. For the self-selecting student however, there is a tremendous amount to learn and do if you can define your own course and demonstrate your value to the organization.

Conclusion

My internship significantly changed my trajectory in the field of Geographic Information science. I went into the summer believing that the urban planning aspect of the work would be of primary interest, but discovered that I was wholly engaged by the technical GIS and data science aspects. I am now aggressively pursuing a career in spatial method and algorithm creation, data science, and aspects of machine learning – all technical skillsets that I advanced dramatically through my internship. Calthorpe Analytics was a wonderful environment in which to spend a summer in California. The workplace culture of bike rides and foosball games kept the overall intensity of the constant drive to perform and innovate in check, making for easy in-office relationships, and healthy, happy employees.

Figures

Figure 1: New network connectivity for greenfield development:


1
Existing road
Existing geometry
2
Bounding box with 10,000 random points
3
Clip to original geometry
4
K-means of points creates new nodes
5
Delauney triangulation – new edges
6
Remove edge violations
7
Test each node for distance to existing network
8
Connect new network to existing network

Sources

1) Calthorpe Analytics 2016. *What We Do* [online]. Calthorpe Analytics.
Available from: http://calthorpeanalytics.com/index.html#questions
Accessed 12/7/2016

2) Calthorpe Analytics 2016. *Software for Sustainable Decision-Making* [online]. Calthorpe Analytics.
Available from: http://calthorpeanalytics.com/index.html#software
Accessed 12/7/2016

3) Southern California Association of Governments 2016. *SCAG; Innovating for a Better Tomorrow* [online]. Available from: www.scag.ca.gov
Accessed 12/7/2016

Appendix:

Code Examples:

Example 1:

```
from sqlalchemy import create_engine
import pandas as pd
import numpy as np
import geopandas as gpd
import shapely as shp

import matplotlib.pyplot as plt

engine = create_engine('postgresql:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx')
con = engine.connect()

pd.set_option('display.float_format', lambda x: '%.2f' % x)

pSql_city = '''
    SELECT
        parcel as parcelno,
        currentland as land_val,
        currentimpr as imp_val,
        currenttotal as sum_val
    FROM
      public.taxparcels
'''

pSql_count = '''
    SELECT
        parcelno,
        sum_landvalue as land_val,
        sum_improvementvalue as imp_val,
        (sum_landvalue + sum_improvementvalue) as sum_val
    FROM
      public.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
'''

pSql_parcels = '''
    SELECT
        wkb_geometry,
        geography_id,
        parcelno,

        pop,
        emp,
        du_detsf_sl,
        du_detsf_ll,
        du_attsf,
        du_mf,

        bldg_sqft_detsf_sl,
        bldg_sqft_detsf_ll,
        bldg_sqft_attsf,
        bldg_sqft_mf,

        (bldg_sqft_detsf_sl +
        bldg_sqft_detsf_ll +
        bldg_sqft_attsf +
        bldg_sqft_mf +
        bldg_sqft_retail_services +
        bldg_sqft_restaurant +
        bldg_sqft_accommodation +
        bldg_sqft_arts_entertainment +
        bldg_sqft_other_services +
        bldg_sqft_office_services +
        bldg_sqft_public_admin +
        bldg_sqft_education +
        bldg_sqft_medical_services +
        bldg_sqft_transport_warehousing +
        bldg_sqft_wholesale) as total_sqft
    FROM
      base_load.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
'''

pSql_ldc = '''
    SELECT
        a.land_development_category,
        a.gid,
               b.geography_id

    FROM
        base_load.xxxxxxxxxxxxxxxxxxxxxxxxxxxxx a,
        public.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx b

    WHERE
        st_intersects(b.wkb_geometry,a.wkb_geometry) AND
        st_within(st_pointonsurface(b.wkb_geometry),a.wkb_geometry)
'''

pSql_type = '''
    SELECT
        gid,
        geog_key
    FROM
      urbanfootprint_reference_datasets.xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
'''

# ### data
city_df = pd.read_sql_query(pSql_city, engine)
county_df = pd.read_sql_query(pSql_count, engine)
