2019

Machine Learning Algorithms and Applications for Lidar, Images, and Unstructured Data

Biswas Parajuli

FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

MACHINE LEARNING ALGORITHMS AND APPLICATIONS FOR LIDAR, IMAGES, AND UNSTRUCTURED DATA

By

BISWAS PARAJULI

A Dissertation submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2019

Biswas Parajuli defended this dissertation on April 19, 2019.
The members of the supervisory committee were:

Piyush Kumar

Professor Directing Dissertation

Yiyuan She

University Representative

Xiuwen Liu

Committee Member

Peixiang Zhao

Committee Member

The Graduate School has verified and approved the above-named committee members, and certifies that the dissertation has been approved in accordance with university requirements.

To my parents, Bidya and Bishnu

ACKNOWLEDGMENTS

On comparing my current self to the one from the past, when I started out on this journey, I realize that the process has transformed me with life lessons, learning and invaluable experiences. I am thankful to everyone, for every cultural and political ripple, and for every random universal perturbation for contributing towards this evolution and I hope for its continuity.

First and foremost, I would like to express my deepest appreciation to Piyush for being the beacon in guiding me all along not just to this dissertation as the final destination but also towards a more exploratory perspective for the future. His ideas and nudges were integral to this work and its completion would not have been possible without them. I am extremely grateful to Dr. Xiuwen Liu, Dr. Peixiang Zhao and Dr. Yiyuan She for being in the committee, providing feedback and always helping me in the process with open hearts. I am highly indebted to Dr. Fengfeng Ke for her support and also the opportunity to be a part of her interdisciplinary project.

I cherish the moments of euphoria and frustration in the lab that I got a chance to share with Soheila and Ahana while running experiments, engaging in necessary or unnecessary questions and discussions, or during breaks depending on the degree of bitterness of coffee. Many thanks to Tathagata who I was fortunate to collaborate with on multiple projects. I would also like to thank Dr. Sonia Haiduc and Mohammad for the chance to work on a joint project. I have also been helped, constructively criticized, encouraged, uplifted and motivated by many as friends, mentors, roommates, colleagues, staff and strangers who I thank from the bottom of my heart.

I am thankful to the Department of Computer Science, NSF and AFRL (Dr. Eduardo Pasiliao) for continued support and funding for my education. Lastly, I owe it all to my parents and their struggle which put me in a position of privilege so that I could embark on this long trip.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Aerial imagery of geographic regions in the form of Lidar and RGB images aids different tasks like survey, urban-planning, mapping, surveillance, navigation, localization and others. Most of the applications, in general, require accurate segmentation and identification of variety of objects. The labeling is mostly done manually which is slow and expensive. This dissertation focuses on roads as the object of interest and aims to develop methods to automatically extract road networks from both aerial Lidar and images. This work investigates deep convolutional architectures that can fuse the two types of data for road segmentation. It presents a design which performs better than the state-of-the-art RGB-only methods. It also describes a simple, disk-packing based algorithm which translates the road segmentation into a OpenStreetMap-like road network graph while giving improved accuracies in terms of connectivity, topology and reduction in outliers.

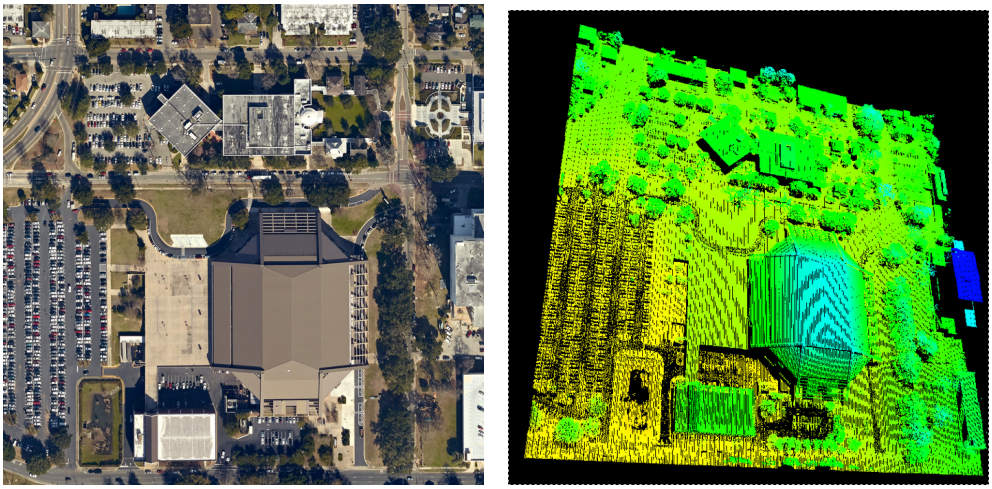
This dissertation also presents a truth finding algorithm based on iterative outlier removal which can be used for reaching a consensus when information sources or ensembles of trained machine learning models are at a conflict. In addition, it introduces a full and published book on Python programming based on the experiences this research provided. The hope is to contribute towards teaching and learning Python.

CHAPTER 1

INTRODUCTION

Physical roads stretch across the geographic terrain forming a network. For computational purposes, the road network can be represented as a graph embedded in a three-dimensional space where the road centerlines form the edges and each node is defined by a tuple of three floating point numbers, namely, latitude, longitude and height. These road network graphs lie at the heart of many map oriented tasks, including, self-driving cars, localization [3] and automatic drone navigation [4]. Understanding traffic patterns on the urban road networks also helps in reducing commute times. Cities can plan safer cycling routes based on the extracted road networks. Moreover, road mapping is a ubiquitous task used by industries across all spectrum. Apart from the proprietary Apple and Google Maps, one of the most widely used open source mapping systems is OpenStreetMap (OSM). These mapping services mostly rely on manual labeling from volunteers [5]. OSM is currently in need of a revamp in terms of diversifying its applications [6] and improved correctness. Bastani et. al [7] report a discrepancy of 14% between OSM and TorontoCity Dataset. Resolution of these issues requires a system which automatically extracts accurate road network graphs from publicly available datasets. The output of this system can augment manual road network mapping and save time because the map curators and users can simply adjust the created graph overlay. It also helps in correcting existing road network maps.

The main goal of this dissertation is to utilize multi-sensor terrain data acquired from the aerial viewpoint for extracting accurate road network graphs. The dissertation addresses challenges that are encountered in the pursuit of the goal. First, the aerial data constitutes two different types of data, namely, high resolution multi-channeled images and air-borne Lidar. Thus, disparity-reducing data fusion methods are central to this dissertation which are explored in Chapter 2. These methods combine both Lidar and images and use state of the art deep convolutional neural networks to produce road segmentation. Second, since the



(a) *RGB* (b) *Lidar*

Figure 1.1: A geographic region visualized as an RGB image (Figure 1.1a) and as a three dimensional point cloud (Figure 1.1b) from the aerial viewpoint. The point cloud visualization is color coded with respect to the point height values.

data acquisition process typically involves multiple sensors, consensus should be established among the sensors. Algorithms to do the same are presented in Chapter 4. To achieve the final goal, Chapter 3 presents a simple, disk-packing based algorithm that converts road segmentations into undirected road network graphs with higher accuracy in terms of connectivity, topology and noise compared to the known state-of-the-art methods. The final chapter, Chapter 6, describes the development process of a book on Python, titled "Python Application Programming" and incorporates experiences gathered during the course of this research. Below, a preview of the main challenges concludes this chapter.

Before we extract road network graphs, we need to first locate where the roads are. This is known as the road detection or road segmentation problem and we want to solve it with the help of aerial geographic data. Aerial sensors collect information about the geographic terrain as high resolution multi-channeled images and three-dimensional Lidar point clouds. The images generally encode signals in the visual spectrum as RGB values (Figure 1.1a). Lidar captures the geographic coordinates of the aerially visible surface (and sometimes invisible surface if the last returns are also taken). A software visualization of Lidar corresponding to

Figure 1.1a is shown in Figure 1.1b. Lidar data is a set of points and each point is defined by three coordinates, namely, latitude, longitude and height. If we want to leverage both the datasets for the task of road detection, we face a few challenges. We discuss them in the remainder of this section.

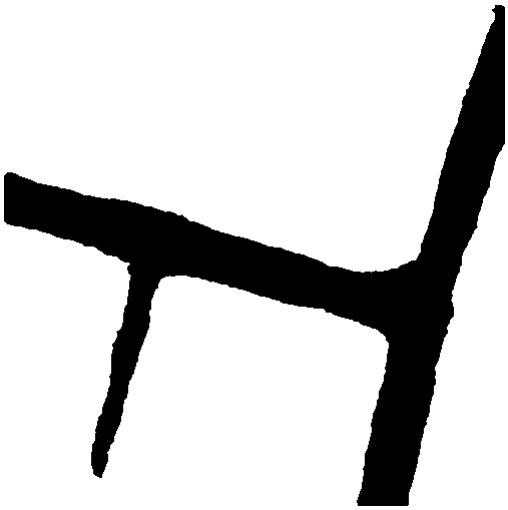


Figure 1.2: Road segmentation

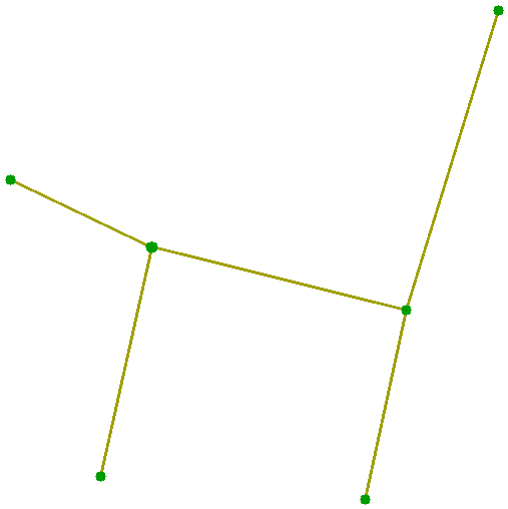


Figure 1.3: Road network graph

Road segmentation algorithms produce per-pixel road predictions for image-like inputs (Figure 1.2). These predictions which lie in a two dimensional plane are usually rasterized and noisy and appear as disconnected blobs or ribbons of different shapes and sizes. There can also be areas like parking lots and roof tops which are classified as roads. The translator should replace the true positive regions with the corresponding centerlines and omit the false positive regions to finally produce a graph (Figure 1.3). A separate edge prediction mechanism may be required to address the false negative regions which correspond to the discontinuities in the output segmentation.

The ability to efficiently decipher truthful information has gained importance due to the presence of highly networked infrastructure, including online resources, social media, internet of things and others. We frequently get conflicting information from multiple sources about an item of interest and need to make a choice as to which source to trust. In relation to this dissertation, there can be multiple aerial sensors collecting data for the same geographic region. Some of the sensors may be faulty and can record contending information. Thus,

it would be useful to have an algorithm that is able to filter out all the inconsistencies and identify the most reliable sensor.

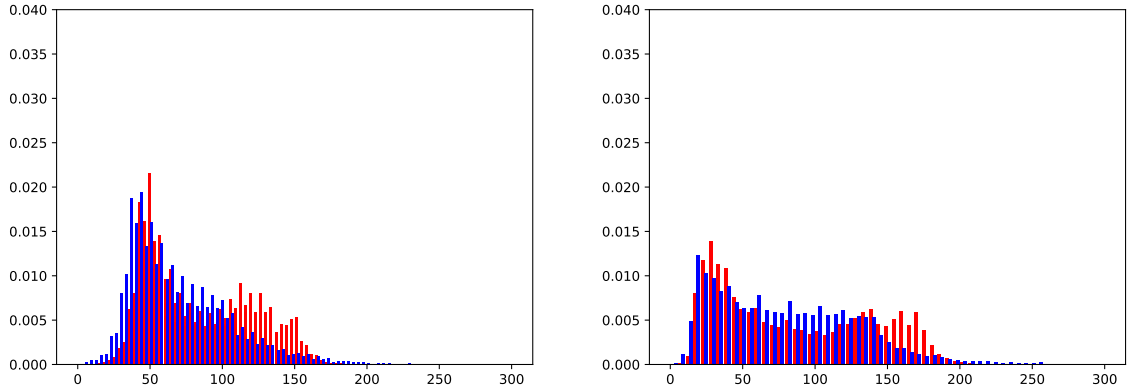
CHAPTER 2

FUSION OF AERIAL LIDAR AND IMAGES FOR ROAD SEGMENTATION

2.1 Introduction

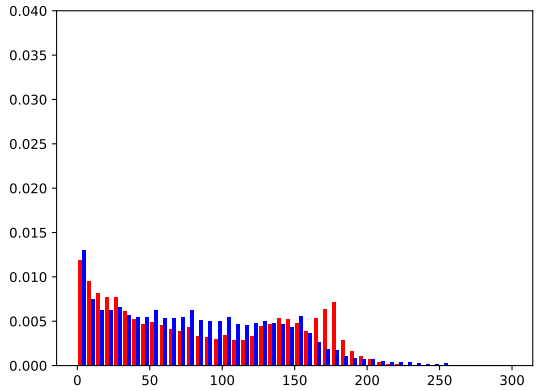
The problem of identifying objects in aerial imagery arises in many applications, such as mapping [2], automatic road navigation [7], unmanned vehicles [8], natural disaster management [9], surveillance and urban planning [10]. Remotely sensed data usually constitutes very high resolution multi-channeled images defined by pixels along with air-borne Lidar defined by 3-dimensional point clouds. The cost of collecting these types of data is going down. Public and private entities, both large and small, are making big investments in them. Most of the applications, in general, require accurate segmentation and identification of variety of objects which is mostly done manually. Further, existing applications seldom utilize both Lidar and RGB images in conjunction because they differ in their modalities, their numeric value ranges differ, fusing them is not trivial and it is also computationally expensive to process them together.

In this chapter we focus on the specific problem of extracting roads from both aerial lidar and high resolution images (0.5 feet per pixel). We present fully convolutional architectures which fuse the two types of data and show their results on the problem. We use a simple Lidar processing unit, along with a stack of 3 SegNets [11] to achieve better segmentation than just using the RGB images. We also released our dataset so that it can become one of the new benchmark datasets for road extraction. The dataset download information can be found at: https://bitbucket.org/biswas/fusion_lidar_images

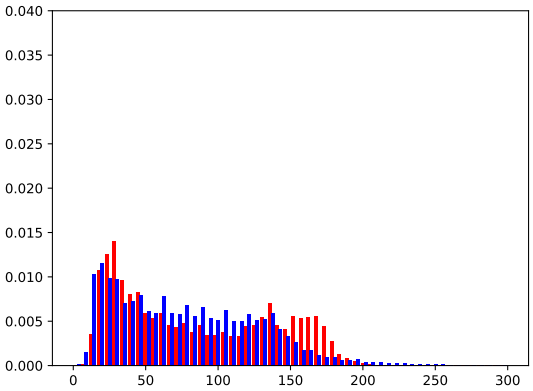


(a) *Distribution of Blue channel*

(b) *Distribution of Green channel*

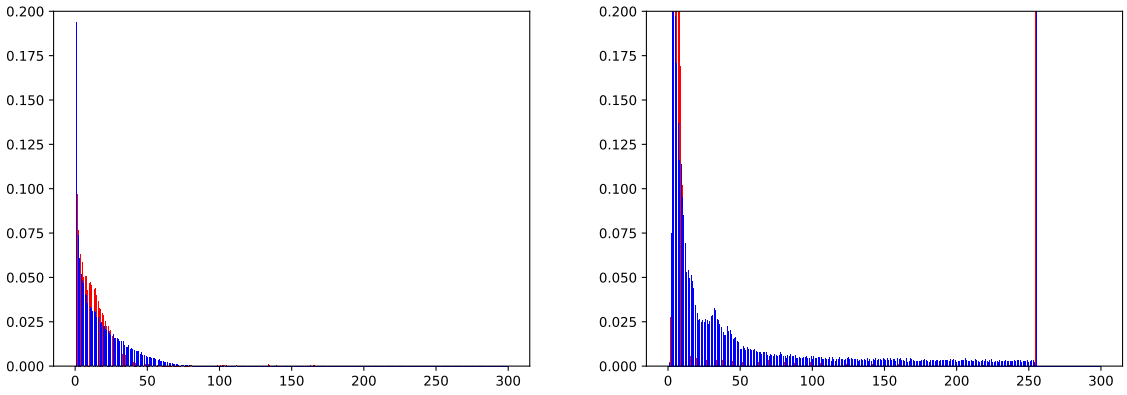


(c) *Distribution of Red channel*



(d) *Distribution of Gray channel*

Figure 2.1: Probability density of the RGB channels for 50000 random pixel positions in a random 2500×2500 tile. Color code in the histograms: red for roads and blue for non-roads.



(a) *Distribution of unscaled height values*

(b) *Distribution of LPU output*

Figure 2.2: Probability density of depth values for 50000 random pixel positions in the same tile as in Figure 2.1. The unscaled depth values (measured from sea-level) are mostly clustered between 0 to 75 feet (Figure 2.2a) and with the LPU (Figure 2.2b), the values spread out with a degree of visible discrimination between roads (red color in the plot) and non-roads (blue).

2.2 Related Work

Road network detection in remotely sensed imagery (both pixels and voxels) is not a new problem [12, 13, 14, 15, 16]. Solutions typically involve heuristics or hand engineered features based on color, contrast, texture, parallel edges, flatness and others that capture human understanding of roads. SIFT, Haar, Hog, Zernike moments and co-occurrence matrices are some of the well-known image features that capture the visual perception. Depth information from Lidar gives a more accurate formulation of local road geometry with features like surface normals, curvatures and point feature histograms [17]. It also allows discrimination based on height.

Many of the methods for road detection are unsupervised [12, 13, 18, 19, 20]. Bajcsy et. al. [12] use visual spectral properties and shapes of roads to define low-level operators for road strip detection, road growing, thinning and intersection finding in low resolution satellite images. Fischler et. al. [13] perform edge detection to get local road segments and later establish connections among them to get the whole road network. Hu et. al. [18]

use depth information to validate candidate road segments obtained after applying Hough transform on high resolution images. Zhao et. al. [19] use depth image to extract ground mask with candidate road regions by filtering out elevated objects. They combine intensity image with the ground mask and label road pixels by using an EM algorithm.

Classical supervised methods can be used for road detection with sufficient annotated training images. In this setup, feature vectors are generated for each pixel in a given neighborhood along with the class label which are fed to a learning algorithm like Support Vector Machine. This comes with a few problems. First, non-uniform road widths force us to heuristically fix the neighborhood size during feature computation. Second, the number of training examples is too large since the images are high-resolution with billions of pixels. To fix this, we could attempt to train within a small geographic region with a manageable number of pixels but it limits the type of roads that can be learned. Another solution could be down sampling. Third, individual pixels are classified independently. Finally, selecting and creating hand engineered features is not trivial.

Mnih et. al. [21] try to avoid above mentioned pitfalls by using neural networks as they can scale to large datasets well. Their neural network takes input patches of size 64×64 pixels and learns to predict if smaller 16×16 patches at their centers are roads. It automatically learns relevant features from the interdependence among all the pixels in the bigger 64×64 contexts. During training and prediction, individual patches are processed without considering the neighboring patches. As a post-processing step, they employ another neural network to exploit the structure present in the neighborhood of predicted patches affirming that having a larger context is better for detecting roads.