ldc_df = pd.read_sql_query(pSql_ldc, engine)
type_df = pd.read_sql_query(pSql_type, engine)
parcels_df = pd.read_sql_query(pSql_parcels, engine)

# ### append and clean
values_df = city_df.append(county_df)
dup_series = values_df.duplicated(subset=['parcelno'])

dups = []
for line in dup_series.iteritems():
    if line[1]:
        dups.append(line[0])

parcels_all = values_df.drop_duplicates(subset=['parcelno'])


# ### merge
parcels_df = parcels_df.merge(parcels_all, how='left', on='parcelno')
ldc_df = ldc_df.merge(type_df, how='left', on='gid')
df = parcels_df.merge(ldc_df, how='left', on='geography_id')


# ### calculations
condition = ( (base_canvas['base_pop'] + base_canvas['base_emp']) / base_canvas['base_acres_gross'] <= 2) & \
                (base_canvas['intersection_density_sqmi_focal'] < 25)

 base_canvas.loc[condition,'base_land_development_category'] = 'rural'


# In[23]:
sum_type=['land', 'imp', 'sum']
du_type =['detsf_sl', 'detsf_ll', 'attsf', 'mf']


# ##### norm by parcel area in sqft
```

```
for t in sum_type:
    df['norm_'+t+'_val'] = df[t+'_val'] / df['total_sqft']


# ##### total residential sqft and du
df['res_sqft_tot'] = df['bldg_sqft_detsf_sl'] + df['bldg_sqft_detsf_ll'] + df['bldg_sqft_attsf'] + df['bldg_sqft_mf']
df['du_tot'] = df['du_detsf_sl'] + df['du_detsf_ll'] + df['du_attsf'] + df['du_mf']

# ##### percent residential by total building sqft in parcel
df['percent_res'] = df['res_sqft_tot'] / df['total_sqft']

# ##### value / du
agg_list = ['geog_key', 'land_development_category', 'du_detsf_sl', 'du_detsf_ll', 'du_attsf', 'du_mf']
#agg_dict = {'du_detsf_sl':np.sum, 'du_detsf_ll':np.sum, 'du_attsf':np.sum, 'du_mf':np.sum}
for t in sum_type:
    for d in du_type:
        df['val_perUnit_du_'+d+'_'+t] = np.where(df['du_'+d] > 0, df[t+'_val'] / df['du_'+d], 0)
        agg_list.append('val_perUnit_du_'+d+'_'+t)
        #agg_dict['val_perUnit_du_'+d+'_'+t] = np.sum

df_sums = df[agg_list].groupby(['geog_key','land_development_category']).sum()

df_out = pd.DataFrame()
for t in sum_type:
    for d in du_type:
        df_out['ave_perUnit_du_'+d+'_'+t] = df_sums['val_perUnit_du_'+d+'_'+t] / df_sums['du_'+d]

df_out.to_csv('C:/Users/sam/Desktop/AveValueDU.csv')


# ### out to sql

df.to_sql('parcel_land_val', engine, schema='base_load', if_exists='replace')

con.execute('ALTER TABLE base_load.parcel_land_val alter column wkb_geometry set data type geometry')
```

Example 2:

```
from sqlalchemy import create_engine
import pandas as pd
import numpy as np
import pandana as pdna
import geopandas as gpd
import shapely as shp
from pandana.loaders import osm
from scipy.spatial import Voronoi

engine = create_engine('postgresql://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx')
con = engine.connect()

config_dict = dict(
    data_table='uf_loadedxxxxxxxxxxxxxxxxx',
    data_schema='base_load',
    data_geom_col='wkb_geometry',
    data_index_col='gid',
    prop_table='proportion_table',
    vor_table='voronoi_temp',
    base_schema='base_load',
    net_name='C:/Users/sam/Desktop/xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    bbox=(xxxxxxxxxx, yyyyyyyyyyyyy, xxxxxxxxxx, yyyyyyyyy), #tuple (y-min, x-min, y-max, x-max) in srid 4326 WGS 84
    agg_vars = ['acres_parcel_res','acres_parcel_emp','acres_parcel_mixed_use','du','pop','emp','emp_ret']
)

class vor_data_assign(object):
    ## creates nodes from predefined network saved in HDF5 format (net_name = path to file)
```

```
def __init__(self, settings):
    self.s = settings

    net_data = pd.HDFStore(self.s['net_name'])
    self.nodes = net_data.nodes

    # Formats dict input
    var_list = []
    temp = self.s['data_index_col']+', '
    for string in self.s['agg_vars']:
        if string == 'pop':
            temp += 'pop as pop1, '
            var_list.append('pop1')
        else:
            temp += string+', '
            var_list.append(string)

    self.s['data_string'] = temp[:-2]
    self.var_list = var_list

def create_new_osm_network(self):
    self.network = osm.network_from_bbox(self.s['bbox'][0], self.s['bbox'][1],self.s['bbox'][2], self.s['bbox'][3])
    self.network.save_hdf5(self.s['net_name'])

## creates network
def create_network(self, out=False):
    print 'Building network...'
    self.network = pdna.Network.from_hdf5(self.s['net_name'])
    print 'Done'

## creates voronoi table
def vor_from_nodes(self):

    print 'Creating Voronoi Table...'
    point_array = self.nodes.values
    vor = Voronoi(point_array)

    # creates shapely polygons from valid voronoi regions (polygons)
    region_poly = []
    for region in vor.regions:
        if len(region)>2 and -1 not in region:
            poly = []
            for point in region:
                poly.append(vor.vertices[point])
            region_poly.append(shp.geometry.Polygon(poly))

    #create geopanda and shapely polygon for geoprocessing (within)
    region_gdf = gpd.GeoDataFrame(geometry=region_poly)
    bbox_clip = shp.geometry.Polygon([[self.s['bbox'][1],self.s['bbox'][0]],
                                      [self.s['bbox'][3],self.s['bbox'][0]],
                                      [self.s['bbox'][3],self.s['bbox'][2]],
                                      [self.s['bbox'][1],self.s['bbox'][2]]])

    region_gdf = region_gdf[region_gdf.within(bbox_clip)]

    # df of wkb geoetry of regions
    region_df = pd.DataFrame(map(lambda x: shp.wkb.dumps(x).encode('hex'), region_gdf.geometry), columns=['geometry'])

    # writes voronoi polygons to Postgres, alters table for geometry, index, removes invalid polygons
    region_df.to_sql(self.s['vor_table'], engine, schema=self.s['schema'], if_exists='replace')

    con.execute('ALTER TABLE {base_schema}.{vor_table} ALTER COLUMN geometry SET DATA TYPE geometry'.format(**self.s))
    con.execute('CREATE index ON {base_schema}.{vor_table} USING gist(geometry)'.format(**self.s))
    con.execute('DELETE FROM {base_schema}.{vor_table} WHERE NOT st_isvalid(geometry)'.format(**self.s))

    del region_poly, region_gdf, region_df
    print 'Done'

## Creates proportion table - long run time (8+ hours)
def proportion_table(self):
        psql_prop = '''
            CREATE TABLE {base_schema}.{prop_table}
            AS SELECT
            a.{data_index_col} as block_id,
            st_area(st_transform(a.{data_geom_col}, 4326)) as block_area,
            b.index as poly_id,
            st_setsrid(b.geometry,4326) as geom,
            st_centroid(st_setsrid(b.geometry,4326)) as centroid,
            st_area(st_setsrid(b.geometry,4326)) as poly_area,
            st_area(st_intersection(st_makevalid(st_transform(a.{data_geom_col}, 4326)),     st_setsrid(b.geometry,4326))) as sub_area,
            st_area(st_intersection(st_makevalid(st_transform(a.{data_geom_col}, 4326)), st_setsrid(b.geometry,4326))) /
st_area(st_transform(a.{data_geom_col}, 4326)) as percent_of_block,
            st_area(st_intersection(st_makevalid(st_transform(a.{data_geom_col}, 4326)), st_setsrid(b.geometry,4326))) /
st_area(st_setsrid(b.geometry,4326)) as percent_of_poly

            FROM
            {data_schema}.{data_table} a,
            {base_schema}.{vor_table} b

            WHERE
            st_intersects(st_transform(a.wkb_{data_geom_col}, 4326), st_setsrid(b.geometry,4326))
            '''.format(**self.s)

        print 'Creating Proportion Table...'