The recent fully convolutional architectures for semantic segmentation, for example, FCN [22], SegNet [11], U-Net [23] and DeepLab [24], built in the encoder-decoder framework [25] allow even larger contexts. Instead of convolving throughout the image to extract input context patches and classifying their center pixels independently, these deep CNNs first encode the input image into coarse output maps which are then upsampled to dense pixels in the decoder stage. This gives an end-to-end, pixels-to-pixels, low memory system

which learns to map complete input images to their corresponding complete ground truth masks. These are general object segmentation architectures.

For road specific segmentation, most of the state-of-the-art solutions apply deep networks but use only aerial images. Mattyus et. al [2] develop a similar encoder-decoder network called DeepRoadMapper with a Residual Network [26] block and fully convolutional layers. They first segment out roads from satellite images to capture majority of the road links and then build a OpenStreetMap-like road network map from them. Cheng et. al [27] use two deep CNNs in conjunction to extract both the roads and their centerlines. Costea et. al [3] extract roads and junctions from aerial images and match with existing maps from OpenStreetMap to geolocalize the input image in GPS denied environments. Our fusion architecture can be built from any of the existing network designs and only depends on the availability of resources. For our purposes, we use SegNet-like architecture because it has a memory efficient upsampling scheme which gives flexibility in increasing the training batch size. In addition, SegNet uses batch normalization [28] in its layers.

The next challenge is to combine both high resolution aerial images and airborne Lidar to create spatially meaningful image-like inputs to CNNs. Since the RGB image pixels represent the color space and the depth values come from three dimensional (3D) Lidar readings, we observe differences in their modalities. In Figures 2.1 and 2.4, we show the probability density of the different feature channels for an example tile in our dataset (Section 2.5). The distributions are disparate. So, mixing Lidar with RGB in this bimodal setting is not trivial [29]. Moreover, Lidar scans are coarser than images and every pixel location may not have its corresponding Lidar point. This requires transforming Lidar depth into depth image such that the values assigned to every pixel preserve the 3D geometry and help to discriminate roads from non-roads. For detecting and segmenting objects from on-ground viewpoint in 3D scans, [30, 31, 22] use HHA encoding of depth given by [32]. The HHA channels per pixel are, namely, horizontal disparity, height from ground and the angle between local surface normal and direction of gravity. In our case, the viewpoint of the bimodal data is aerial. Since the geometry of roads is mostly defined by flatness in that viewpoint, we propose a simple depth encoding scheme which emphasizes more on the lower values of relative height.

2.3 Lidar Processing Unit

Algorithm 1 LPU

function LPUPIXEL(D, i, j) ▷

\# Where:

\# D is the set of Lidar points

\# (i, j) is the pixel position being computed

\# Pre-computed Constants:

\# $\alpha-$Scaling factor

\# $\beta-$Percentage for robust min computation

\# $r-$ Neighborhood radius size

\# Assumptions about function calls:

\# $pos(i, j)$ is the geographic location of the pixel

\# $Sort_z(X)$ sorts points in X based on z-coordinate

\# $\mu_z(X)$ returns mean of z-coordinates of points in X

\# $d(x, y)$ is the geographic distance between two positions on the globe

$S = Sort_z(\forall p \in D \mid d((p_x, p_y), pos(i, j)) \leq r)$

$\eta = \alpha\left(\mu_z(S) - \mu_z(S[(1-\beta)|S|, |S|])\right)$

return $\max(\lfloor \eta \rfloor, 255)$

end function

One of the commonalities of the architectures we worked on, is our Lidar Processing Unit (LPU). This block processes the Lidar data to a single image that is then fed into various architectures as if it was just another channel like RGB. This is an alternative to computing a lot of extra geometric features [33] or creating a more complex architecture [29] which is harder to train.

The input to our LPU is the Lidar point cloud, and the extents of an image. The output is a single channel image which contains a 8-bit depth feature per pixel. The LPU is parameterized by three scalars - α, β, and r. The algorithm that computes the output per pixel is presented in Algorithm 1. Intuitively, the LPU tries to capture how flat the Lidar point cloud is around a given position by computing the average height of the Lidar points in the neighborhood and subtracting the minimum height. Instead of computing just the

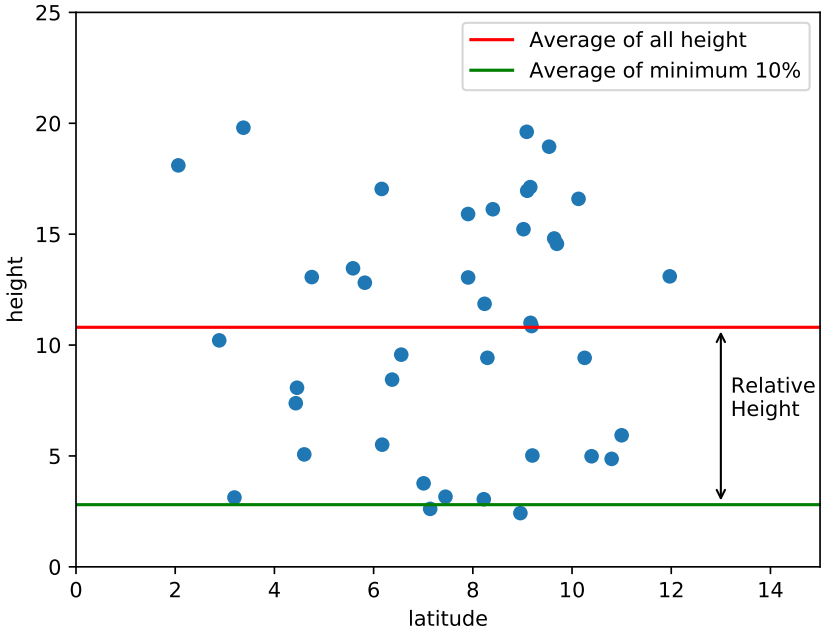
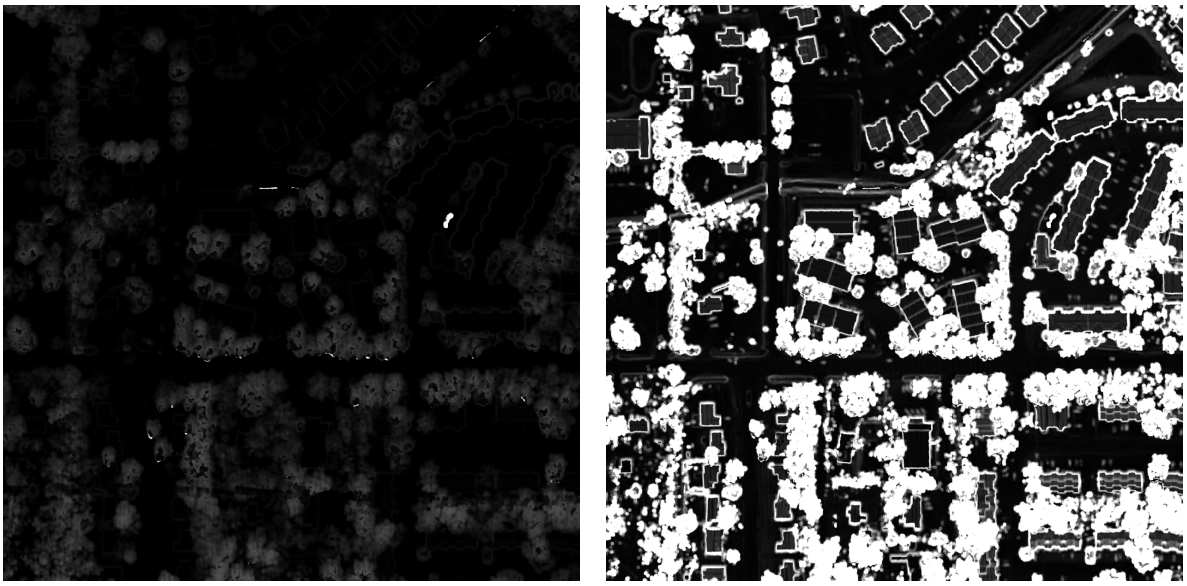


Figure 2.3: Relative height computation.

minimum in z-coordinates, it computes a robust minimum by first sorting the points in z and then taking the average of β-percent of the points with lowest z. Another optimization we added to the LPU is that we only consider the last return. Earlier returns are considered non-ground and are discarded (not part of D). The total number of Lidar points in our input, or $|D| = 1,257,476,297$.

Given that the Lidar point clouds are almost uniformly distributed, our LPU algorithm runs in $O(n)$ time by using a grid based nearest neighbor search. To optimize the values of α, β, and r, we do a grid search on these values using a single depth channel as input to SegNet with a total of 300 training images with roads in them and a set of 100 testing images. This optimization led us to fix these values to $\alpha = 32, \beta = 0.1$, and $r = 4$ feet. An example of the output of LPU with these parameters is shown in Figures 2.8b, 2.4b.



(a) *Unscaled height values from Lidar* (b) *LPU output*

Figure 2.4: LPU transformation of image containing unscaled height values (Figure 2.4a) to an enhanced depth image (Figure 2.4b).

2.4 Fusion Architectures

All the architectures we experimented with are extensions or generalizations of the SegNet encoder-decoder architecture. Our main goal was to optimize IoU metric (Intersection over Union). For all the architectures, we modify SegNet [11] such that it accepts input tensors with both width and height in powers of 2 and does a two class semantic segmentation. Below we list 5 different types of architectures reported in the experimental section.

2.4.1 SegNet

SegNet [11] is a fully convolutional, encoder-decoder architecture derived from VGG-16 [34]. The encoder consists of the first 13 convolutional layers of VGG-16 and excludes the terminal fully connected layers. Each convolution layer is followed by a batch normalization layer and then a non-linear transform layer forming a three layered convolution unit. So, in a convolution unit, the feature maps output by every convolutional layer undergo batch normalization (BN) [28] which reduces variation in input distribution to the next layer and

is known to speed up training of deep neural networks. Santurkar et al. [35] show that BN impacts training by smoothing the optimization landscape. The batch normalized output goes through an element-wise non-linear transform with ReLU (rectified non-linearity). SegNet organizes the 13 convolutional units into 5 different encoder units. The first two contain two units whereas the last three have 3. The last layer of each encoder unit is a pooling layer with a 2×2 window and a stride $= 2$ which selects only the maximum value within the window. This max pooling reduces the size of the input by half along both the dimensions. During max-pooling, SegNet also keeps track of the indices of the pooled maximum values. Since there are 5 encoder units, the SegNet encoder reduces the original input resolution a factor of 2^5. The advantage of using a sequence of downsampling encoder units is that since the convolution window (3×3) remains constant throughout the network, the convolutional layers in the deeper parts of the network attain a wider context.

The decoder of SegNet is a mirror image of its encoder such that each decoder unit upsamples the input by a factor of 2 and, in the process, utilizes the max pooled indices from the respective encoder. Eventually, the last decoder unit produces a tensor with the same resolution (width and height) as the original input but with k channels where k is the number of object classes that we expect to segment. Individual, per-pixel k-dimensional vectors of this final feature map go through a soft-max function which normalizes each vector into a probability distribution of k probabilities corresponding to k classes. A pixel location is given the class label with the maximum probability. Thus, given an input image, SegNet classifies all of its pixels at one go.

We directly use SegNet to fuse the feature channels. We stack the three channels from RGB image with the single channel from the depth image to create the four-channeled input. This is a form of early fusion where the initial kernels in the first convolutional layer perform pixel-wise dot product operations on the four numbers from the feature channels to give single numbers. Although we use the default configuration of SegNet for the road extraction problem, we do modify the loss function for this architecture. Apart from the default Softmax, we experiment with Dice, IoU and XOR.

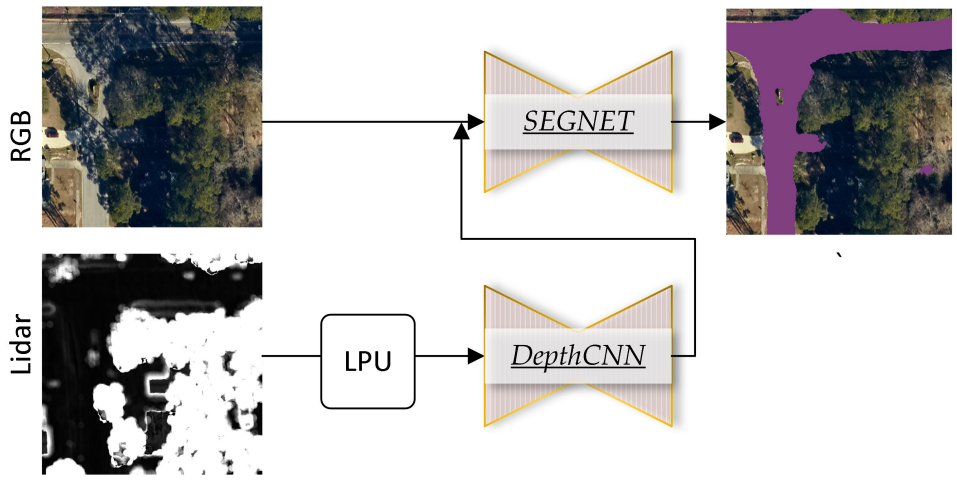


Figure 2.5: *PreConv* architecture for processing Lidar and RGB simultaneously.

2.4.2 PreConv

In this architecture, we convolve the depth channel to create a 4-channel image. We concatenate this image with the RGB image and pass into SegNet. See Figure 2.5. We use Tree of Parzen Estimators algorithm [36] to build the depth convolution (DepthCNN) unit. It has two convolutional layers (8 and 4 filters respectively) with 3×3 kernels and 1 padding.

2.4.3 ElePreConv

This is very similar to the PreConv architecture. In this architecture, we first append a zero-channel image to RGB. This 4-channel image is then added elementwise to the output of DepthCNN unit in PreConv architecture. This can be thought of as one step of FuseNet [29].

2.4.4 RFNorm

In this architecture, we first train cheap Random Forest classifiers for individual feature channels without considering any neighborhood. We perform hyperparameter optimization to fix the depth and the number of trees in the Random Forest for each channel. Random forests are fast to learn and can be included in the preprocessing stage. These classifiers output per-channel and per-pixel probability scores, thereby, transforming any feature into

a value between 0 and 1. We then concatenate these normalized feature channels and feed into the segmentation network.

2.4.5 TriSeg

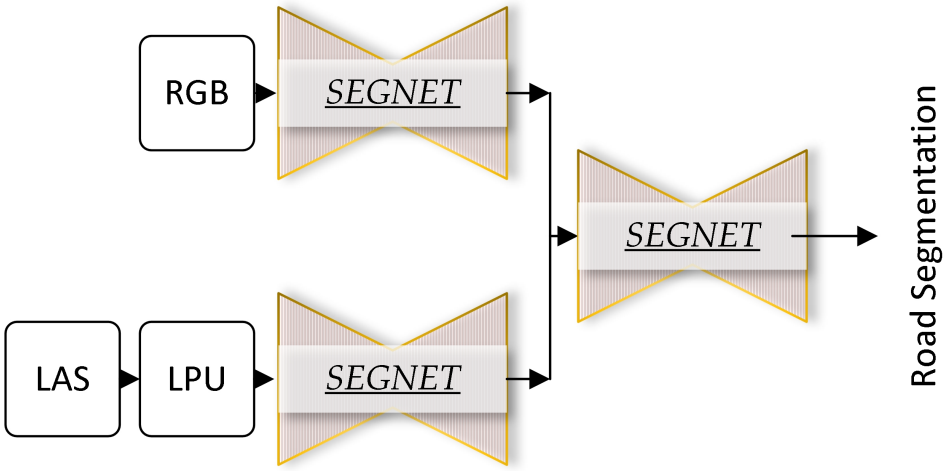


Figure 2.6: TriSeg architecture for processing Lidar and RGB simultaneously.

This architecture is a union of 3 separately optimized SegNets. SegNet 1 is optimized for RGB and outputs a softmax layer with probabilities of a pixel being road. SegNet 2 takes the depth image as input and has the same output as SegNet 1. SegNet 3 takes two channel input with probabilities from SegNet 1 and 2, and outputs the classification. All three SegNets are independently optimized. See Figure 2.6.

We chose TriSeg instead of two SegNets and an FC layer because of memory constraints, ease of training and tuning. We would like to point out a few advantages of the TriSeg architecture: (1) We could not fit two SegNets and an FC layer in our GTX 1080 card with 11GB memory. (2) TriSeg is easy to parallelize (we trained the two SegNets on different machines independently) and easy to optimize. Both PreConv and ElePreConv needed hyperparameter optimization for yielding decent classification accuracies. (3) It is more robust to hyperparameter optimization (for the results, we did not tweak the parameters at all for this architecture). (4) It is easy to implement, given a working SegNet implementation.

We hope that others who might have similar datasets of large size, might benefit from trying out this approach.

2.5 Dataset

We use TLCGIS's [37] high resolution GeoTIFF images and LAS files to build our dataset. The dataset comes from the Tallahassee-Leon Geographic Information Systems Department, which is a joint inter-local agency formed in 1990 between the Leon County Board of County Commissioners, City of Tallahassee, and the Leon County Property Appraiser's Office. TLCGIS acquires Lidar data every 2 years and maintains a very high-quality segmentation of the entire county, which is generated and verified manually. Lidar provides them with the ease of obtaining accurate land data that is of a much higher quality than what has been produced in the past with traditional technology. The complete data has 876 large tiles of size 10K $\times$ 10K out of which we select 63 tiles (See Figure 2.7). For our experiment, we divide these large size tiles to 500 $\times$ 500 resolution tiles. This dataset is challenging since roads have different directions, can be anywhere within the training frame, and can cover anywhere between 0 to greater than 50% of the training frame (Table 2.1). In contrast, the CamVid dataset used in SegNet, the road is always in front of the vehicle. Tallahassee is also rich in vegetation and many of the road segments lie under dense canopy which adds challenge to the road detection problem from the aerial view point. The density and size of our Lidar data is also higher.

2.6 Experiments

2.6.1 Set Up

We conduct multiple experiments with different network architectures with varying feature sets and loss functions. Each experiment uses a single 11 GB GeForce GTX 1080 card on an Intel i7-based workstation with 32 GB RAM. Our implementation was done using Python and Caffe [38]. Given a standard SegNet, it takes, on average, 0.225 second per 512 $\times$ 512

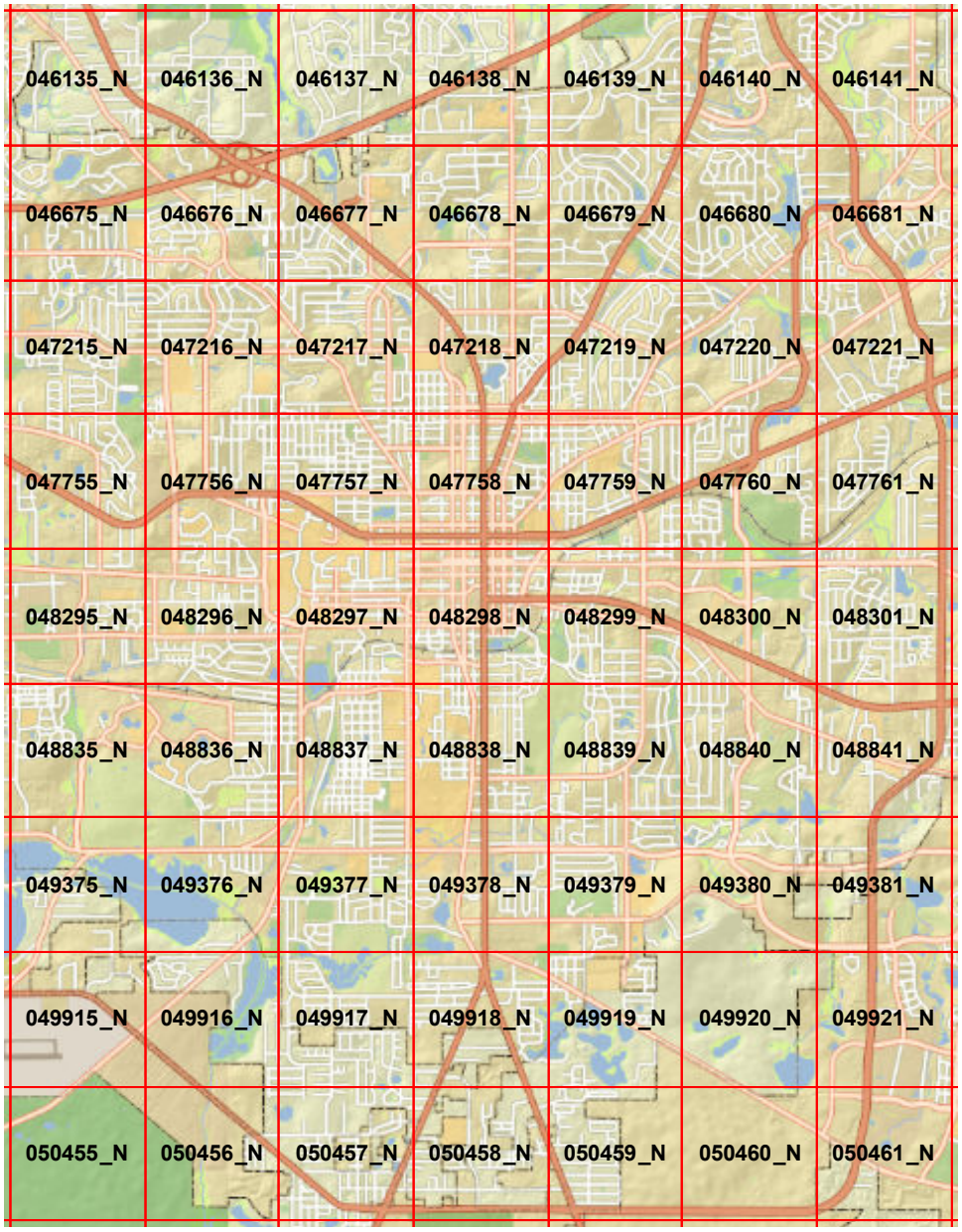
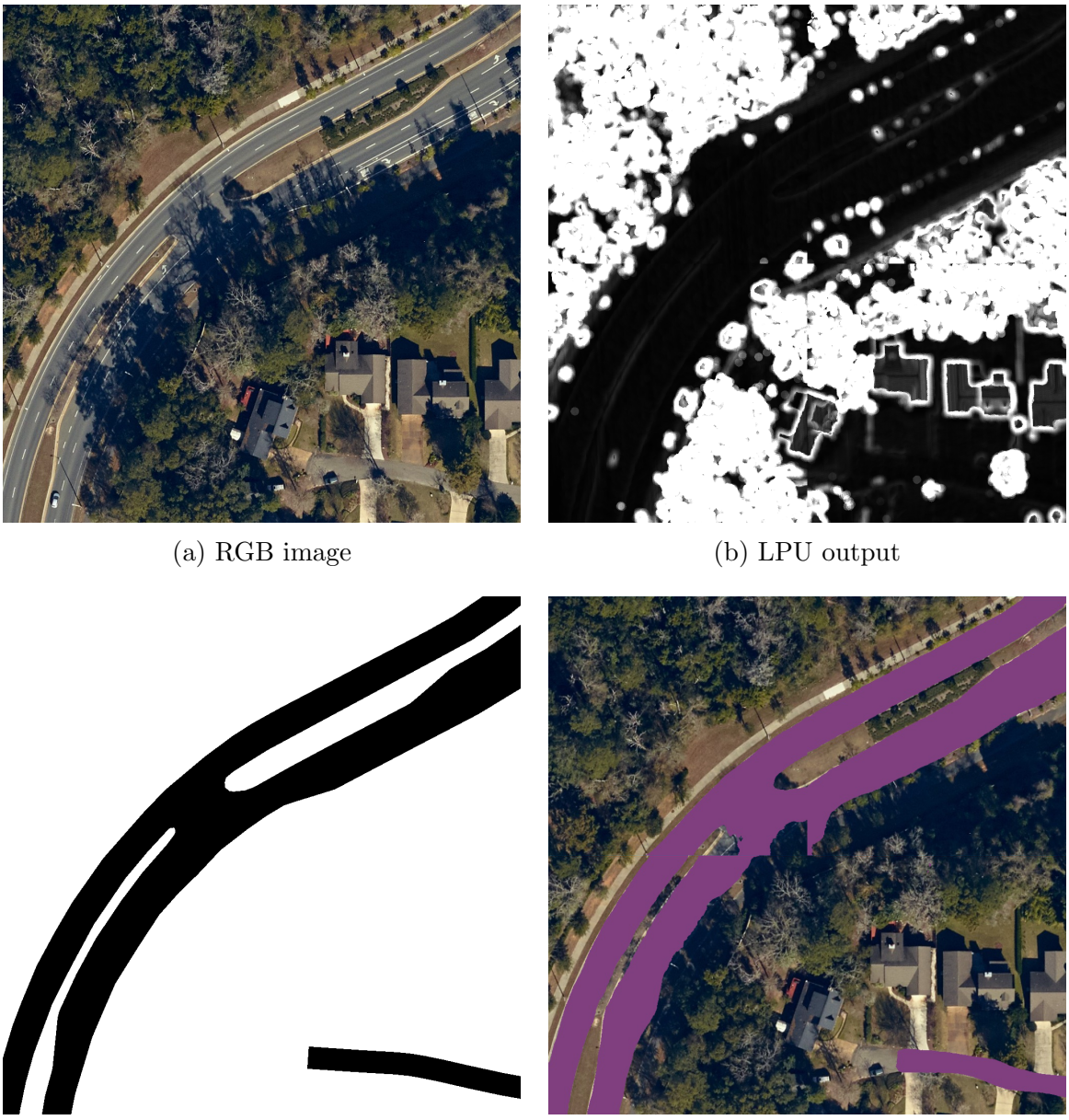


Figure 2.7: Tiles of Tallahassee for which we have LAS and GeoTIFF data. Each tile is of size 5000 feet $\times$ 5000 feet and has a resolution of 10000 $\times$ 10000. The total number of Lidar points associated with this data is upwards of 1.2 billion.



(a) RGB image

(b) LPU output

(c) Ground Truth Road Segmentation

(d) TriSeg output

Figure 2.8: Example of TriSeg road segmentation

Table 2.1: Number of tiles selected for train and test from the respective bins based on the percentage of road pixels

Road percentage	Total tiles	Train/Test Random Sample Size
0	11724	1K
1-5	1570	1K
5-10	3863	1K
10-20	5135	1K
20-30	1372	1K
30-40	547	200
40-50	217	200
50-	74	60

input RGB image to complete a forward and a backward pass through the network. Thus, we downsample the input to make the experimentation more feasible.

Dataset Subsampling. To sample geographic regions for road detection, we first bin all the available aerial image tiles based on road percentage (see Table 2.1) by using the accurate road masks that come with the dataset. The number of tiles picked at random from each bin is listed in Table 2.1. Our random sample consists of a total of 5640 tiles, which includes tiles with no roads in it. As tiles with larger road percentages are rare, we randomly select a bigger percentage of tiles from those bins. We divide the selected tiles into training, test and validation sets with each having 2640, 2400 and 240 tiles respectively.

Training. We use the same training set to learn a different model for each variant of the architecture. Since SegNet is the core of each network, we use its default hyperparameters in most cases. We fix the training mini-batch size to 4 due to the overall size of the network for 512×512 input and the GPU memory limit. We mostly use cross-entropy (Softmax loss) as the loss function. As there is a significant imbalance between the road and non-road regions in a tile, we use median frequency balancing while computing the loss [39]. For this, we compute the frequencies of the road and the non-road classes in the training set. The median frequency is the average of the two class frequencies. We then divide the computed

median frequency with the respective class frequencies to get 1.6903 and 0.2934 as weights for road and non-road respectively.

On our dataset, SegNet takes approximately 16 minutes to do 1000 iterations. Other modifications, like Dice (15 minutes) and XOR (15 minutes) loss functions, take similar time. For one tile, our unoptimized Python implementation takes 55 seconds to compute the depth image.

Baselines. We compare our fusion architectures, including TriSeg with RGB only SegNet and the DeepRoadMapper [2] implementation provided by Bastani et. al [7]. DeepRoadMapper's road segmenter is made up of a ResNet block with 55 convolutional layers which encodes the input to 1/8 of the original resolution. It is then followed by 3 fully convolutional layers which eventually upsample the encoded input to the original size with the help of skip connections and feature maps from the encoder. It optimizes a soft IoU loss where the softmax layer outputs are directly used to compute the loss instead of the indicator functions. We train this baseline for 50000 iterations with a batch size of 4. We use the default Adam optimizer and the learning rates set by [7]. We also learn models for individual feature channels to evaluate improvements gained after fusion.

Evaluation Metrics. To find the best snapshot and to measure the performance of a model, we use true positive rate (TPR), true negative rate (TNR) and intersection over union (IOU) as the evaluation metrics. Here, we exclude overall accuracy because of the class imbalance. All of the metrics are derived from the confusion matrix. TPR (TNR) indicates what fraction of the road (non-road) pixels were correctly classified as roads (non-roads). IOU is an even more stringent metric to evaluate accuracy of a segmentation. If TP, FP and FN are the counts for true positive, false positive and false negative respectively, then TP is the intersection and the sum of TP, FP and FN is the union. IOU is the ratio TP/(TP + FP + FN). IOU takes into account the higher values of TPR when the road classification encroaches the background. For each metric, We further compute its per tile average measure (A-TPR, A-TNR, A-IOU) and pixel-wise global measure (G-TPR, G-TNR, G-IOU).

Table 2.2: Comparison of TriSeg with baselines on per tile average metrics. **A-TPR**, **A-TNR** and **A-IoU** denote the average true positive rate, true negative rate and intersection over union metric per test image. **Iter** denotes the training iteration at which the snapshot of the model was taken.

Architecture	**A-TPR**	**A-TNR**	**A-IoU**	**Iter**
TriSeg	0.796	0.982	**0.746**	33K
SegNet (RGB,Depth)	0.787	0.981	0.732	41K
SegNet (RGB)	0.770	0.982	0.714	38K
DeepRoadMapper	0.748	0.985	0.692	50K
ElePreConv	0.801	0.976	0.710	31K
RFNorm (RGB,Depth)	0.761	0.978	0.685	36K
SegNet (HSV)	0.715	0.980	0.665	50K
SegNet (Blue)	0.680	0.983	0.655	43K
PreConv	0.756	0.973	0.655	28K
SegNet (Green)	0.639	0.981	0.614	35K
SegNet (Red)	0.624	0.984	0.617	43K

We train each model for 50K iterations at maximum and save the model snapshots at every 1K iteration. We select the snapshot which gives the best G-IOU on the validation set and use it for testing. We discuss the results below.

2.6.2 Results

We report our findings in Tables 2.2, 2.3 and 2.4. For road segmentation tasks, RGB only SegNet already performs almost as good as or better than most of the listed models. Although in previous work [33], it was claimed that HSV works better than RGB for road segmentation, on our dataset, HSV did not help. Using HSV color space instead of RGB worsened the segmentation. Based on the experiments with only one of the channels of RGB, we see that no individual channel is driving the full performance of RGB SegNet, although the Blue channel does the best among the three. This could be useful information for a system that has only a one channel camera attached. It was not surprising to find out that three weak features, when combined, perform better. Proceeding with this hypothesis, the series of experiments we performed where we combine depth with RGB does not show significant improvements. Depth alone performs poorly with only 0.41 G-IOU while direct

Table 2.3: Comparison of TriSeg with baselines for the full set of test tiles. **G-TPR**, **G-TNR** and **G-IoU** denote the global true positive rate, true negative rate and intersection over union metric.

Architecture	**G-TPR**	**G-TNR**	**G-IoU**	**Iter**
TriSeg	0.877	0.983	**0.784**	33K
SegNet (RGB,Depth)	0.869	0.983	0.778	41K
SegNet (RGB)	0.860	0.984	0.773	38K
DeepRoadMapper	0.839	0.986	0.763	50K
ElePreConv	0.881	0.979	0.765	31K
RFNorm (RGB,Depth)	0.857	0.980	0.752	36K
SegNet (HSV)	0.823	0.982	0.730	50K
SegNet (Blue)	0.801	0.985	0.722	43K
PreConv	0.840	0.975	0.713	28K
SegNet (Green)	0.773	0.983	0.690	35K
SegNet (Red)	0.761	0.985	0.689	43K

Table 2.4: Effect of feature normalization with Random Forests

Input Channels for SegNet	G-TPR	G-TNR	G-IoU	Iter
Blue	0.801	0.985	0.722	43K
Green	0.773	0.983	0.690	35K
Red	0.761	0.985	0.689	43K
RFNorm(Blue)	0.791	0.976	0.677	38K
RFNorm(Red)	0.729	0.984	0.657	35K
Depth	0.602	0.933	0.412	18K
RFNorm(Depth)	0.718	0.97	0.593	24K
RFNorm(RGB)+RFNorm(Depth)	0.857	0.980	0.752	36K
RGB + RFNorm(Depth)	0.868	0.981	0.768	40K

concatenation of RGB and depth nudges the G-TPR by only 0.5%. We apply **PreConv** and **ElePreConv** to learn transformations that reduce differences in modalities between RGB and Depth. We achieve better G-TPR with **ElePreConv** but worse G-IOU indicating overflow of predicted roads. To further improve the mixing of depth and RGB, we do late fusion with **TriSeg**. It beats RGB only SegNet and the state of the art (DeepRoadMapper) by 1.3% and 2.1% respectively in G-IOU. On visual inspection of the output segmentations, we observe portions of parking lots and flat surfaces with shadows being classified as roads accounting for the spill over. There are also roads under dense canopies which the models miss. The models also classify centers of large, flat, asphalt rooftops as roads. An example **TriSeg** output is given in Figure 2.8.

In the other experiments, we replaced the feature channels with Random Forest classifier probability scores based on the RFNorm architecture. This normalization was not effective when we tried to directly concatenate the input channels for fusion (see Table 2.4). The global IoU metric worsened with this approach for Random Forest normalized RGB and Depth channels. In case of individual channels, RFNorm degraded the metric for Blue channel by 4.5% whereas it improved for Depth channel by almost 18%.

We also experimented with multiple loss functions, and their combinations, including Dice, IoU, XOR, and Softmax. IoU, as suggested in [40], did not converge on our dataset in $50K$ iterations. Dice seemed easier to converge than XOR and beat XOR in the first $50K$ iterations. Both Dice and XOR were overfitting, and hence we had to optimize the weight decay parameter using the L_2 norm. Dice, with 78% global IoU, came close to **TriSeg** but could not beat it.

2.7 Conclusion

In this paper we present a simple and deep convolutional network architecture to process aerial imagery with Lidar data. The main motivation behind this work is the segmentation of road networks by considering both RGB and Lidar data for aerial images. We explored many different variants of CNNs for this application. We also use a new and large dataset from our city, which has high resolution imagery as well as high quality Lidar for training.

Our work is based on SegNet and we compare various approaches for processing this bimodal dataset. A natural extension of this work is to do aerial segmentation for bimodal data with larger number of types, such as buildings, water, forest, pavements, parking lots and rail tracks. We hope to make progress on this problem in the near future.

CHAPTER 3

RECONSTRUCTING ROAD NETWORK GRAPHS FROM BOTH AERIAL LIDAR AND IMAGES

3.1 Introduction

Due to the current trend towards offering mobile location-based services, the task of accurately and efficiently detecting road networks carries immense importance [41]. Applications like autonomous navigation and routing [4], self-driving [42], localization [3] and others use road networks in the form of graphs that are embedded in a three (or two) dimensional space where the road centerlines form the edges and each node is defined by a tuple of three floating point numbers, namely, latitude, longitude and height. These road network graphs are mostly created by volunteers with manual labeling [5] which is a slow and expensive process and is prone to errors. For example, one of the most widely used open source mapping systems, OpenStreetMap (OSM) is currently in need of improved correctness. Bastani et. al [7] report a discrepancy of 14% between OSM and the ground truth in TorontoCity Dataset. Thus, we consider the problem of automatically extracting accurate road network graphs for large geographic regions.

In recent days, a variety of datasets are currently being used for extracting road networks. One of the common datasets comprises GPS traces due to the proliferation of portable navigation devices on cell phones and vehicles [43, 44]. The collection and use of personal GPS traces raises privacy concerns [45] and may also result in discrepancy in coverage density because some roads in remote and rural areas will have negligible traffic compared to the roads in and around urban cores. Overall, to mitigate these shortcomings, we need a dataset which does not encroach upon users' personal privacy, provides wider road coverage and is also cost effective to collect. Based on the current state of remote sensing for data acquisition,

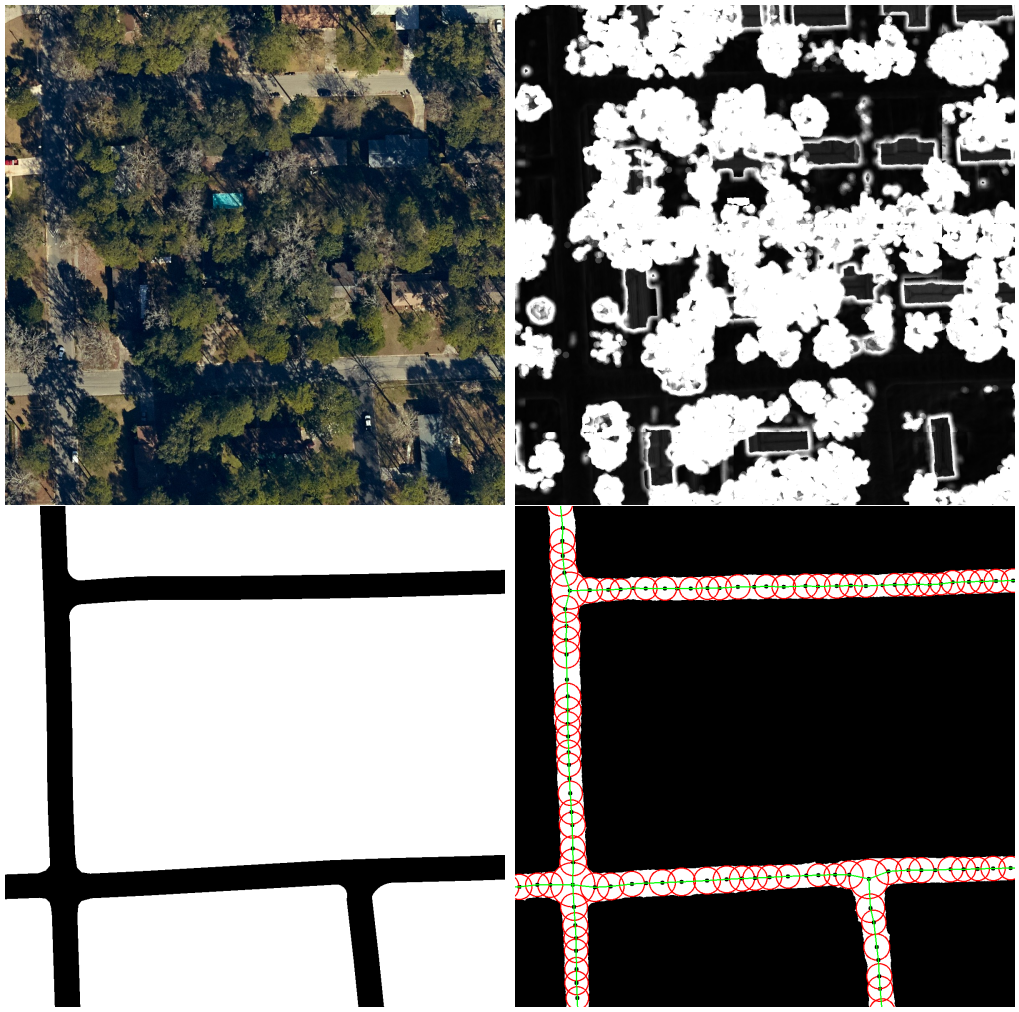


Figure 3.1: Given the aerial RGB (top-left) and the depth (top-right) images, we segment out roads and apply disk-packing (bottom-right) to get the graph. The red circles, their centers and the green links are the packed disks, the graph nodes and the graph edges respectively. The bottom-left image is the road ground truth.

collection of very high resolution multi-channeled aerial images defined by pixels along with air-borne Lidar defined by 3-dimensional point clouds over vast swathes of geographic regions is getting cheaper [46]. Thus, for road network graph extraction, we consider the dataset of aerial Lidar and RGB images.