        con.execute(psql_prop,engine)
        print 'Done'

    #computes the aggregation of the specified vars proportionally to their avaibility in the voronoi polygons
    def compute_network(self, net=''):

        #data
        pSql_data='''
            SELECT
            {data_string}
            FROM
            {data_schema}.{data_table}
            '''.format(**self.s)

        pSql_prop='''
            SELECT
            block_id,
            poly_id,
            st_x(centroid) as x,
            st_y(centroid) as y,
            percent_of_block,
            percent_of_poly
            FROM
            {base_schema}.{prop_table}
            '''.format(**self.s)

        try:
            df_prop = pd.read_sql_query(pSql_prop, engine)
        except:
            user_imp = raw_input('Proportion Table Not Found: Create Now (8+ hours run) y/n : ')

            if user_imp in ('y', 'Y'):
                self.vor_from_nodes()
                self.proportion_table()
                df_prop = pd.read_sql_query(pSql_prop, engine)
            else:
                return

        print 'Computing Network Aggregation...'

        df_data = pd.read_sql_query(pSql_data, engine, index_col=self.s['data_index_col'])

        #creates master df
        df = df_prop.merge(df_data, how='left', left_on='block_id', right_index=True).astype(object)
```

```
## Aggregate data on Voronoi Polygon
    sum_df = pd.DataFrame()
    sum_df['poly_id'] = df.poly_id

    # create block-proportial sums
    for var in self.var_list:
        sum_df[var+'_poly'] = df[var] * df.percent_of_block


    #group by polygon id
    group = sum_df.groupby('poly_id')

    #apply np.sum to all block-proportional data columns
    var_dict = {}
    for var in self.var_list:
        var_dict[var+'_poly'] = np.sum

    agg_df = group.agg(var_dict)

    # Data reassigned spatial context
    df_coords = df[['poly_id', 'x','y']].drop_duplicates('poly_id')
    df_poly_sum = agg_df.merge(df_coords, how="left", left_index=True, right_on='poly_id')

    del df_coords

## Network set
    x, y = df_poly_sum.x, df_poly_sum.y

    #look for preexisting network (avoids multiple graph memory issue in pandana)
    if net=='':
        try:
            df_poly_sum['node_ids'] = self.network.get_node_ids(x,y)
        except:
            self.create_network()
            df_poly_sum['node_ids'] = self.network.get_node_ids(x,y)
            print 'Continuing Network Aggregation...'
    else:
        self.network = net
        df_poly_sum['node_ids'] = self.network.get_node_ids(x,y)


    self.network.precompute(401)

    # create dataframe, assign data to the network by node, and populate with network aggregation
    df_net_results = pd.DataFrame()
    for var in self.var_list:
        self.network.set(df_poly_sum.node_ids, variable=df_poly_sum[var+'_poly'], name=var)
        df_net_results[var+'_net_sum'] = self.network.aggregate(401,type="sum", decay="flat", name=var)

    df_poly_sum = df_poly_sum.merge(df_net_results, how='left', left_on='node_ids', right_index=True)

    df = df.merge(df_poly_sum, how='left', on='poly_id')

    del df_poly_sum, df_net_results
```

Example 3:

```
###########################################################################################
#
#    attributes:
#
#    bbox - bounding box (xmin, ymin, xmax, ymax) in 4326 for osm network ingest
#    net_name - location of HDF5 network store
#    pSql - sql string for data
#    crs_data - epsg code for projected data
#    engine - connection engine
#    con - engine.connect object
#    net_data - HDF5 data store object
#    nodes - master list of nodes
#    m_nodes_gdf - master gdf of nodes
#    edges - master list of edges
#    edges_raw - copy of store 'edges'
#    data_gdf - gdf of data_gdf
#    m_edges_gdf - master gdf of edges
#    nodes_gdf - gdf of new nodes
#    poly - shapely polygon of geometry
#    basic_poly - shapely polgon of nodes exterior ring
#    edges_gdf - gdf of new edges
#    prox_table - df of closest network edges to nodes of new subnet (ordered)
#    connect_id - index value of node inserted into existing network
#
#    methods:
#
#    refresh_network() - re-ingests a network from osm (by bbox), writes to HDF5 store
#    data_in() - creates core df and gdf tables
#    find_new_nodes(geography_id, # of intersections(n)) - finds set of n new nodes within geometry
#    edge_poly() - creates simple edge set from exterior ring of new nodes
#    edge_delaunay() - creates complex edge set based on delaunay triangulation
#    trim_net() - removes edges from set where the edge intersects the underlying geometry,
#              does not allow for orphaned nodes
#    find_closest_point() - creates an ordered table of information on the closest edges from the new nodes
#    add_connect_edge(series from above table) - addes an edge from point to closest edge
#    replace_intersect_edge(series from above table) - replaces closest edge with two edges linking to network
#    add_nodes_HDF5() - adds new nodes to HDF5 store
#    add_edges_HDF5() - adds new edges to HDF5 store
#
################################################################################################


    def __init__(self, user_dict):

        self.bbox = user_dict['bbox']
        self.net_name= user_dict['net_name']
        self.pSql = user_dict['psql'].format(schema=user_dict['schema'], data_table=user_dict['data_table'])
        self.crs_data = user_dict['crs_data']

        self.engine = create_engine(user_dict['connect_string'])
        self.con = self.engine.connect()

    def refresh_network(self, num=2):

        try:
            pdna.network.reserve_num_graphs(num)
        except:
            pass

        self.network = osm.network_from_bbox(self.bbox[0],self.bbox[1],self.bbox[2],self.bbox[3])
        self.network.save_hdf5(self.net_name)

    def data_in(self):

        self.net_data = pd.HDFStore(self.net_name)
        nodes, edges = self.net_data.nodes, pd.DataFrame(self.net_data.edges.values, columns=['from', 'to', 'imp'])
        edges_raw = self.net_data.edges

        gdf = gpd.read_postgis(self.pSql, self.engine, geom_col='geometry')
        gdf.crs = {'init' : self.crs_data} #sets crs to match values

        nodes_gdf = gpd.GeoDataFrame(geometry=map(lambda x, y: shp.geometry.Point(x, y), nodes['x'], nodes['y']))
        nodes_gdf.index = nodes.index
        nodes_gdf.crs = {'init' :'epsg:4326'} # sets crs to match osm standard
        nodes_gdf.to_crs({'init' : self.crs_data}, inplace=True) # reprojected to match data

        edges = edges.merge(nodes, how='left', left_on='from', right_index=True)
        edges = edges.merge(nodes, how='left', left_on='to', right_index=True, suffixes=['_from', '_to'])
        edge_array = edges[['x_from', 'y_from', 'x_to', 'y_to']].as_matrix()