In this paper, we present a pipeline which automatically builds road network graphs from publicly available bimodal dataset containing airborne Lidar and images. In the first step we use a deep learning architecture that fuses both Lidar and RGB images to obtain highly accurate road segmentation while reducing the disparity in the dataset modalities. We choose to supplement RGB images with depth information from Lidar because in case of roads that are occluded (for example beneath dense canopy), the last return of Lidar can still penetrate through the gaps among the branches and capture the surface geometry of roads [47]. We also ensure larger neighborhood context for road vs non-road classification by selecting a suitable encoder-decoder architecture that uses dilated convolution [48]. The output given by the trained segmentation model consists of per-pixel road predictions which lie in a two dimensional plane. These predictions are usually rasterized and noisy and appear as disconnected blobs or ribbons of different shapes and sizes. In the next step, we apply a disk-packing based curve reconstruction algorithm that translates the raw road segmentation into planar road network graphs.

Our road network graph extraction algorithm, first, replaces the true positive regions with a series of largest possible discs which are approximately centered along the corresponding centerlines. This also removes most of the noise, false positive regions and outliers. The algorithm, then, connects the centers of the packed discs that intersect to produce a preliminary graph. Finally, the remaining non-intersecting discs are connected with the support of the output road segmentation by doing a directional cone neighborhood search. We run experiments on a real-world dataset and show that our method performs better than the current state of the art in various metrics.

The paper is organized as follows. We discuss related work in the following section. In section 3.3 we formulate the problem and elaborate on the dataset peculiarities and our assumptions. We cover an existing graph extraction method and present our disk-packing

based algorithm in Section 3.6. We define various graph similarity metrics in Section 3.7. We describe the dataset and the experiments that we conducted along with their results in Section 3.8. We conclude along with our future plans in Section 3.9.

3.2 Related Work

Many road network graph extraction methods are related to curve reconstruction [49, 50] where the goal is to retrieve the road centerlines as connected line segments from a point set sampled from the original road surface. GPS traces provide such a sample [51, 52, 44]. Any method that uses this dataset can take advantage of the chronological ordering of points in the trajectories if it chooses to. Chen et. al [51] sample a set of points S with a $(b \cdot r_{min})$-net such that all points in the input trajectories are within $b \cdot r_{min}$ of at least one point in S and all points in S are at least $b \cdot r_{min}$ apart. $b = 2\sqrt{6}$ is a constant chosen for algorithmic guarantees and r_{min} is a constant for the road width. In reality, road widths vary [21]. They construct a Voronoi diagram with $p \in S$ as the sites and select the Delaunay edges which correspond to the Voronoi edges that intersect at least one input GPS trace. This gives a structure graph with chains of connected Voronoi cells. They select portions of input trajectories that lie within each Voronoi cell to create the final road network graph. In our case, we do not have the trajectory information because during road segmentation the pixels are classified independently. We also do not have the privilege of using principled sampling since we have to rely on the sampling provided to us by the trained road segmentation model.

In cases where aerial imagery and Lidar are used as the dataset, there are a few instances where researchers tweak and extend some parts of their road detection method to correct the output predictions [12, 21, 20]. Bajcsy et al. [12] use road-growing and thinning operators while detecting the road network which also simultaneously skeletonize the predictions to give a graph as output. Most often the corrections are done independently as a post processing step. Zhao et al. [20] use an adaptive, multi-step marching algorithm with voting to replace the detected road ribbons with centerlines. They infer the missing connections by modeling the type of intersection among the candidate road segments as a graph and determining the segment labels with an MRF framework.

One of the more general tasks for graph generation is replacing two dimensional ribbons or blobs with their centerlines or skeletons. Zhang et al. [53] extract the skeleton of the object in an image by iteratively removing only the contour pixels while preserving the connectivity in the object. Voronoi diagrams can also be used to calculate the medial axis of a polygon [54]. Since these skeleton finding methods are sensitive to noise, the predicted road segmentation needs to undergo morphological operations like dilation or erosion to smoothen the kinks and remove the outliers. In case the prediction blobs are disconnected, the resulting skeleton requires some kind of post-processing to maintain connectivity. To address noise and connectivity issues, Aanjaneya et al. [55] extract branching filamentary structures from noisy datasets by using a spherical shell to infer the degree of points.

Currently, use of deep convolutional neural networks (CNN) for road network detection and graph extraction from aerial images has become common [21, 27, 2, 7]. Mnih et al. [21] use a neural network to obtain road predictions and later they employ another neural network to exploit the structure present in the neighborhood of predicted patches to make corrections. Cheng et. al [27] use two CNNs to segment roads and get their centerlines. These methods address connectivity issues in the detected road networks but they do not translate them into graphs.

Road network graphs can be obtained from the road segmentation or centerlines by following a certain procedure: (1) apply morphological thinning [53] to the segmentation to get 1-pixel thick skeleton (2) do a generic breadth-first search along the skeleton to trace paths as piecewise linear curves (3) transform the curves into edges (4) postprocess to either remove dangling segments or connect nearby degree-1 vertices. There are other ways to get the skeleton as well. Instead of thinning, Voronoi diagrams can also be used to calculate the medial axis of the road segmentation polygons [54]. DeepRoadMapper [2], a state of the art, follows these steps to arrive at a preliminary graph. First, it segments out roads with a Residual Network [56] block based deep CNN. which has 55 convolutional layers which encodes the input to 1/8 of the original resolution. It is then followed by 3 fully convolutional layers which eventually upsample the encoded input to the original size with the help of skip connections and feature maps from the encoder. It optimizes a soft IoU

loss where the softmax layer outputs are directly used to compute the loss instead of the indicator functions. It applies thinning [53] on the road predictions to obtain a skeleton with predicted road centerlines and vertices which is then translated to a graph. This graph is prone to topological errors. To make corrections, it samples candidate links within a certain neighborhood of each vertex using heuristics. The bounding box of each such link is cropped from the aerial image and another CNN is used to predict if the link containing patch is a road. If it is, then the link is inserted into the predicted graph. Even after a complex post processing, the method does not perform well at regions where the segmentation is poor due to either occlusions or complexity of the road topology [7].

The more recent state-of-the-art RoadTracer [7] skips the road segmentation phase and directly extracts the road network graph from aerial images in an iterative fashion. It is more suited for warm-start scenarios where preliminary but incomplete road network graph is available. Instead of classifying individual pixels independently, it does a guided search in a depth-first manner with the help of a decision CNN and adds edges and nodes to the output graph iteratively. Based on a fixed size RGB patch centered at the current node and the graph that has been predicted so far, the CNN decides whether to stop or to walk in a certain direction, which is defined by an angle. The length of the walk and the distribution of the angles are pre-determined. There are a few shortcomings to this method as well. Since this method requires starting locations that must lie on the road, for cold-start cases where no pre-existing map is available, we cannot do away with the road segmentation phase. Due to the fixed walk length, the search can land on top of an occlusion, thereby, terminating the search. In case the input image contains disconnected roads, each road fragment should be given its own starting location and the search algorithm should be applied as many times. Furthermore, the training procedure not only needs the ground truth graph but also the intermediate stages of the partially predicted graphs in every iteration. Since the partial graphs are not available, they are engineered dynamically by finding the ground truth path which is most similar to the path predicted so far with a map-matching algorithm.

The existing approaches for road network graph extraction are affected by occlusions and the ones which use thinning to get the preliminary graphs are prone to outliers. We

try to address these weaknesses by using both aerial lidar and images in conjunction to enhance the road segmentation accuracy. We also improve the output graph quality with our disk-packing based curve reconstruction algorithm.

3.3 Problem Definition

Given a feature representation of a geographic terrain as an input image, a trained road segmentation model, M_{seg} labels individual pixels as either road or non-road with pixel values 1 and 0 respectively. M_{seg} produces an image-like segmentation output I, where the pixel positions form a unitary integer grid that can be interpreted as a planar (two-dimensional, Eucledian) coordinate space. All the points in this coordinate space are grouped into two sets, $S_{rd} = \{(i,j) \forall I(i,j) = 1\}$ and $S_{nrd} = \{(i,j) \forall I(i,j) = 0\}$. In essence, M_{seg} samples points that are most likely to be on road surfaces and the global Intersection-Over-Union (IOU) metric gives a measure of its sampling quality.

The core problem this paper addresses is to translate S_{rd} into a reconstructed undirected graph $G = (V, E)$ that is embedded in the Eucledian plane. Points in S_{rd} do not form a well-behaved geometric object and it cannot be described with well-defined equations. Locally, given a neighborhood, the geometry is elongated that can be approximated with a rectangular strip. Globally, S_{rd} can be regarded as a union of such strips. The widths of the strips vary, their perimeters may not be smooth and they can be disconnected owing to occlusions such as trees. Going from S_{rd} to G involves replacing the local, approximate rectangles with their centerlines and adding them to E while maintaining connectivity to construct G.

3.4 Road Segmentation

Road Segmentation accuracy has a direct impact on the road network extraction problem. For this paper, we chose to improve one of the recent road segmentation CNN architectures, named TriSeg, that merges both Lidar and image data to solve the problem [1]. TriSeg consists of 3 separate units. Unit 1 and Unit 2 process RGB and Depth images independently

in parallel and produce their respective outputs as road segmentation softmax probabilities. Unit 3 performs late fusion by concatenating the outputs of the first two units and gives the final road segmentation.

We make two changes to TriSeg. We replace SegNet [11] with Deeplabv3+ [57] since it increases the size of the neighborhood context by using atrous convolution. Instead of simply concatenating the output from the first two units of TriSeg, we average the input RGB image to get a grayscale channel and append as an additional channel to the input of the third TriSeg unit.

We experiment with the dataset released by Parajuli et al [1] and show a few output examples along with their corresponding inputs in Figure 3.2. The architecture does well even at places where the roads are covered by trees. It affirms the benefit of using aerial Lidar in conjunction with RGB images for road segmentation. Table 3.1 shows that we can achieve 5% improvement in the global IoU metric by making the above mentioned changes. We will refer to this new CNN Model as M_{seg} and use it to build our algorithm.

Grid-shifting. Both random and deterministic shifting of grids has been used in the past for various problems [58, 59, 60]. We apply a deterministic grid-shifting technique to extract a small improvement in the road network segmentation accuracy. The technique is applicable to any CNN problem where the input image has to be cut down to smaller sizes because CNN input sizes are restricted to GPU RAM sizes. Below we describe our technique applied to the road segmentation problem.

Our goal is to extract road network graphs for larger $K \times K$ resolution tiles from their corresponding road segmentations. To obtain the road segmentation in the original resolution, we partition the larger tile into smaller sub-tiles of size $k \times k$ where $k \ll K$ and is dependent on the memory constraints of M_{seg}. We feed each $k \times k$ sub-tile into the road segmenter M_{seg} independently. Given a sub-tile, M_{seg} cannot learn from the context present in the neighboring tiles while trying to segment roads along the tile edges. So, if there is an occlusion that covers the road along the sub-tile margin, M_{seg} can assume it to be the end of a road segment due to the absence of context. This creates gaps in the segmentation

Table 3.1: Comparison of TriSeg with M_{seg} on Parajuli et al.'s [1] dataset: G-TPR, G-TNR and G-IoU are the global true positive rate, true negative rate and intersection over union respectively.

Architecture	**G-TPR**	**G-TNR**	**G-IoU**	**Iter**
TriSeg	0.877	0.983	0.784	33K
M_{seg}	0.93	0.984	0.834	50K
M_{seg}	0.929	0.985	**0.837**	100K

output and results in missing links in the final road network graph. We can see a few of such examples in Figure 3.3. As a solution, we perform a second road segmentation but after shifting the original grid by $\frac{k}{2}$ pixels along both the directions. This forces the margins of the original sub-tiles to lie towards the middle of the new sub-tiles and M_{seg} gets more context while detecting roads.

We get the final road segmentation after combining the two segmentation outputs with and without grid-shift. For merging the segmentations, we simply take the union of the two where a pixel location is considered to be a road if at least one of the two segmentation outputs labels it as a road pixel. The Union method gives a global IoU metric of 81.2% whereas for the non-grid-shifted segmentation output, it was 79% for the dataset described in Section 3.8. We also experimented with a centrality based approach where the segmentation is given higher confidence based on how far the pixel is from the center of the image. This method performs worse than the union method (80.6%). Thus, we use the union-merged output as our final road segmentation which we further process to extract the road network graph.

3.5 Thinning Based Approach

In this section we discuss one of the most common methods used to translate road segmentation output into road network graphs. Since the road segmentation algorithm samples points that lie on the roads, if we can replace each of the segmented regions with their respective centerlines, we get a skeleton that approximates the topology of the actual

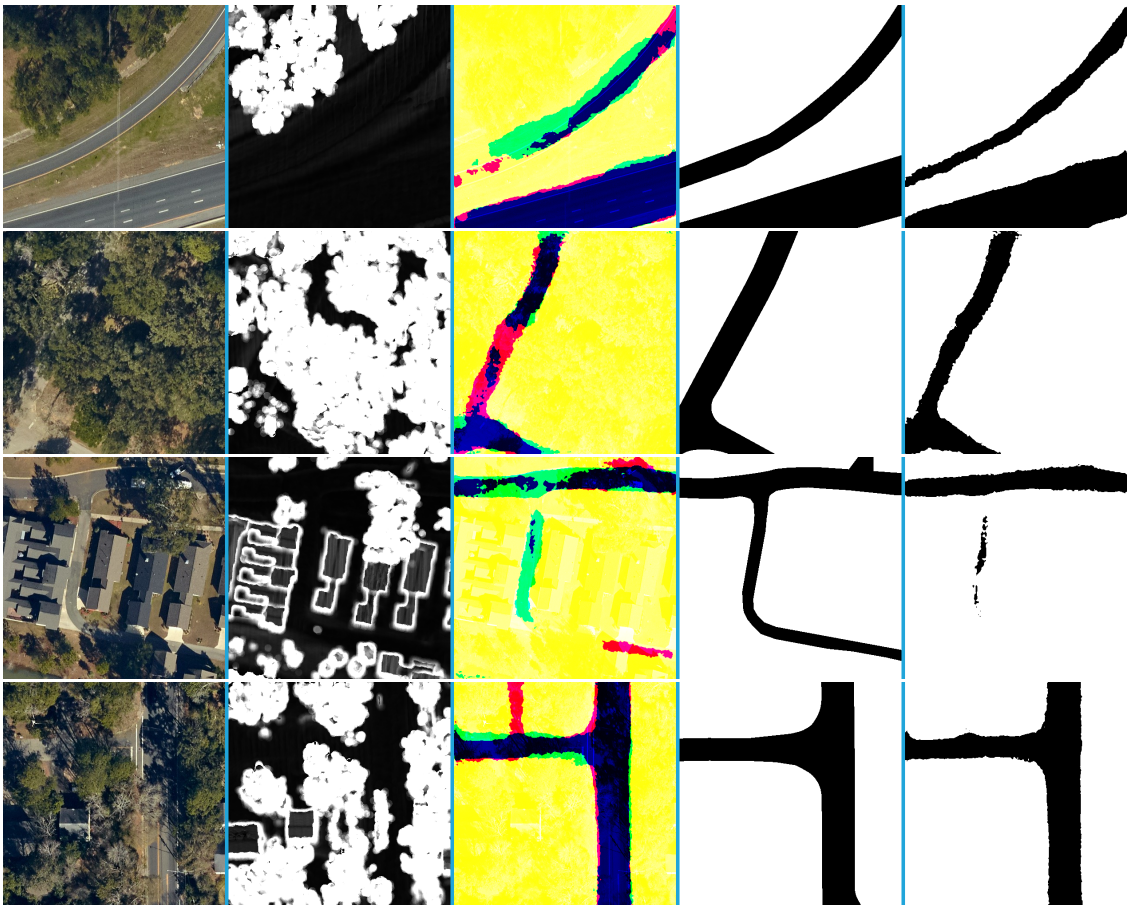


Figure 3.2: Sample output by M_{seg}. Each row consists of 5 images from left to right: RGB, depth, M_{seg} Unit 3 input, ground truth, output segmentation.