        edge_list = []
        for edge in edge_array:
            edge_list.append(shp.geometry.LineString([(edge[0], edge[1]), (edge[2], edge[3])]))

        edges_gdf = gpd.GeoDataFrame(geometry=edge_list)
        edges_gdf = edges_gdf.merge(edges[['from','to','imp']], how='left', left_index=True, right_index=True)
        edges_gdf.crs = {'init' :'epsg:4326'}
        edges_gdf.to_crs({'init' : self.crs_data}, inplace=True)

        self.nodes, self.m_nodes_gdf, self.edges, self.edges_raw = nodes, nodes_gdf, edges, edges_raw
        self.data_gdf, self.m_edges_gdf = gdf, edges_gdf

    def find_new_nodes(self, id_num, intersections): #data gdf, id, number of intersections to create

        geo_poly = self.data_gdf[self.data_gdf.geography_id == id_num].geometry.values[0]
        poly = self.data_gdf[self.data_gdf.geography_id == id_num].geometry.bounds.values

        minx, miny, maxx, maxy = poly[0][0], poly[0][1], poly[0][2], poly[0][3]
        rand_points_list = []

        #random points
        for n in range(0,10000):
            x_rand, y_rand = np.random.randint(minx, maxx), np.random.randint(miny, maxy)
            rand_points_list.append(shp.geometry.Point(float(x_rand), float(y_rand)))

        rand_points_gdf = gpd.GeoDataFrame(geometry=rand_points_list)
        rand_points_gdf.crs = {'init' :'epsg:32154'}

        #points within geometry
        point_obj = shp.ops.cascaded_union(rand_points_gdf.geometry)
        points = point_obj.intersection(geo_poly)

        #Kmeans clusters
        est_mini = MiniBatchKMeans(init='random', n_clusters=intersections, n_init=10)
        fit = est_mini.fit(points)

        #nodes from cluster centers
        nodes_list = []
        for point in fit.cluster_centers_:
            nodes_list.append((point[0], point[1]))

        #arranges points counter clock-wise
        mean_x = np.sum(x[0] for x in nodes_list) / len(nodes_list)
        mean_y = np.sum(x[1] for x in nodes_list) / len(nodes_list)

        def arctan(x):
            return(math.atan2(x[0] - mean_x, x[1] - mean_y) + 2 * math.pi) % (2 * math.pi)

        l = sorted(nodes_list, key=arctan)
        out_poly = shp.geometry.Polygon(l)

        #geometry from reordered points
        geom_list =[]
        for point in l:
            geom_list.append(shp.geometry.Point(point[0], point[1]))
            new_nodes_gdf = gpd.GeoDataFrame(geometry=geom_list)
            new_nodes_gdf['coords'] = l

            #assigns ids from master node gdf
            new_ids = []
            ids = self.nodes.index.max() + 1
            for n in range(len(new_nodes_gdf.geometry)):
                new_ids.append(ids + n)

            new_nodes_gdf['index_']=new_ids
            #new_nodes_gdf.set_index('index', inplace=True)

            self.nodes_gdf, self.poly, self.basic_poly = new_nodes_gdf, geo_poly[0], out_poly

        def edge_poly(self):  #creates to/from indexed edges from exterior edges of polygon

            coords = self.basic_poly.exterior.coords

            line_list = []
            id_list = []
            length_list = []

            for i in range(len(coords) -1):
                line = shp.geometry.LineString((coords[i], coords[i+1]))
                line_list.append(line)
                length_list.append(line.length)
                if self.nodes_gdf.index[i] == self.nodes_gdf.index[-1]:
                    id_list.append((self.nodes_gdf.index_[i], self.nodes_gdf.index_[0]))
                else:
                    id_list.append((self.nodes_gdf.index_[i], self.nodes_gdf.index_[i+1]))

            edges_gdf = gpd.GeoDataFrame(geometry=line_list)
            edges_gdf['edge_id'] = id_list
            edges_gdf['distance'] = length_list

            self.edges_gdf = edges_gdf

        def edge_delaunay(self):  #creates to/from indexed edges of all unique edges created by DeLaunay triangulation of new nodes

            array = []
            for i in range(len(self.nodes_gdf.coords)):
                array.append(self.nodes_gdf.coords.values[i])

            np.asarray(array)

            mesh = Delaunay(array)

            id_list = []
            line_list = []
            length_list = []

            for simp in mesh.simplices:
                for i in range(0,3):
                    if i == 2:
                        line = shp.geometry.LineString([self.nodes_gdf.coords[simp[i]],self.nodes_gdf.coords[simp[0]]])