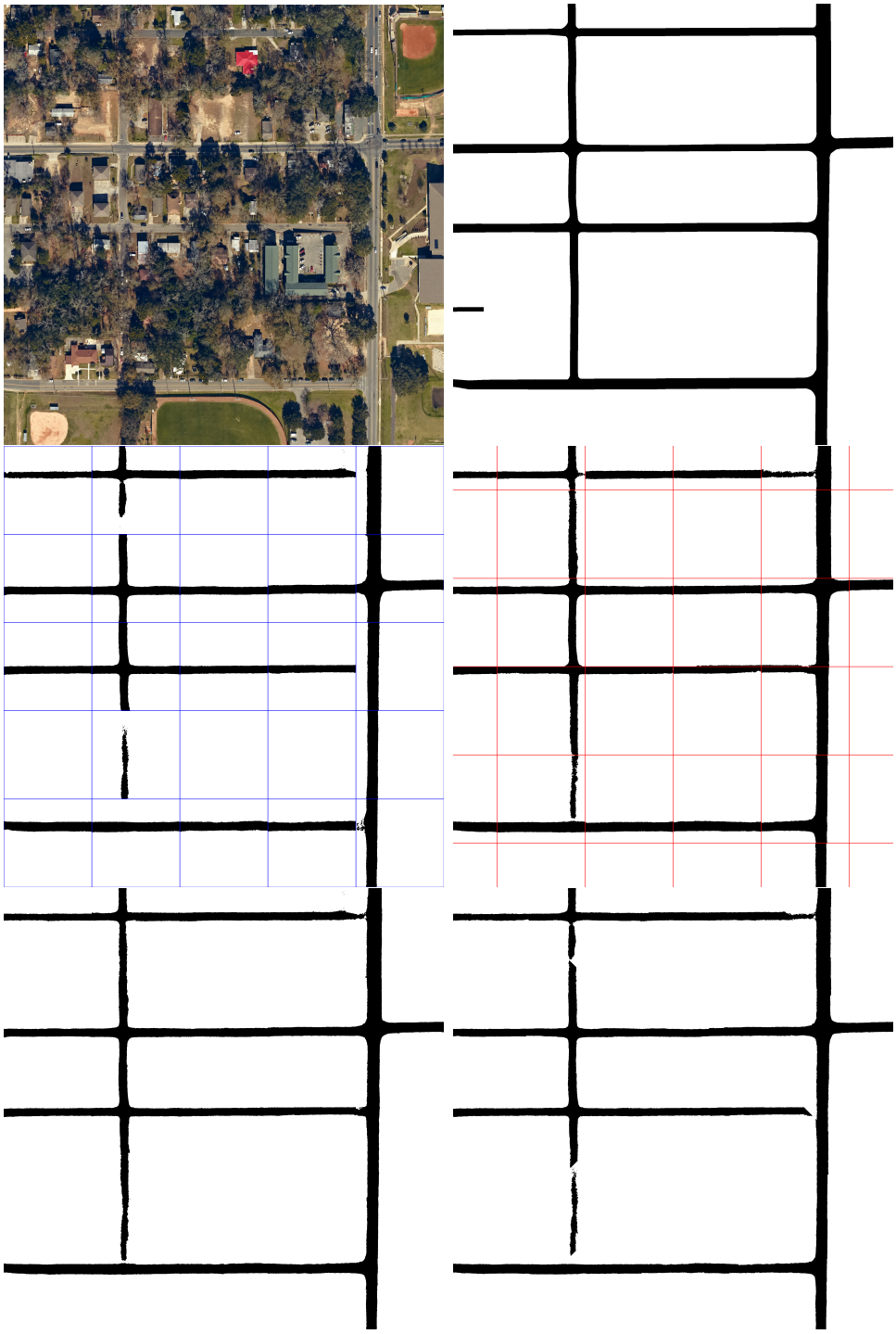


Figure 3.3: Effect of sub-tiling on the segmentation. In the original tiling (mid left, blue grid), we observe gaps in road segmentation along the tile margins when compared to the ground truth (top right). When we shift the grid, some of the gaps are covered (mid right, red grid). For obtaining the final road segmentation, the results of union-merging (bottom left) are better compared to centrality based merging (bottom right).

road network graph. So, the vertices and the edges selected for such a graph should capture the road centerlines while preserving the road topology and connectivity.

3.5.1 Preprocessing

Even before applying any graph extraction method, the holes in the road segmentation should be patched and the rough edges along the segmentation blobs should be smoothened. If not, the topology of the output graph can be drastically altered by the holes and also the outliers. In case of the skeleton based method described below, the resulting skeleton may have unwanted branches and may not follow the required road topology. As a solution, morphological dilation can be applied which updates every pixel with the max value in their neighborhood. Deciding on the optimal size of the dilation neighborhood can be tricky and is often done manually. Delaunay triangulation [61] can also be used to fill up the holes where only the pixels corresponding to the road segmentation are triangulated. Any triangulated simplex whose longest edge length is less than a predefined threshold is filled as a road.

3.5.2 Skeletonization

The skeleton based approach is one of the most common graph extraction method. It proceeds in two main stages: (a) Segmentation thinning to get the skeleton (b) Path tracing along the skeleton to get the graph. The road segmentation is first dilated and then thinning [53] is applied. The thinning algorithm iteratively removes the contour pixels of the road segmentation blobs while preserving connectivity. It produces a 1-pixel thick skeleton which approximates the road centerlines.

The next task is to select vertices and edges from the skeleton and construct a graph. This can be achieved with a generic breadth-first search along the skeleton. The search traces paths as piecewise linear curves between pixel positions with degree not equal to 2. The degree of a pixel is determined by the number of road pixels in its adjacent neighborhood. Each path is simplified with Ramer-Douglas-Peucker algorithm [62]. A collection of vertices along the simplified paths and their edges forms a graph.

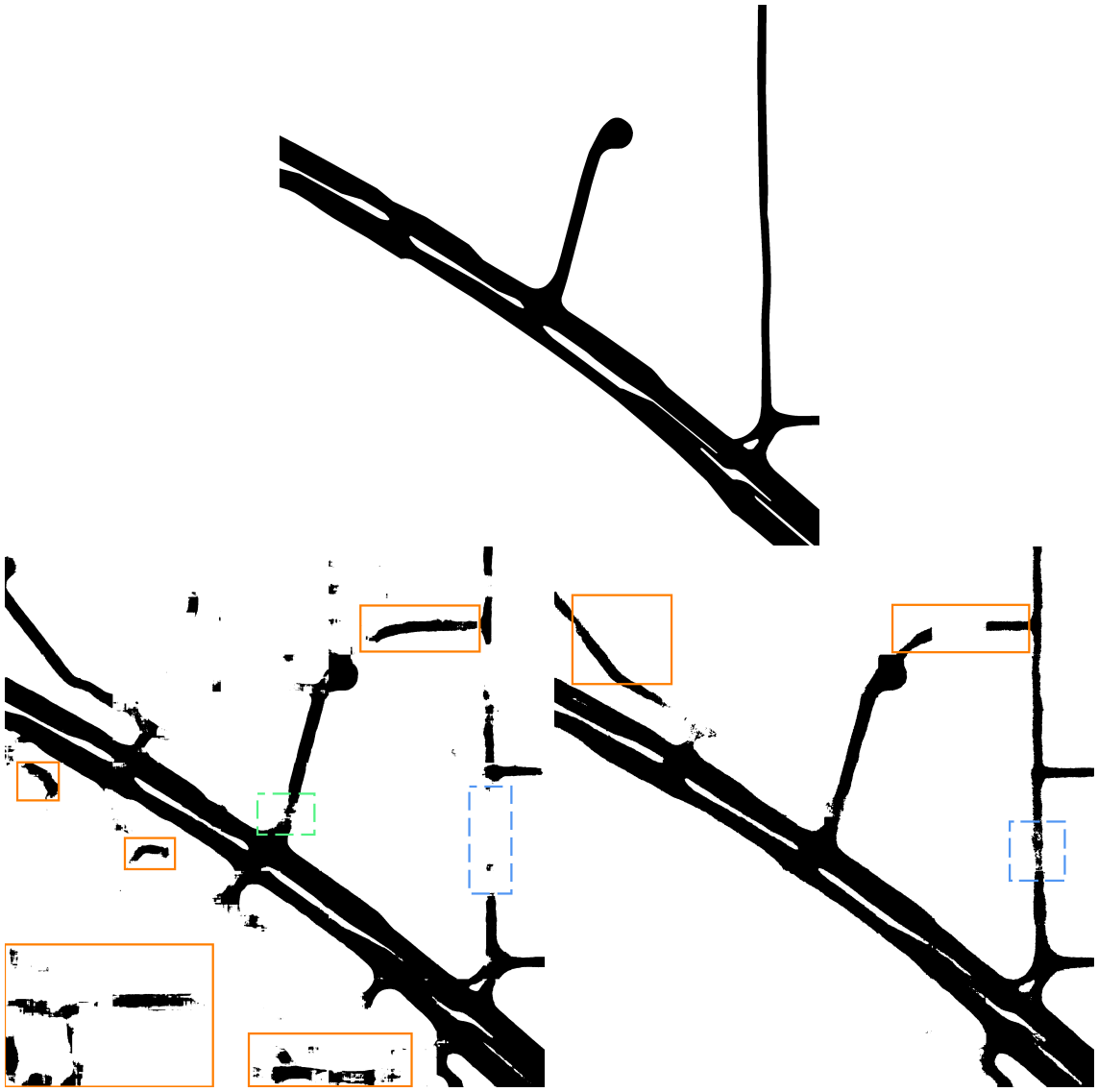


Figure 3.4: Outliers, corrugated edges and holes in road segmentation output. *Top*: ground truth road segmentation. *Bottom left*: Segmentation output by DeepRoadmapper [2]. *Bottom right*: Segmentation output by M_{seg}. Orange rectangles correspond to the outliers. Green box shows corrugated edges. Blue boxes depict holes and missing links.

This is a simple method but it has a few drawbacks. If the thresholds are not chosen optimally, we can get disparate output graphs. For example, a large dilation neighborhood can merge parallel but close road segments. In case of the Ramer-Douglas-Peucker algorithm, a large enough threshold may hamper the topological accuracy of the graph. There is a need for a robust and a more general method that is free from hand engineered thresholds. Thus, we propose a method based on disk packing.

3.6 Disk Packing Based Approach

In this approach, we greedily cover S_{rd} with a minimal set $D = \{D_1, ..., D_n\}$ of n closed disks where D_i, centered at $c_i = (x_i, y_i)$ has a radius $r_{c_i} \geq r_{min}$ and is equidistant to atleast two points in S_{nrd}. If $dist(c_i, c_j)$ is the Euclidian distance between c_i and c_j and $r_{c_i} + r_{c_j} \geq dist(c_i, c_j)$, then D_i and D_j intersect. We create a planar disk intersection graph G_{disk} from a set of such intersecting disks [63]. The disk centers and the lines connecting the centers of the intersecting disks are the vertices and the edges of G_{disk} respectively. We take G_{disk} as a preliminary graph as there are still missing links because M_{seg} is not perfect and gives an IOU metric of < 1. Next, we describe how we select D and resolve connectivity issues in G_{disk}.

3.6.1 Disk Selection

We start with the road segmentation output I which may contain holes, corrugated edges and outliers. These can drastically alter the topology of the output graph. We partially address this issue by morphologically dilating I with a neighborhood of size r_{morph} which updates every pixel in I with the max value in that neighborhood. We apply Canny edge detection algorithm [64] on I, and find the set S_{edge} that contains points along the edges in I. For efficient nearest neighbor search, we construct a KD-tree KDT_{nrd} with points in S_{edge}. For every point $p \in S_{rd}$, we find the nearest $q \in S_{edge}$ at a distance $r_p = dist(p, q)$ with the help of KDT_{nrd} and then insert the tuple (p, r_p) into L_{disk}, a list that keeps track of candidate disks. Here, r_p is the radius of the disk centered at p. To proceed greedily, we

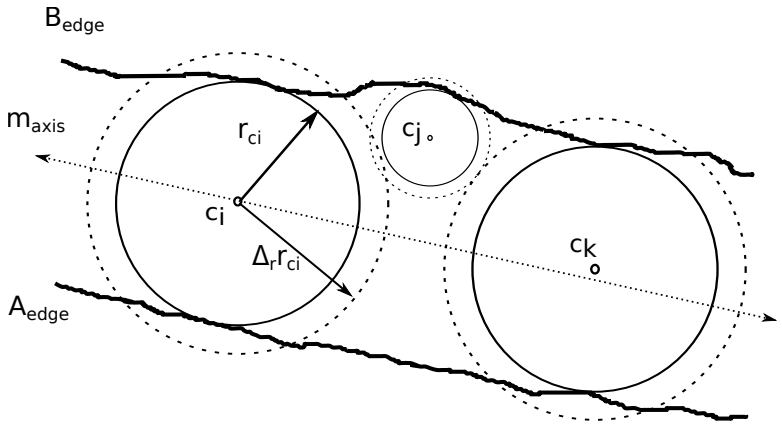


Figure 3.5: Detecting outlier disk: A_{edge} and B_{edge} belong to S_{edge}, the set containing pixels along the edges in the segmentation output I. m_{axis} is the ground truth road centerline. The dotted circles around each disk represent their corresponding shells. The disk centered at c_j is an outlier disk because its shell contains only a portion of A_{edge} which translates to a single connected component. The other two are valid degree two nodes in the graph since their shells contain two connected components that lie almost opposite each other.

iterate through L_{disk} in descending order with respect to the disk radius such that we select the largest disk first. For each p with disk radius r_p, we drop all points and disks that are covered by the disk centered at p with radius r_p. We also drop those disks whose radius is less than r_{min}.

Eventually, we want the selected disks to lie as close as possible to the medial axis [65] of actual roads. If the edges in I were perfectly smooth, each such disk would be equidistant to two (or more in case of junctions) points in S_{edge}. In reality, the edges still contain variably sized concavities. So even if a disk is equidistant to two points in S_{edge}, it can be far away from the road centerline. Figure 3.5 demonstrates an example. We want to automatically drop such disks that predominantly lie only on one of the half spaces created by the closest centerline. Given a disk (c, r_c), we first find all the points in S_{edge} which lie within $\Delta_r \cdot r_c$ of c. Here, $\Delta_r > 1$, is the radius expansion factor. We then count the number of connected components n_{comp} present in the $\Delta_r \cdot r_c$ shell of c. If $n_{comp} < 2$, it is highly likely that (c, r_c) is a noise disk and, thus, it can be dropped. Our method is similar to the shell neighborhood search described by Aanjaneya et al. [55].

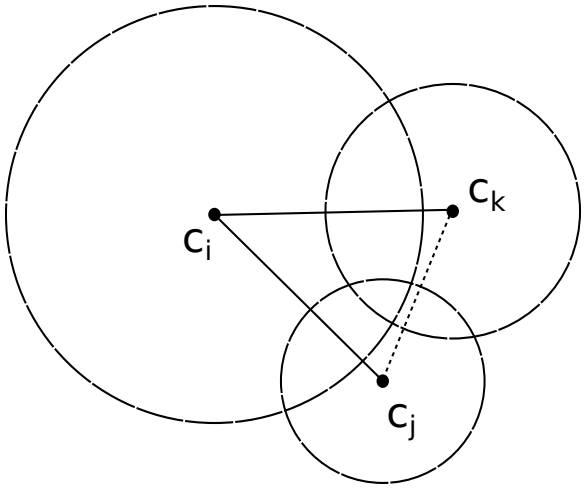


Figure 3.6: Triangles in disk intersection graph. When three disks with centers c_i, c_j, c_k intersect, they create a triangle. If $r_{c_i} > r_{c_j}$ and $r_{c_i} > r_{c_k}$, we remove the shortest edge (c_j, c_k) and keep (c_i, c_k) and (c_i, c_j).

3.6.2 Disk Intersection Graph

The disk-packing method discussed above gives a set $\mathcal{B}$ of n disks. To obtain the disk intersection graph G_{disk}, we do a $k = 10$ nearest neighbor search for each c_i of $B_i \in \mathcal{B}$. We assume that the upper bound for the maximum degree of any road junction is 10 and, moreover, on a random set of 20 tiles in our dataset, we find the maximum degree of any junction to be 4. If a pair of disks centered at c_i and c_j either touch or intersect, we add a new edge (c_i, c_j) to G_{disk}. Although this method takes $O(n \log n)$, since the maximum number of disks in a given square tile with the side length of 1250 feet is ≤ 400, we do not see significant performance penalties. While polynomial time ($O(n)$) approximation schemes [63] can be used in cases where bigger tiles with larger number of disks are considered, we use smaller tiles and can compute the exact solution without having significantly longer execution times.

We should be careful while inserting edges based on disk intersection. In case there are three disks that intersect each other, c_i, c_j and c_k, as shown in Figure 3.6, the graph will end up with a triangular loop. This loop translates to two U-turns within a single road width which is not a feasible scenario in road engineering [66]. So, we avoid a triangular loop by discarding the edge which has the smallest length.

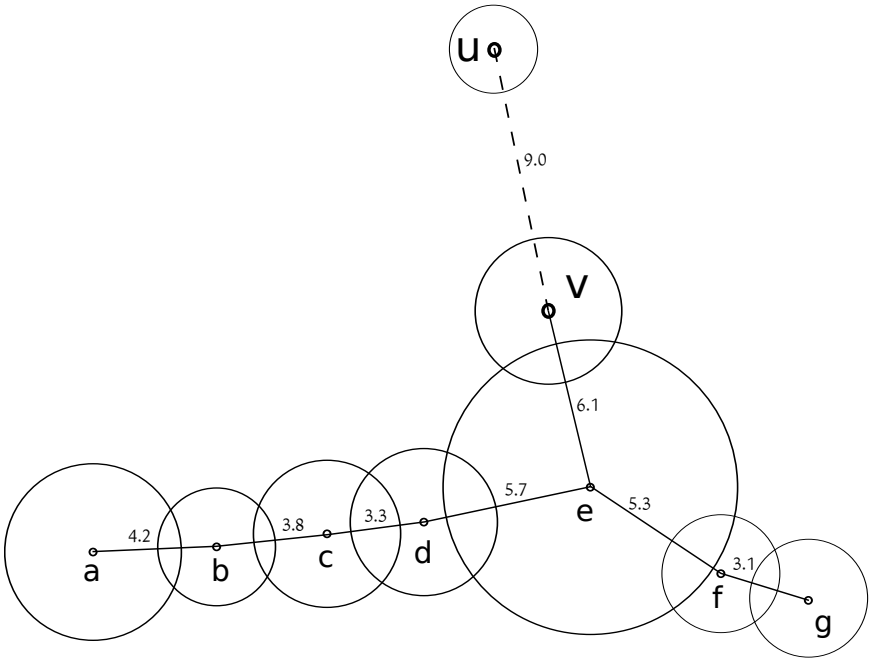


Figure 3.7: Outlier edge for degree 0 connection. u is a degree 0 node with v as its nearest neighbor, so, (u, v) is a candidate edge. E_v is the set of edges that belong to the connected component of v. Each edge is labeled with its length. Here, $\mu_{E_v} = 4.5$ and $\sigma_{E_v} = 1.2$ are the mean and the standard deviation of the lengths of of the edges in E_v respectively, and the length of the candidate edge $len(u, v) = 9.0$. Since, $len(u, v) - \mu_{E_v}| = 4.5 \geq 2.4 = 2 \cdot \sigma_{E_v}$, (u, v) is an outlier edge.

3.6.3 Connectivity

Owing to the imperfect segmentation, G_{disk} still has missing links. The task that remains is to predict these links and insert them in G_{disk}. Often there are neighboring regions in I which are partially segmented as roads but could not be covered by any disk in $\mathcal{B}$. This allows us to apply voting while predicting edges. Given a candidate edge $e = (u, v)$, we impose the following conditions for it to be a valid edge:

Condition 0: If u is a degree 0 node with v as its nearest neighbor and E_v is the set of edges in the connected component of G_{disk} containing v, we check if e is an outlier edge based on the distribution of the lengths of edges in E_v. If μ_{E_v} and σ_{E_v} are the mean and the standard deviation of the lengths of of the edges in E_v, and $len(e)$ is the length of $e = (u, v)$, e is

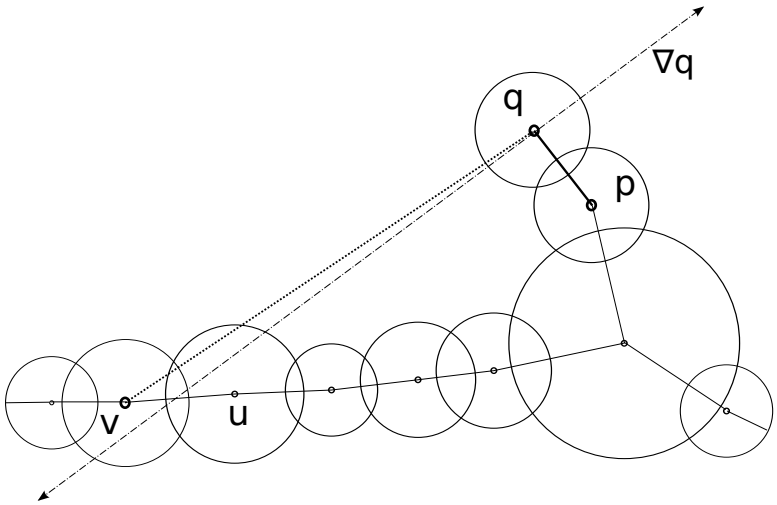


Figure 3.8: Candidate edge (q, v) intersects disk at u. q is a degree 1 node connected to p. (q, v) is a candidate edge because v is the nearest neighbor of q in the cone ∇_q with $\theta_{\nabla_q} = \pi$. Although (q, v) may satisfy the road vote count (*Condition 1*), it creates an incorrect topology and is discarded since it intersects an existing disk at u (*Condition 2*).

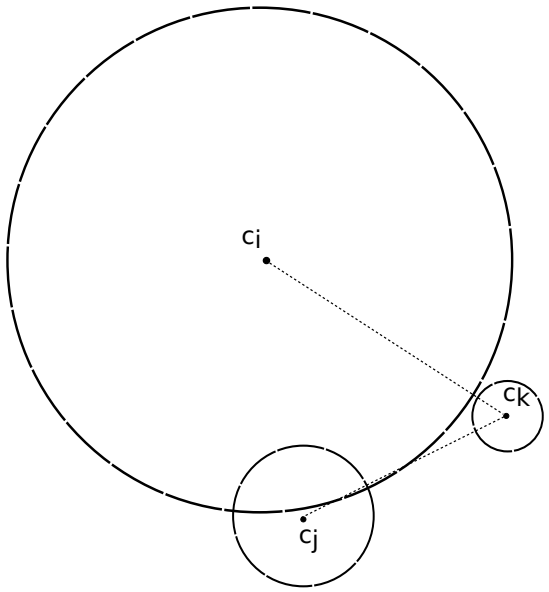


Figure 3.9: Candidate edge crosses disk. Given three disks centered at c_i, c_j and c_k with radius r_{c_i}, r_{c_j} and r_{c_k} respectively where c_j and c_k are nearest neighbor to each other since $d(c_i, c_k) > d(c_j, c_k)$, we do not insert the candidate edge (c_j, c_k) as it intersects the disk of c_i. We invoke this condition (*Condition 2*) since we are finding the nearest disks based on the disk centers and not on the distance between their circumferences.

considered to be an outlier edge and, thus, discarded if:

$$|len(e) - \mu_{E_v}| \geq 2 \cdot \sigma_{E_v}$$

Figure 3.7 shows an example.

Condition 1: Consider the strip of width β pixels centered on the segment (u, v). If γ-fraction of the pixels in this strip are classified to be road then this predicate returns true. Both these parameters are computed using hyperparameter tuning (Section 3.6.3).

Condition 2: If e intersects only $\{B_u, B_v\} \in \mathcal{B}$ then return true (A false example is shown in Figure 3.8).

In our Algorithm, Condition 2 is checked only if Condition 1 returns true. Now we are ready to explain how we generate our candidate edges.

Choice of Nodes. For predicting the missing edges, we consider only those candidate edges that will be incident on degree 0 and degree 1 nodes in G_{disk}. This decision is based on the analysis of the nodes present in a randomly sampled set of 20 tiles each covering an area of $1250 \times 1250 ft^2$. Each of these tiles contains a node at every junction and at every significant bend as well as at every 100 (or less) feet. Since 91% of the nodes in the graphs are degree-2 nodes, there is a > 0.9 probability that, for a missing edge (u, v), u and v are degree 2 nodes which appear as degree 1 or degree 0 nodes in G_{disk}. Out of the 258 junctions (nodes with degree > 2) present in the graphs, 236 are degree-3, 22 are degree-4 and no nodes have degree ≥ 5. Moreover, only 7 junction pairs in the chosen tiles lie adjacent to each other and the rest have degree 2 nodes between two consecutive junctions. So, even for junctions, in most cases, missing edges result in adjacent degree 1 nodes, and inserting the missing edge for such degree 1 nodes will automatically insert the missing edge of the junctions. Thus, for predicting the missing links, we focus on degree 0 and degree 1 nodes in G_{disk}.

Establishing Connections. We first handle the connectivity of degree 0 vertices. We connect each degree 0 vertex, q with its nearest neighbor, v if the new edge (q, v) satisfies the two conditions mentioned above. After this, we drop any remaining degree 0 vertex as

outliers. Next, we process the degree 1 vertices. A degree 1 vertex q has a direction because it is an end point of an existing edge (p, q) which is also a line segment on the plane. Since roads are less likely to make sharp turns, we assume (p, q) will maintain its original direction and the vertex v to which q will eventually be connected will lie in the cone with q as the apex. The axis of this cone is obtained by extending the edge (p, q)beyond the point q. To give some room to wiggle while searching for u, we define a cone ∇ of angle θ_∇ and length r_∇ at q. If v is the nearest neighbor of q that falls within the cone ∇, that is the angle at q made by the segments (p, q) and (q, u) is greater than $\pi - \frac{\theta_\nabla}{2}$, then (q, v) is a candidate edge. If the two conditions are satisfied, we insert (q, v) into G_{disk}.

Our algorithm is similar to provable curve reconstruction algorithms [49, 67], where a set of points, P, which is an ϵ-sample of a smooth curve C, is taken as the input and used to generate a graph G. The graph G connects sample points which are adjacent to each other by connecting nearest neighbors and then considering the half-space opposite to a degree 1 node for candidate edges. However, we use a cone of angle θ_∇ and tune it as a hyperparameter by considering values from $\frac{\pi}{3}$ to $\frac{7\pi}{6}$ and using increments of $\frac{\pi}{6}$. Serendipitously, we get the best results by using $\theta_\nabla = \pi$, which matches with the theoretically provable algorithm. Hence, on the basis of the proofs provided in [49], for all parts of the road (smooth curve), where the ϵ-sampling property is satisfied, our algorithm is capable of correctly reconstructing the curve.

Connectivity from Thinning. In the process of selecting disks that have radius of at least r_{min}, we discard those road segmentation regions that are thinner than r_{min}. Although it is advantageous in removing outliers, it can result in missing links even at well-connected but thin regions. Thinning can be a boon in this scenario because it preserves the road continuity even when there is just a single road pixel along the path. Thus, we also consider the edges given by thinning while trying to connect the degree 1 nodes.

We first apply thinning as described in Section 3.5.2 to get the graph G_{thin} from I. We register the vertices of G_{disk} with those of G_{thin} with the method described in Section 3.7. For a degree 1 node $u \in G_{disk}$ we scan its cone neighborhood to find the candidate node $v \in G_{disk}$. We then find the corresponding vertices $u^*, v^* \in G_{thin}$. If there is a path in

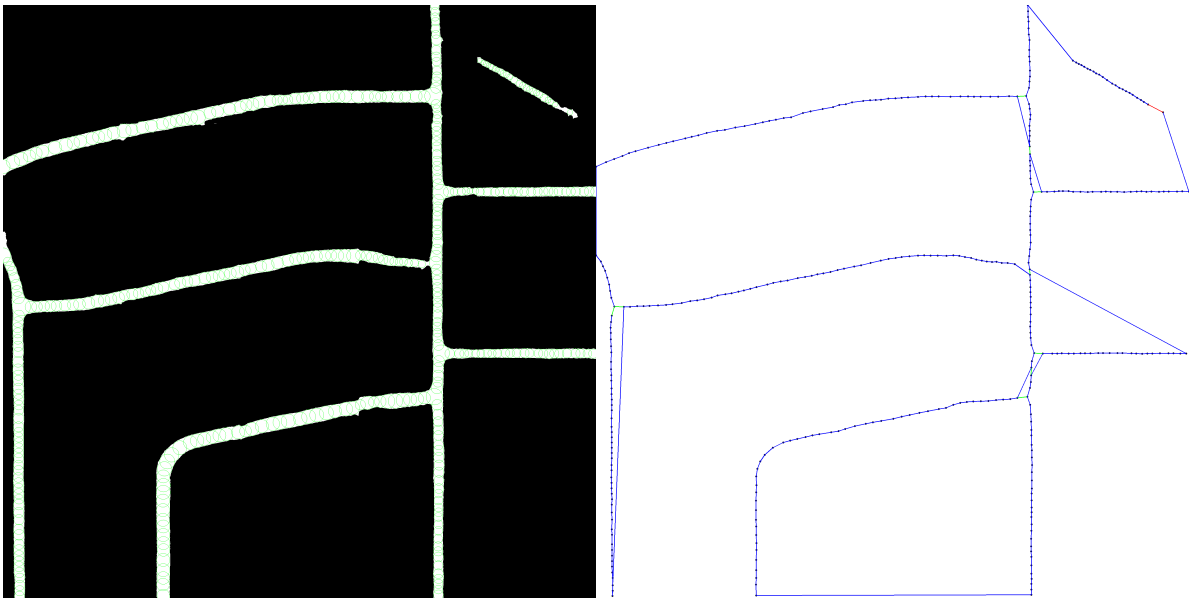


Figure 3.10: A Traveling Salesman Tour (right, blue) along the vertices of the disk intersection graph (left). The white regions in the left image correspond to road segmentation and the green circles are the packed disks. The tour captures most of the edges but it also adds edges at places where there are no roads. In some cases, the tour connects nodes that are not the closest so as to avoid visiting them more than once. So, TSP tour doesn't help in determining the connectivity.

G_{thin} between u^* and v^* and the shortest path length between the two is within α of the distance between u and v, then we add an edge (u, v) into G_{disk}. We treat α as a learnable hyperparameter. Based on our experiments, $\alpha = 0.95$.

A Traveling Salesman Tour [1] with Euclidian distance as the edge cost can also give candidate edges but we chose not to use it because the tour is optimized to capture the global context whereas the missing links are mostly local. In some experiments we observe that the TSP tour evades even those vertex pairs that are nearest to each other and have $> 90\%$ of the road votes (Figure 3.10).

Hyperparameters. There are a few hyperparameters that govern our disk-packing based graph extraction method. We tune them with the Tree of Parzen Estimators algo-

[1] http://www.math.uwaterloo.ca/tsp/concorde.html

Table 3.2: Hyperparameters for disk packing based method for road network extraction.

Hyperparameter	**Symbol**	**Value**
Cone Angle	θ_{∇}	π
Cone radius	r_{∇}	225 feet
Shell Expansion Factor	Δ_r	1.3
Minimum Disk Radius	r_{min}	6.5 feet
Road Vote Count Width	β	11 feet
Road Vote Fraction	γ	0.2
Morphology Radius	r_{morph}	4.5 feet

rithm [36] on a random set of 20 tiles using the inverse of the Weighted Shortest Path metric (Section 3.7) as the loss. We list their corresponding values in Table 3.2.

3.7 Road Network Graph Similarity Metrics

Evaluation of a road network graph construction algorithm needs suitable metrics that can validate the predicted graph $G = (V, E)$ and quantize its quality with respect to the corresponding ground truth $G^* = (V^*, E^*)$. If G and G^* are similar in terms of vertex reachability, topology and path distance between registered vertices, then we can say that the algorithm performs well. For road networks, the ground truth usually consists of: (1) Map information containing road centerlines from OpenStreetMap[2] (2) Centerline graph extracted directly from the manually labeled road masks.

There are generic graph similarity metrics [68] to compare graphs, such as graph edit distance [69] but we can not apply these metrics directly on our problem. The vertices of road networks are locally embedded in a two dimensional Euclidian space with latitude and longitude as the coordinates and the same network can be represented using an infinite number of different graphs. To address these concerns, Wiedemann et al. [70] introduce quality measures of an extracted road network in terms of completeness and correctness. For this, they first match the extracted graph segments with the ground truth segments and vice versa by building buffers around the reference segments. If a predicted segment falls

[2] `https://www.openstreetmap.org/`

within certain width and angle of a ground truth segment, the two are considered to be a match. They then build a confusion matrix based on the matched/unmatched counts and define the quality measures.

If we consider the predicted and the ground truth graphs as two point sets of graph vertices, we can apply the **Hausdorff Metric** to find the distance between the two sets. Given the predicted vertex set V and the ground truth set V^*, the directed-Hausdorff metric from V to V^* is computed as follows:

$$\delta_H^{'}(V, V^*) = \max_{u \in V} \min_{v \in V^*} \|u - v\|$$

The Hausdorff metric is then given by:

$$\delta_H(V, V^*) = \max(\delta_H^{'}(V, V^*), \delta_H^{'}(V^*, V))$$

Since graph construction is not accurate there can be outlier vertices in V. This makes taking the maximum of the pair-wise distances as a similarity measure problematic. As a solution, fractional Hausdorff distance can be computed by taking the k-th largest distance instead of the maximum. Still, in this metric, the points are processed independently without considering the topology of the embedded graph. Thus, it is not a good metric for our task.

Another candidate metric is the **Frechet distance**. Given two ordered point sets (or paths) P and P^* and their corresponding parameterizations α and β, the Frechet distance between the two paths is given by:

$$\delta_F(P, P^*) = \inf_{\alpha, \beta} \max_{t \in [0,1]} \|P(\alpha(t)) - P^*(\beta(t))\|$$

Since this metric considers topology of the paths, it is applicable to road network graphs. Still, the graph may contain multiple paths or curves with many junctions. So, to effectively use Frechet distance, some sort of path alignment between the ground truth and the predicted graphs is required.

There are other graph similarity metrics that are applicable to road network graphs which we discuss in the remainder of this section.

3.7.1 Shortest Path Based Metric

Wegner et. al [71] give an evaluation measure based on shortest paths. A source and destination pair (u^*, v^*) is chosen from V^* and the length of the shortest path between these points $l_{(u^*,v^*)}$ is computed. Corresponding to (u^*, v^*), vertex pair (u, v) is chosen from V with $l_{(u,v)}$ as the shortest path length. A number of such source and destination pairs are selected at random and they are categorized into four groups: (a) *correct* when $\left|l_{(u^*,v^*)} - l_{(u,v)}\right| \leq 0.05 \times l_{(u^*,v^*)}$ (b) *too short* when $l_{(u,v)} < 0.95 \times l_{(u^*,v^*)}$ (c) *too long* when $l_{(u,v)} > 1.05 \times l_{(u^*,v^*)}$ (d) *infeasible* when there is no path between u and v. The percentages of vertex pairs that fall in these categories give measures of graph correctness (*correct*), wrong edges (*too short*) and missing edges (*too long*, *infeasible*). We can also assume that the pairs which fall in the *correct* category are aligned. We can then compute Frechet distance on those paths.

3.7.2 Junction Metric

Bastani et. al [7] define a metric based on junctions. In this metric, first, vertices with degrees greater than 2 are taken from both V and V^* as junctions. A mapping registers the two sets of junction vertices. If junction $u^* \in V^*$ corresponds to the junction $u \in V$, two numbers based on edge incidence are computed: (a) $f_{u^*,correct}$ indicates the fraction of edges incident on u^* that are captured around u. This is proportional to the true positive count (b) $f_{u,error}$ indicates the fraction of edges in u that appear around u^* which is dependent on both wrongly predicted edges (false positive) and missed edges (false negative). Instead of relying only on degrees of the junctions, individual edges are identified based on their angle of incidence to compute this metric. It is still not a foolproof metric because in case an entire road segment that doesn't contain any junction is missing, junction metric does not take that into account.

3.7.3 TOPO Metric

Biagioni et. al [44] define a metric based on random exploratory sampling which considers both topology and geometry of the road network graph. They first randomly select a starting location u^* on the ground truth graph and keep track of the paths that radiate outwards up to a certain pre-defined distance. Along each path, evenly spaced discs of fixed radius (called *holes*) are sampled with interpolation. They do the same for $u \in V$ on the predicted graph that corresponds to $u^* \in V^*$ and, here, the discs are called *marbles*. Given these sets of *holes* and *marbles*, a matching is carried out in which if a *marble* within a specified distance from a *hole*, a match occurs. This gives counts of marbles and holes which either matched or mismatched. The same process is repeated for multiple starting locations and a final F-score is computed. If the counts of matched marbles and holes are higher, the F-score also remains higher which finally indicates a higher accuracy in road network graph extraction.

3.8 Experiments

3.8.1 Set Up

We run multiple experiments on the road segmentation produced by M_{seg} from the aerial dataset of Tallahassee. Each experiment used a single 11 GB GeForce GTX 1080 card on an Intel i7-based workstation with 32 GB RAM.

Dataset. The graph extraction experiments proceed in two stages, namely, road segmentation and the translation of output segmentation into road network graphs. We use the same dataset described in [1] but we split the tiles differently as shown in Figure 3.11. We chose this split so as to get larger contiguous regions instead of scattered, non-adjacent tiles for train and test. This ensures road network continuity over a larger region. The dataset has 36 and 27 10000×10000 tiles for train and test respectively and we split each into 500×500 pieces. Thus, the training and test sets include 14400 and 10800 500×500 tiles respectively. Instead of subsampling, we use the whole training set to train M_{seg}.

Figure 3.11: Split of Tallahassee into train (right of the blue margin) and test regions.

For graph extraction, we stack 25 adjacent 500×500 tiles to get larger 2500×2500 tiles because the tile size affects the accuracy of graph extraction. The smaller the tile, the higher the chances of the road regions getting split at the edges at various angles which, in turn, gives road segments of incorrect widths and shapes. This problem can be minimized with larger tile sizes but it cannot be completely eliminated.

Ground Truth Graphs. We consider two ground truth graphs for training and evaluation (a) graph obtained from OpenStreetMap, OSM-GT (b) graph generated from the ground truth road segmentation by applying thinning, TLH-GT. On visual inspection we find differences between TLH-GT and OSM-GT. OSM-GT considers parking lots as part of the road network whereas TLH-GT does not. In case of multi-lane roads that do not have well defined medians, TLH-GT ends up with a single edge owing to the thinning algorithm whereas OSM-GT usually has an edge for every lane. Due to these reasons there are more road edges in OSM-GT than TLH-GT. Besides, TLH-GT is more suitable for the extraction of road network graphs, and this is evident from the fact that all the methods except RoadTracer trained on OSM-GT perform better when compared to TLH-GT rather than OSM-GT. If we compute the Weighted Shortest Path metric (Section 3.7) between OSM-GT and TLH-GT with OSM-GT as the ground truth and measure the number of correct paths in TLH-GT compared to OSM-GT, the *Correct* percentage is only 22%, whereas, with TLH-GT as the ground truth and the OSM-GT as the predicted graph, the same metric is 76%. This indicates that OSM-GT has several extra edges that are not present in TLH-GT and this matches with our intuition based on the visual inspection. So, the evaluation metrics will also differ when we compare the predicted graphs with the two ground truth graphs.