                        id_list.append((self.nodes_gdf.index_[simp[i]],self.nodes_gdf.index_[simp[0]]))
                        line_list.append(line)
                        length_list.append(line.length)
                    else:
                        line = shp.geometry.LineString([self.nodes_gdf.coords[simp[i]],self.nodes_gdf.coords[simp[i+1]]])
                        id_list.append((self.nodes_gdf.index_[simp[i]],self.nodes_gdf.index_[simp[i+1]]))
                        line_list.append(line)
                        length_list.append(line.length)

            edges_gdf = gpd.GeoDataFrame(geometry=line_list)
            edges_gdf['edge_id'] = id_list
            edges_gdf['distance'] = length_list

            edge_copy = edges_gdf.copy()
            for i in edge_copy.index:
                for j in edges_gdf.index:
                    if i != j:
                        if edge_copy.geometry[i].equals(edge_copy.geometry[j]):
                            edges_gdf.drop(i, inplace=True)

            self.edges_gdf = edges_gdf

        def trim_net(self): # removes edges that intersect the boundary of the root geography

            outline = shp.ops.cascaded_union(self.poly.exterior)

            nodes = []
            drop_list = []
            for line in self.edges_gdf.iterrows():
                if line[1][0].intersects(outline):
                    drop_list.append(line[0])
                else:
                    nodes.append(line[1][1][0])
                    nodes.append(line[1][1][1])

            lost_nodes = []
            for row in self.nodes_gdf.iterrows():
                if row[1]['index_'] not in nodes:
                    lost_nodes.append(row[1]['index_'])

            keep_line = ''
            keep_length = 10000
            for line in drop_list:
                if self.edges_gdf.loc[line][1][0] in lost_nodes or self.edges_gdf.loc[line][1][1] in lost_nodes:
                    if self.edges_gdf.loc[line][0].length < keep_length:
                        keep_length = self.edges_gdf.loc[line][0].length
                        keep_line = line

            if keep_line in drop_list:
                drop_list.remove(keep_line)

            for line in drop_list:
                self.edges_gdf.drop(line, inplace=True)

            self.edges_gdf.reset_index(drop=True, inplace=True)

        def find_closest_point(self):

            close_dict = {}
            index = 0
            centers = self.nodes_gdf.coords
            for center in centers:

                #defines line of length 1000 'north' of point
                x, c_y, e_y = center[0], center[1], center[1] + 1000
                line = shp.geometry.LineString([(x, c_y), (x, e_y)])

                #creates radial 'star' of lines
                center_point = shp.geometry.Point(x, c_y)
                radii= [shp.affinity.rotate(line, i, (x,c_y)) for i in range(0,360,10)]
                mergedradii = shp.ops.cascaded_union(radii)

                #set of all network edges that intersect with 'star' object
                edge_set = self.m_edges_gdf[self.m_edges_gdf.geometry.intersects(mergedradii)]

                #finds closest edge
                min_dist = ''
                line_num = ''
                for line in edge_set.iterrows():
                    d = center_point.distance(line[1].geometry)
                    if min_dist == '':
                        min_dist = d
                        line_num = (line[1].name)
                    elif d < min_dist:
                        min_dist = d
                    line_num = (line[1].name)

                #set of intersection points on closest network edge
                union_obj = mergedradii.intersection(edge_set.ix[line_num, 'geometry'])
                try:
                    intersection_points = list(union_obj.geoms)

                    #finds closest point within set of points
                    min_dist = ''
                    point_num = ''
                    i = 0
                    for point in intersection_points:
                        d = center_point.distance(point)
                        if min_dist == '':
                            min_dist = d
                            point_num = i
                        elif d < min_dist:
                            min_dist = d
                            point_num = i
                        i+=1
                    out_xy = intersection_points[point_num].coords
                except:
                    min_dist = center_point.distance(union_obj)
                    out_xy = union_obj.coords

                #adds index:tuple of results for each point
                close_dict[index] = (line_num, min_dist, out_xy)
                index += 1

            out = pd.DataFrame.from_dict(close_dict, orient='index')
            out.columns = ['line_number', 'length', 'coords']
            out['to'] = self.m_edges_gdf.loc[out.line_number.tolist()]['to'].values
            out['from'] = self.m_edges_gdf.loc[out.line_number.tolist()]['from'].values
    #        out['to_coords'] = self.m_edges_gdf.loc[out.line_number.tolist()]['to'].values
            out.index.names = ['node']
            out.sort_values('length', inplace=True)
            out.drop_duplicates('line_number', inplace=True)
            out.reset_index(inplace=True)

            self.prox_table = out

        def add_connect_edge(self, line): #line - series from self.prox_table

            #adds the node that is the intercept point handed from the find_closest_point() output tuple
            self.nodes_gdf = self.nodes_gdf.append({'geometry':shp.geometry.Point(line['coords'][0]),                    'coords':line['coords'][0],
    'index_':self.nodes_gdf.index_.max() + 1}, ignore_index=True)

            #creates a line from the intercept point to the closest node, calculates distance
            line_out = shp.geometry.LineString([line['coords'][0],self.nodes_gdf.loc[line['node']].geometry.coords[0]])
            length = line_out.length

            #appends the line to line_gdf
            self.edges_gdf = self.edges_gdf.append({'geometry':line_out, 'edge_id':(self.nodes_gdf.index_.max(),
    self.nodes_gdf.loc[line['node']].index_), 'distance':length}, ignore_index=True)

        def replace_intersect_edge(self, line): #line - series from self.prox_table
            #drop existing edge
            self.edges_raw.drop((line['to'], line['from']), inplace=True)

            #gets index of connection node
            self.connect_id = self.nodes_gdf.loc[line.node]['index_']

            #create new connection edges (to)
            to_line = shp.geometry.LineString([line['coords'][0], self.m_nodes_gdf.loc[line['to']].geometry.coords[0]])
            self.edges_gdf = self.edges_gdf.append({'geometry':to_line, 'edge_id':(line['to'],
    self.nodes_gdf.loc[line['node']].index_),                    'distance':to_line.length}, ignore_index=True)
            #create new connection edges (from)
            from_line = shp.geometry.LineString([line['coords'][0], self.m_nodes_gdf.loc[line['from']].geometry.coords[0]])
            self.edges_gdf = self.edges_gdf.append({'geometry':from_line, 'edge_id':(line['from'],
    self.nodes_gdf.loc[line['node']].index_),                    'distance':from_line.length}, ignore_index=True)
```

```
def add_nodes_HDF5(self): #new node gdf, existing nodes, HDF5 store obj

    #converts calculated index into gdf index obj
    self.nodes_gdf.set_index('index_', inplace=True)

    #reverts projection to geographic
    self.nodes_gdf.crs = {'init' :'epsg:32154'}
    self.nodes_gdf.to_crs({'init' :'epsg:4326'}, inplace=True)

    #converts points to coordinates for network HDF5
    xlist, ylist = [], []
    for tup in self.nodes_gdf.geometry.values:
        xlist.append(tup.x)
        ylist.append(tup.y)

    self.nodes_gdf['x'], self.nodes_gdf['y'] = xlist, ylist

    #writes nodes to HDF5
    self.nodes = self.nodes.append(self.nodes_gdf[['x','y']])
    self.net_data['nodes']=self.nodes

def add_edges_HDF5(self): #new edges, raw edge df from store, HDF5 store obj

    #create multiIndex obj
    tuples = list(self.edges_gdf.edge_id)
    index = pd.MultiIndex.from_tuples(tuples)

    #creates a formated output df
    out = pd.DataFrame(columns=['from','to','distance'], index=index)
    for line in self.edges_gdf.iterrows():
        out.loc[line[1][1]] = [line[1][1][0],line[1][1][1],line[1][2]]

    out[['from', 'to']] =  out[['from', 'to']].astype(np.int64)
    out['distance'] =  out['distance'].astype(np.float64)

    #writes edges to HDF5
    self.edges_raw = self.edges_raw.append(out)
    self.net_data['edges']=self.edges_raw
```