Baselines. We compare our disk-packing method (Disk-M_{seg}) described in Section 3.6 with two baselines. The first is RoadTracer [7] for which we train two different models, namely, RoadTracer-OSM and RoadTracer-TLH with OSM-GT and TLH-GT as the respective ground truths. The second major baseline is DeepRoadMapper [2]. For these baselines, we use the implementation provided by Bastani et al [7]. We also compare with the graphs produced after directly applying thinning on M_{seg}'s output I (Thin-M_{seg}). Additionally, we disk-pack DeepRoadMapper's segmentation output to get Disk-DeepRoadMapper.

RoadTracer is highly affected by the choice of starting locations. To select at least one starting location per road segment during prediction, we take two approaches: (1) randomly select a node from each connected component in the ground truth graph ($start = random$) (2) apply disk-packing on I and pick all the centers of the packed disks ($start = \mathcal{B}$).

Evaluation Metrics. We evaluate the predicted graph $G = (V, E)$ with the corresponding ground truth graph $G^* = (V^*, E^*)$ with respect to vertex reachability, topology and path distance between registered vertices using three metrics. While Junction Metric [7] measures the number of *Correct* and *Extra* junctions, TOPO metric [44] evaluates the graphs in terms of topology using Spurious and Missing and utilizes the values of Spurious and Missing to compute the F-score, which lies in the closed interval $[0, 1]$.

The original Shortest Path metric [71] gives equal weights to all source/destination pairs and keeps track of the *Correct*, Short, Long and NoPath counts but the longer shortest paths in the ground truth are more valuable for the metric. Thus, we modify the metric by multiplying each count with $\frac{l_{u,v}}{l_{diag}}$, where l_{diag} is the length of the longer diagonal of the tile. In our case, $l_{diag} = 2500\sqrt{2}$ since we have square tiles with side length 2500 pixels. Finally, we normalize the values by dividing each score by the sum of all the weights and report the percentage of *Correct* edges in our Weighted Shortest Path metric (W-SP).

3.8.2 Vertex Registration for Metric Computation

One of the common requirements of most of the graph similarity metrics described above is the registration of the ground truth vertex set with that of the predicted set. To register $u^* \in V^*$ with one of the vertices in V, we first find the edge $(u, v) \in E$ which is nearest to u^* and within a pre-defined threshold r_{max}. We set $r_{max} = 30$ feet which is the maximum lane width for US highways [72]. We project u^* onto (u, v) and get a new vertex p between u and v. If p is within r_{min} of one of the end points of (u, v), say u, then u^* is registered with u. Otherwise, we replace the edge $(u, v) \in E$ with two new edges, (u, p) and (p, v) and then establish correspondence between u^* and p. If there is no edge in E that lies within r_{min} of u^*, u^* remains unregistered. Although this procedure modifies the predicted graph G, its topology remains unaffected.

Table 3.3: Hyperparameter tuning with the Tree of Parzen Estimators for maximizing the shortest path metric (Sample A).

	θ_∇	r_∇	Δ_r	β	r_{min}	r_{morph}	γ	α	Score
0	150.0	1100.0	1.15	2.0	21.0	13.0	0.70	0.55	0.201494
1	120.0	500.0	1.15	2.0	20.0	22.0	0.60	0.60	0.266556
2	150.0	150.0	1.80	12.0	26.0	24.0	0.10	0.90	0.240188
3	120.0	850.0	1.70	16.0	29.0	24.0	0.30	0.55	0.272510
4	180.0	300.0	1.55	10.0	6.0	12.0	0.15	0.95	0.678137
5	150.0	800.0	1.60	14.0	22.0	11.0	0.40	0.90	0.590250
6	60.0	250.0	1.40	20.0	6.0	21.0	0.25	0.60	0.609894
7	210.0	700.0	1.60	6.0	17.0	17.0	0.20	0.85	0.563332
8	210.0	300.0	1.55	2.0	21.0	13.0	0.90	0.85	0.316234
9	60.0	200.0	1.40	8.0	12.0	24.0	0.70	0.85	0.517169
10	180.0	350.0	1.75	14.0	29.0	10.0	0.80	0.80	0.598934
11	90.0	650.0	1.40	2.0	26.0	28.0	0.60	0.55	0.600365
12	120.0	600.0	1.90	10.0	9.0	19.0	0.15	0.70	0.721523
13	120.0	450.0	1.50	14.0	9.0	5.0	0.50	1.00	0.673901
14	**180.0**	**550.0**	**1.30**	**22.0**	**13.0**	**9.0**	**0.20**	**0.95**	**0.754523**
15	180.0	400.0	1.95	8.0	9.0	30.0	0.35	0.70	0.739710
16	120.0	1200.0	2.00	10.0	11.0	6.0	0.35	0.75	0.739737
17	180.0	100.0	2.05	16.0	7.0	29.0	0.55	0.65	0.721471
18	120.0	600.0	1.30	22.0	13.0	9.0	0.15	0.80	0.714752
19	180.0	1000.0	1.95	12.0	16.0	26.0	0.45	0.65	0.725599
20	120.0	200.0	1.25	20.0	15.0	15.0	0.20	0.95	0.747084
21	90.0	500.0	1.15	22.0	11.0	12.0	0.55	1.00	0.722234
22	60.0	1000.0	1.35	18.0	12.0	11.0	0.40	1.00	0.637874
23	120.0	150.0	1.80	8.0	21.0	17.0	0.35	0.85	0.429620
24	210.0	750.0	1.20	14.0	14.0	18.0	0.30	0.95	0.725669
25	150.0	200.0	1.10	16.0	15.0	18.0	0.45	0.85	0.720074

Table 3.4: Hyperparameter tuning with the Tree of Parzen Estimators for maximizing the shortest path metric (Sample B).

	θ_∇	r_∇	Δ_r	β	r_{min}	r_{morph}	γ	α	Score
26	150.0	650.0	1.25	4.0	24.0	22.0	0.50	1.00	0.660403
27	120.0	1100.0	1.45	20.0	20.0	16.0	0.65	0.90	0.690810
28	210.0	400.0	1.10	22.0	19.0	14.0	0.80	0.80	0.692679
29	180.0	250.0	1.35	20.0	17.0	20.0	0.20	0.90	0.698335
30	90.0	350.0	1.35	12.0	13.0	12.0	0.10	0.95	0.477270
31	180.0	550.0	1.70	20.0	5.0	19.0	0.25	0.80	0.428533
32	60.0	900.0	1.50	16.0	16.0	9.0	0.30	0.85	0.429443
33	180.0	800.0	1.25	4.0	28.0	8.0	0.15	0.95	0.423864
34	180.0	250.0	1.45	16.0	16.0	23.0	0.25	0.60	0.726165
35	150.0	550.0	1.65	14.0	15.0	14.0	0.40	0.90	0.677162
36	120.0	750.0	1.15	18.0	12.0	6.0	0.60	1.00	0.665423
37	180.0	600.0	1.60	22.0	22.0	20.0	0.20	0.50	0.659846
38	210.0	300.0	1.55	20.0	10.0	10.0	0.90	0.95	0.702622
39	60.0	1100.0	1.20	6.0	7.0	7.0	0.45	0.80	0.476273
40	120.0	450.0	1.40	14.0	17.0	13.0	0.80	0.85	0.393586
41	180.0	450.0	1.70	22.0	24.0	11.0	0.65	0.95	0.397034
42	90.0	150.0	1.35	18.0	20.0	25.0	0.95	0.75	0.400976

3.8.3 Uniform Vertex Distribution

Some of the similarity metrics, for example TOPO and shortest-path, depend on vertex sampling. The distribution of vertices in the road network graphs may not be uniform depending upon how they were generated. During manual labeling, the labeler may try to minimize the number of edges along the road centerlines. In case of curve reconstruction algorithms, the number of points sampled is directly proportional to the local road curvature [50]. Path simplification with Ramer-Douglas-Peucker algorithm [62] also gives a similar sampling. Thus, to make the vertex distribution uniform, we insert new, evenly spaced vertices into existing edges of the graph without altering its topology.

3.8.4 Results

Our method (Disk-M_{seg}) achieves the best performance for all the metrics when compared to TLH-GT. We present our results in Table 3.5 and identify the best performing methods

Table 3.5: Comparison of graph extraction methods with OpenStreetMap (OSM-GT) as the ground truth.

Method	**OSM-GT**					
	Junction		TOPO			W-SP
	Correct	Extra	Spurious	Missing	F	Correct
RoadTracer-OSM (random)	0.34	0.23	0.27	0.68	0.45	0.09
RoadTracer-TLH (random)	0.15	**0.20**	0.21	0.78	0.34	0.04
RoadTracer-OSM ($\mathcal{B}$)	**0.39**	0.26	0.29	0.54	0.56	0.14
RoadTracer-TLH ($\mathcal{B}$)	0.17	0.27	0.22	0.63	0.50	0.06
DeepRoadMapper	0.28	0.60	0.35	**0.44**	0.60	0.13
Disk-DeepRoadMapper	0.34	0.46	0.30	0.45	**0.62**	0.18
Thin-M_{seg}	0.31	0.86	0.50	0.47	0.52	0.20
Disk-M_{seg}	0.26	0.32	**0.19**	0.51	0.61	**0.16**

Table 3.6: Comparison of graph extraction method with Tallahassee road segmentation data (TLH-GT) as the ground truth.

Method	**TLH-GT**					
	Junction		TOPO			W-SP
	Correct	Extra	Spurious	Missing	F	Correct
RoadTracer-OSM (random)	0.39	0.73	0.43	0.67	0.42	0.15
RoadTracer-TLH (random)	0.37	**0.29**	0.24	0.68	0.45	0.14
RoadTracer-OSM ($\mathcal{B}$)	0.48	0.70	0.44	0.49	0.54	0.24
RoadTracer-TLH ($\mathcal{B}$)	0.42	0.37	0.26	0.49	0.61	0.20
DeepRoadMapper	0.57	0.75	0.40	0.30	0.64	0.40
Disk-DeepRoadMapper	0.69	0.65	0.35	0.28	0.68	0.56
Thin-M_{seg}	0.81	0.90	0.48	**0.22**	0.62	**0.75**
Disk-M_{seg}	**0.72**	0.39	**0.21**	0.27	**0.76**	**0.75**

Table 3.7: The per-tile and the global counts of the graph edges given by Disk-M_{seg} grouped based on how the edges were added.

Edges From	**Per-tile Count**	**Global Count**
Disk-intersection	357.532	149091
Degree 0	0.367	153
Degree 1	5.149	2147
Thinning	0	0
Total	363.048	151391

for each metric. In case of junction metric, we highlight the methods that perform well for the *Correct* and *Extra* scores individually. Besides, we also identify the method (in italics) that achieves the best balance between *Extra* and *Correct*, indicating the ability to detect junctions correctly without detecting too many spurious junctions. For TOPO and W-SP metrics, we highlight the best performing method for F-score and Correct respectively.

While our method produces the best performance for TLH-GT, it is also competitive in terms of TOPO and Weighted Shortest Path metric for OSM-GT. In fact, the evaluation metrics for all the reported methods are far better when we use TLH-GT instead of OSM-GT as the ground truth because OSM-GT has more road edges (Section 3.8.1) compared to TLH-GT. So, during metric computation with OSM-GT, a major fraction of the sampled vertices can belong to these extra edges and reduce the metric scores.

RoadTracer performs poorly for almost all the metrics in case of TLH-GT and only achieves good performance for junction metric in case of OSM-GT. However, RoadTracer has the advantage of utilizing existing graphs, and can be trained separately for OSM-GT and TLH-GT. Our dataset covers a vegetation rich geographic region with a small urban core because of which it is not guaranteed that the selected starting locations or the locations selected by its decision CNN will fall exactly on the roads. If such a location happens to fall on the canopy, RoadTracer directly terminates its search missing out on detecting significant portions of the roads. It also excludes regions around the tile edges because its decision CNN uses a window of a fixed size. Table 3.5 shows that RoadTracer's performance is affected by the choice and the number of starting locations. The metrics are better if we use all disk centers in $\mathcal{B}$ instead of one randomly selected point per connected component because number of connected components $\ll |\mathcal{B}|$ and points in $\mathcal{B}$ are more likely to be on visibly cleaner road regions. On the other hand, DeepRoadMapper and M_{seg} can only be trained on segmentation masks, which are available for TLH-GT only and causes the segmentation to be performed in accordance with TLH-GT.

From the results in Table 3.5, we observe that the quality of the output graph depends on the IoU metric given by the road segmenter. M_{seg} with grid-shifting (81.2% IoU) is better

than DeepRoadMapper (68.3% IoU). So, irrespective of which graph extraction method we use, the majority of the metrics are higher for M_{seg}'s output.

Our Disk-packing based graph extraction approach performs better than thinning and has several advantages over thinning. Between Thinning and Disk-packing for the same M_{seg} road segmentation, both give almost equal accuracy in terms of connectivity and correct paths. As we see in Table 3.7, Disk-M_{seg} on average adds at least 5 missing edges per tile. Thin-M_{seg}, by design, does not add any new edges but still performs as good as Disk-M_{seg} in the Weighted Shortest Path metric. This can be attributed to the fact that Thin-M_{seg} preserves the road continuity even when there is just a single road pixel along the path whereas Disk-M_{seg} does not place disks on regions that are thinner than r_{min} which results in missing links. Since Thin-M_{seg} is better in this regard, we can borrow connections into the disk-packed graph from the thinned graph where possible. We tried to borrow edges from the skeleton graph G_{thin} as described in Section 3.6, but due to the two conditions (*Condition 1*, *Condition 2*) and our choice of only degree 0 and degree 1 nodes, we were not able to add any new edges(Table 3.7). As thinning allows outliers and unwanted branchings, it predicts spurious edges which adversely affect the F-score in case of TOPO and increase the number of extra junctions in Junction metrics. For these metrics, our disk-packing method captures significantly fewer number of erroneous junctions compared to thinning (a margin of 51%) and also performs 14% better in TOPO's F-score.

3.9 Conclusion and Future Work

The aim of this paper was to use both aerial Lidar and images to extract roads as graphs. We presented an algorithm based on disk-packing to transform road segmentation output to a road network graph where the edges of the graph lie along the road centerlines. We observed that our method is better at discarding outliers and is able to capture the network topology more accurately while being almost close to the existing method that uses thinning in terms of connectivity. As our approach de-couples segmentation from graph reconstruction and since better segmentation always translates to a more accurate graph, we can aim for optimizing road segmentation with the best available CNN architecture. M_{seg}

and our graph extraction method can be incorporated with existing road graph extraction methods like DeepRoadMapper and RoadTracer, in order to improve their performance.

There are still ambiguities related to the ground truth graphs due to parking lots, medians and lane discrimination. In the near future we plan to address these issues by segmenting each ambiguous object separately and building the graph in a fine-grained fashion. Another aspect that we have not explored in this paper is the reconstruction of directed graphs, especially using a reduction from problems in TSP, linear programming or convex quadratic programming.

CHAPTER 4

NON-PARAMETRIC ESTIMATION OF TRUTH FROM AUTHORITATIVE SOURCES

4.1 Introduction

We frequently get conflicting information from multiple sources about an item of interest and need to make a choice as to which source to trust. From booking a hotel, to finding the information about a book, this challenge of truth finding is present everywhere. Search engines can be partially helpful as they provide a ranking of the sources based on a ranking algorithm. However, even highly ranked sources often disagree. Thus, it would be useful to have an algorithm that is able to filter out all the inconsistencies and identify the most reliable source.

Several variations of the *Truth Finding* problem have been studied [73, 74, 75, 76, 77]. Most of the algorithms for this problem are optimized for either numeric or non-numeric data [77]. In practice, datasets are often a collection of different data types and as such a unified framework for truth finding is necessary. Furthermore, there are many scenarios where a small set of authoritative sources are available, but conflicting. Thus designing an unified, automated truth determination system, for conflicting authoritative sources is important and is the primary motivation of our work.

There are several problems in trying to create a unified approach for truth determination. Unlike numeric data, it is not immediately obvious how to define correlation between non-numeric data and even if it is done, then also it is not clear how this information can be leveraged by any truth finding algorithm. In case of numerical data it is fairly intuitive to model the truth generation process as a real valued random variable [77]. It is impossible to use such a generative model for non-numeric data, because methods like maximum like-

lihood (ML) work in the real space [78]. Moreover, if the number of samples is small, as is the situation in our case, the ML estimates can be highly biased [78], even for numeric data.

For numeric data, similarities between reported values can be leveraged to determine the truth. For example, if three authoritative sources report the price of a book as \$20, \$21 and \$100, it is highly likely that the actual price of the book is near the former values rather than the later. A similar approach can be used for non-numeric data. There is a rich theory of learning with similarity functions [79] and it is possible to define similarity functions for different data types. If the truth finding algorithm uses similarities between the values reported by different sources, it can work with any data type, as long as it is possible to define a similarity function. This observation was first used for truth determination in case of real valued data by Zhao et al. [77] and is the primary motivator for this work.

An important aspect of any truth finding algorithm involves the estimation of source qualities. As noted in [77], the quality of a source not only depends on the proportion of correct answers but also on how often the value is close to the truth. For numeric data this lets us model the source confidence as a random variable with an unknown mean and variance. For non-numeric data, this is not possible. Thus, for this type of data the source confidence can be estimated as the proportion of correct answers and the variance can be implicitly inferred using the similarities between the values reported by the sources. This motivates the use of source-to-source similarities in our algorithm.

The truth finding problem was first studied by Yin et. al [73]. Dong et. al [74] studied the impact of source dependency on truth finding and the effect of *endogenous* information on source confidence [80]. Li et al [81] studied the problem of trustworthiness of data provided by sources in deep web and that of detecting *copy dependency* among the sources [82]. More recently, truth finding problems have been studied by using generative models [77] and with the aim of providing a theoretical guarantee to the estimates [83]. In a recent paper Wan et al. [84] introduce and study the problem of finding *trustworthy opinion*. In another recent work Xiao et al. [85] study the problem of constructing confidence interval estimates for truth discovery.

This paper is organized as follows: we define our notations at the end of this section and formulate the truth finding problem in Section 4.3. We describe the proposed algorithms for Schema Matching and Truth Finding in Sections 4.4 and 4.5 respectively along with their theoretical backgrounds. We describe the experiments and the observed results in Section 4.6.

Notation: $\mathbb{E}$ is used to denote expectations and $\mathbb{P}$ to denote probabilities. π, Π, ϕ and φ are used for functions and f is used for embeddings. Other lowercase Greek letters like $\mu, \lambda, \eta, \alpha, \beta$ are used to denote constants. Points are denoted by lowercase letters like p, x, y. Vectors and matrices are denoted by boldface lowercase and/or uppercase letters respectively, like $\mathbf{x}$ and $\mathbf{X}$. Sets are denoted by calligraphic letters like $\mathcal{D}$. P is used for probability distributions.

4.2 Related Work

The problem of merging information from different sources has been studied before. The database community was aware of the problem of schema merging for quite a long time [86, 87, 88, 89, 90, 91]. The underlying problem in schema merging is as follows: given two database schemas, create a new schema by merging them together, so that certain constraints are satisfied. In this paper we study a problem that is slightly different from the schema merging problem. The goal is to be able to determine the most reliable source for an attribute of an object, from a collection of several different sources, that may potentially provide conflicting information. This problem is more relevant for unstructured data obtained from the Internet as responses for search queries. This problem was first studied in detail by Yin et. al [73]. They called the problem the *Veracity* problem and studied algorithms for the same. In order to find the most reliable source for a given attribute, they use two key ideas, namely, source confidence and fact confidence. They came up with a new algorithm which they called *TruthFinder* for iteratively estimating the source confidence and the fact confidence starting from an initial guess.

Several variation of this problem have been studied since then. Dong et al. [74] study a variation of the veracity problem, where explicit dependency between the sources is assumed.

The authors use a Bayesian Framework for detecting the dependencies and use the idea of a dependency graph for determining the most dependable source.

Apart from the regular Internet, the *Deep Web* is another source of data that is becoming more and more prevalent. Li et al [81] study the problem of trustworthiness of the data provided by sources of the deep web. They consider two types of data, stock data and flight data, and demonstrate that the information from the sources on the deep web are inconsistent. In a follow up paper Li at al. [82] study the problem of detecting the *copy dependency* among the sources of information on the deep web.

In another paper [80], Dong et al. looks at the problem of determining the trust worthiness of the sources using *endogenous* information. They argue that the actual trust worthiness of a source should depend on the quality of the information provided by the source and hence this should be the determining factor for computing the same. They introduce the idea of *Knowledge Based Trust* (KBT), that is used for computing the trustworthiness of a source by analyzing the quality of the information provided by the source.

We formulate the truth finding problem as a problem of *outlier detection* and propose an algorithm for iteratively removing outliers from a distribution of sources. Outlier detection problems have been studied for a very long time. Interested readers are directed to [92] for a survey and [93] and [94] for different approaches for solving the outlier removal problem.

4.3 Problem Formulation

Let $\mathcal{O}$ denote a set of objects. We represent an object by its set of attributes $O = \{t_1, t_2, \ldots, t_k\}$, which is the schema of the object. Let the set of different sources be denoted by $\mathcal{S} = \{S_1, \ldots, S_n\}$. For an object $O \in \mathcal{O}$, a source $S \in \mathcal{S}$ provides values for each attribute in a permutation of a subset of the schema for O. We denote this as $O^S = \{t^S_{i_1}, \ldots, t^S_{i_{k_S}}\}$, which denotes the values of the attributes $\{t_{i_1}, \ldots, t_{i_{k_S}}\}$. O^S is a permutation of a subset of $\{t_1, t_2, \ldots, t_k\}$, t_{i_j} is used to denote an attribute whereas $t^S_{i_j}$ is used to denote the attribute's value as reported by S. For an attribute t and a source S we denote the source confidence for the attribute by p^t_S. The truth finding problem can be stated as follows: given an object O and an attribute t find a source $S^* \in \mathcal{S}$ such that $t^{S^*}_{i_j} \geq_T t^S_{i_j}\ \forall S \in \mathcal{S}$. Note that $\geq_T$ is a

total order which denotes the fact that confidence of the left hand side is more than that of the right hand side of the inequality.

4.4 Schema Matching

Given an object $O = \{t_1, \ldots, t_k\}$, let $S_i, S_j \in \mathcal{S}$ be two sources for O. Let the values reported by S_i be $O^{S_i} = \{t^{S_i}_{l_1}, \ldots, t^{S_i}_{l_{k_i}}\}$ and that reported by S_j be $O^{S_j} = \{t^{S_j}_{m_1}, \ldots, t^{S_j}_{m_{k_j}}\}$ both of which which may be a permutation of a subset of $\{t_1, t_2, \ldots, t_k\}$, that is $k_i, k_j \leq k$, $k_i \neq k_j$. The *schema mapping* (or *matching*) problem seeks to find a bijective mapping π_{ij} such that: $\pi_{ij}(\Pi_{ij}(O^{S_i})) = \Pi_{ij}(O^{S_j})$ where Π_{ij} is the subset operator such that

$$|\Pi_{ij}(O^{S_i})| = |\Pi_{ij}(O^{S_j})| \leq k$$

This mapping is required because the truth finding algorithm works by considering pairs of attribute values. Hence, for every pair of sources, knowing this mapping allows us to compute the relevant pairs. Though not a part of the truth finding algorithm, this preprocessing step is required in many cases as the attribute mapping is not available.

We represent the value of each attribute in $O^{S_i} \cup O^{S_j}$ as a vertex of a complete bipartite graph $G_{ij} = (V_{ij}, E_{ij})$ where $V_{ij} = O^{S_i} \cup O^{S_j}$. There are $k_i \cdot k_j$ edges in the complete graph. Finding the bijective mapping π_{ij} is equivalent to determining a perfect matching in G_{ij}. We pose this as a simple binary classification problem.

Let us suppose that $\varphi_l; l = 1, \ldots, L$ are similarity functions such that:

$$\varphi_l(O^{S_i} \times O^{S_j}) \to \mathbb{R};\ l = 1, \ldots, L$$

Each of the $k_i \cdot k_j$ edges can be weighed by a vector of L similarity values, which are the results of applying the similarity functions $(\varphi_l; i = 1, \ldots, L)$, to the values at the incident vertices for that edge. Moreover, each edge of G_{ij} can be labeled by (+1) if that edge is in the perfect matching and (-1) if it is not. Thus determining the perfect mapping can now be cast as that of learning a binary decision boundary. An edge of G_{ij} is a vector in L-dimensional feature space and each vector has a label, that is either +1/-1. There are several algorithms that can be used to learn this decision boundary. For this work we decided to use the *Support*

Vector Machine [95] learning algorithm. We report the results of using this method on a dataset of book authors in Section 4.6.

We are now ready to describe the truth finding algorithm. For the rest of the discussion, we concentrate on a fixed attribute t of an object $O \in \mathcal{O}$ and describe a method of determining the truth for this attribute.

4.5 Truth Finding

Given two sources S_i and S_j $i \neq j$, an attribute t, let $\varphi_t(S_i, S_j) \rightarrow \mathbb{R}$ be a similarity function that takes as input the values for the attribute t as reported by the sources S_i and S_j respectively and returns a real number. Apart from assuming, that the sources are authoritative, we also assume that: 1) all the sources are independent and 2) the source confidence for a source S, for the attribute t, is estimated by the proportion of correct answers for t.

Our algorithm is based on the following observation: irrespective of whether the attribute is numerical or otherwise, the truthful source can be determined by modeling the similarities between the values reported by the different sources. This was first observed for real valued attributes in [77]. To this end we start by representing the sources as points in a metric space $M = (X, d_X)$, where the metric d_X is defined using $\varphi_t(S_i, S_j)$ $\forall i \neq j = 1, \ldots, n$. We first show that if these points conform to a certain distribution, defined by the intra-point distances, then a simple outlier removal algorithm can be used to iteratively remove outliers and estimate the truth. In order to remove the dependency on the form of the distribution, we embed the points in Euclidean space. Under the assumption of a low distortion embedding, we show that there is an algorithm for removing the outliers in a way that one can estimate an outlier free version of the underlying distribution.

In order to model the Euclidean embedding, we assume that there is a random variable F_t for the attribute t with a probability distribution P having location parameter μ and scale parameter σ. The Euclidean embedding f of the truth $t^{S_t^*}$ is such that for given $\epsilon > 0$, there exists a $\delta > 0$ such that $\mathbb{P}(||f(t^{S_t^*}) - \mu||_2 \leq \epsilon) \geq 1 - \delta$. Our assumption about the value $f(t^{S_t^*})$ is motivated by a similar assumption in the Gaussian Truth Model [77]. Under

this assumption, if we can estimate the variance of the underlying distribution well, then a sample from this *core* will be representative of the truth.

The first step of our algorithm is the generation of a complete graph that represents the similarities between the values reported by the different sources. Next, we use an outlier removal algorithm in the underlying metric space or an embedding of the same in the Euclidean space, for determining the truth. Now we describe these steps in detail.

4.5.1 Truth Discovery in Metric Space

Let $G(\mathcal{S}, E)$ be a graph with vertices denoting sources. For every pair of sources S_i, S_j $i \neq j$, there is an edge $e = (S_i, S_j) \in E$ with weight given by: $w_{ij} = p^t_{S_i} \cdot p^t_{S_j} \cdot \varphi_t(S_i, S_j)$ representing the assumption of independence. However, there can be other ways of modeling the dependency between sources and hence different representations for the weight. One approach would be to learn it from the data. We note that $G(\mathcal{S}, E)$ defines a finite metric space $M = (X, d_X)$, where the points in the metric space correspond to the sources and the distance between them correspond to the shortest path distance. The final step of our algorithm is to remove *outliers* from M in order to filter out the truth. To this end we first describe what we mean by an outlier. Our definition of an outlier is inspired by the definition of outliers given by Knorr et al. [96].

Definition 4.5.1 (λ-outlier) *Let $p^* \in X$ be a point representing the unknown truthful source S^*_t and the rest of the $n-1$ points represent the other sources. A point $p_o \in X$ is called the λ-outlier for $\lambda > 0$ if there exists a $\gamma > 0$ and $\lambda \gg \gamma$ and a ball $B(p^*, \gamma)$ that contains $n-1$ points such that $p_o \notin B(p^*, \gamma)$ and $\min_{p \in B(p^*, \gamma)} d_X(p, p_o) \geq \lambda$.*

Our definition of an outlier corresponds to that of Knorr et al. [96] because in our case a point p_o is an outlier if $\frac{n-1}{n}$ of the points are at a distance greater than λ from p_o and hence this definition naturally generalizes the notions of outliers based on statistical tests of significance [96]. Next we state a result that gives us a method to remove λ-outliers. More specifically we assume that, for every point we can compute an *outlier score*, such that when we remove the point with the largest score, we remove the λ-outlier. We also assume that the

distribution of the points is such that after removal of the λ-outlier, another point becomes the λ-outlier with respect to p^*. Thus if we continue this process iteratively, we will be left with p^*. Given a point p_s, its average distance is defined as $\mu_{p_s} = \frac{\sum_{p_s \neq p \in X} d_X(p_s,p)}{|X|-1}$ where $|X| = n$. Next we claim that μ_{p_s} can be used as the *outlier score*. Please note that we defer proofs of all theorems and lemmas to a journal version of this work for space constraints.

Lemma 4.5.1 *Suppose that we have a n point metric space $M = (X, d_X)$, where the points in the metric correspond to sources. Let us suppose that there is $p^* \in X$ representing the unknown truthful source and the rest of the $n-1$ points represent other sources of information. Let p_o be a λ-outlier. Then we have the following:*

- *For any point $p \in B(p^*, \gamma)$, its average distance $\mu_p \leq ((1 - \frac{1}{n})\gamma + \frac{\lambda}{n})$*
- *For the λ-outlier p_o its average distance is $\mu_{p_o} \leq ((1 - \frac{1}{n})\lambda + \frac{\gamma}{n})$*
- *For the above two results it follows that for every $p \in B(p^*, \gamma)$ we have $\mu_p < \mu_{p_o}$*

Algorithm 2 uses lemma 4.5.1 to remove λ-outliers.

Algorithm 2 Outputs the most reliable source

Require: **X**: Points representing sources in metric space

```
1: function TruthCoreAvg(X)
2:     n ← Size(X)                          ▷ n: total number of sources
3:     while n ≥ 3 do
4:         m ← AverageDist(X)
5:         x ← Max(m)
6:         X ←  X - x
7:         n ← n-1
8:     end while
9:     t ← Select(X)
10:    return t
11: end function
```

Algorithm 2 operates by computing the average distance of all the points in X and then removing the point with the maximum average distance. The correctness of the algorithm follows from 4.5.1. The pseudo-code 2 uses two subroutines:

- AverageDist($\mathbf{X}$): Computes the average distance of all the points in X
- Select($\mathbf{X}$): Given a set with two elements, it returns a single element of the set based on some criteria.

We note that the running time of the algorithm is cubic in the size of X. Now we describe another truth finding algorithm using Euclidean embedding, with the goal of improving the running time.

4.5.2 Metric Embedding

Metric embedding studies methods for embedding metric spaces into normed spaces in a way that the original distances are preserved. This is known as *isometric* embedding. However, it can easily be shown that this is not always possible [97]. What we can hope for, however, is an embedding such that the distortion of the distances is minimized. Let $||\cdot||_p$ denote the L_p-norm. Formally:

Definition 4.5.2 (Metric Embedding) *Given a metric space M and a d-dimensional real space with the L_p- norm, denoted by $\mathbb{R}_p^d$, an embedding of M into $\mathbb{R}_p^d$ is an* injective *mapping from M into $\mathbb{R}_p^d$.*

We are interested in methods for computing Euclidean embeddings ($p = 2$) such that for any pair $x_i, x_j \in X$, $||f(x_i) - f(x_j)||_p \geq d_X(x_i, x_j)$. The distortion is defined as $max_{x_i,x_j \in X} \frac{||f(x_i)-f(x_j)||_p}{d_X(x_i,x_j)}$ and we always want embeddings with the minimum possible distortion. There are several theoretical guarantees that are available for the quality of the metric embeddings [97]. Low distortion embeddings can be easily computed using Semi-Definite programs [98]. SDPs can be solved in poly time [99] but they don't scale well and hence are not practical. We note that the embedding computed from the SDP will have a distortion that is bounded above by $\kappa \cdot \log n$ [97] for some constant κ but in general the actual achievable distortion will be much less. We use Multi-Dimensional Scaling [100] for computing the Euclidean embedding. Standard MDS has a complexity of $O(n^3)$ but there are variations with good approximation guarantees that can be computed in $O(n \log n)$ time [101]. In our case n is small which means that maximum distortion is always small. However, even if the

distortion is small, there is a possibility that points that are outliers in M, are no longer outliers in the embedded Euclidean space. It has been empirically observed that MDS preserves outliers better than other dimensionality reduction techniques [102, 103]. Results from our experiments also support this fact. Next we state a result that gives the conditions under which a λ-outlier is preserved after using an embedding with $O(\log n)$ distortion.

Theorem 4.5.2 *Consider the metric space M. Let $p^* \in X$ represent the unknown truthful source. Let p_o be a λ-outlier. Let $\min_{p \in B(p^*,\gamma)} d_X(p, p_o) = l$. Let f be a metric embedding with maximum distortion $O(\log n)$. In order for the λ-outlier to be preserved under f l must be at least $\lambda + \gamma \cdot (\eta \cdot \log n - 1)$ for some constant $\eta > 0$.*

Now we describe an outlier removal algorithm in the Euclidean space that assumes that the points representing values reported by sources are sampled from an underlying distribution P. This algorithm is guaranteed to remove a certain number of outliers and leave an outlier free restriction to the underlying distribution with high probability. It was first used by Dunagan et al. [104].

4.5.3 Outlier Removal in Euclidean Space

In one dimension, a point is an outlier if its distance from the mean is greater than a constant times the standard deviation.When the dimension d ≥ 2, assuming that the origin is centered at the mean, we define an outlier as follows [105]:

Definition 4.5.3 (α-outlier) *A point $\mathbf{p} \in X \subseteq \mathbb{R}^d$ is an α-outlier if there exists some direction $\mathbf{w}$ in which the squared distance of the point from the mean along $\mathbf{w}$ is greater than α times the average squared distance of all the points in X from the mean along $\mathbf{w}$. In other words: $(\mathbf{w}^T\mathbf{p})^2 > \alpha \mathbb{E}_{\mathbf{x} \in X} \left[(\mathbf{w}^T\mathbf{x})^2\right]$*

Hence there are infinitely many directions along which a point can be an outlier and thus the problem of outlier removal in high dimensions is inherently intractable. To overcome this, we place the points in isotropic position [106]. Given a probability distribution P and the ability to sample from it we can put the distribution in isotropic position as follows [107]:

Theorem 4.5.3 *[107] Suppose P is a probability distribution with the ability to sample from it. Let $\mathbf{x} = (\mathbf{x_1}, \ldots, \mathbf{x_n})$ denote a sample drawn from P. Let $\mathbf{M} = Cov(\mathbf{x})$. If $\mathbf{M}$ is positive definite then:*

- *There exists a matrix $\mathbf{T}$ such that $\mathbf{M} = \mathbf{T}^2$*
- *Define $\mathbf{y} = \mathbf{T}^{-1}\mathbf{x}$. This transformation preserves the α-outliers*
- *The distribution $\mathbf{y} = \mathbf{T}^{-1}\mathbf{x}$ is in isotropic position*

The result also holds if the matrix $\mathbf{M}$ is positive semi-definite and the distribution P is rounded to the span of $\mathbf{M}$. This result gives us a way to remove outliers without searching along all directions w.

Lemma 4.5.4 *Given an isotropic distribution, any point $\mathbf{x}$ that is an outlier for some direction w is also an outlier in the direction $\mathbf{x}$*

Thus given a distribution in isotropic position, it is easy to identify and remove outliers. For the samples from P in isotropic position, the α-outliers are exactly the points that satisfy $(\mathbf{p}^T\mathbf{p})^2 > \alpha$. Dunagan et al. [104] proved that under certain conditions on the underlying distribution, given $\epsilon > 0$ there exists $\alpha = \tilde{O}(\frac{n}{\epsilon})$ such that after removing the α-outliers we are left with an outlier free restriction of the original distribution with $1 - \epsilon$ of the probability mass of the original distribution. For our problem, instead of removing the α-outliers at every step, we remove the max outliers and continue to do this iteratively till we are left with just two points. We are now ready to formally describe the algorithm. The pseudo-code for the algorithm is shown in listing 3.

The different subroutines that are used in the algorithm are described below:

- Isotropic($\mathbf{S}$): Given a sample from a distribution P, this subroutine puts the distribution in isotropic position.
- Outlier($\mathbf{M}$): Returns the point with the maximum norm
- Select($\mathbf{S}$): Given a set with two elements it returns the single element with the highest estimated source confidence

Algorithm 3 Outputs the most reliable source

Require: **S**: Points representing sources in Euclidean space

```
1: function TruthCore(S)
2:     n ← Size(S)                          ▷ n total number of sources
3:     while n ≥ 3 do
4:         M ← Isotropic(S)
5:         x ← Outlier(M)                   ▷ Point with largest norm
6:         S ← S - { x }
7:         n ← n-1
8:     end while
9:     t ← Select(S)
10:    return t
11: end function
```

Complexity. The time to compute the covariance matrix once from scratch is $O(nd^2)$, the time to round the distribution once from scratch is $O(d^3 + nd^2)$. The time to find and remove an outlier is $O(nd)$. Finally the time to re-round the resulting distribution through error propagation is $O(nd)$ which is done n times. Thus this gives us a total running time of $O(n^2d + nd^2 + d^3)$. For $d = O(1)$ we get a complexity of $O(n^2)$.

The crux of this algorithm is the fact that it gives us a way of estimating the variance of an arbitrary probability distribution P with high probability, provided we can draw representative samples from P. Next we state a result from [104] which establishes this fact. Let γ be an arbitrary constant and $\delta \in (0, 1/4]$ be a quality parameter and suppose that the distribution P is centered at the origin. As before let $\mathbf{x} = (\mathbf{x_1}, \ldots, \mathbf{x_n})$ denote a sample drawn from P. $\mathbf{M} = Cov(\mathbf{x})$ and let us assume that $\mathbf{M}$ is positive definite. Then by theorem 4.5.3 $\mathbf{M} = \mathbf{T}^2$. Let us write $E(M) = \{\mathbf{x} : \mathbf{T}^{-1}\mathbf{x} \leq 1\}$. $E(M)$ is called the primal inertial ellipsoid [107].

Theorem 4.5.5 *[104] Suppose that we draw $n = \tilde{O}(d\frac{\gamma^2}{\delta^2})$ samples from the distribution P. Let $\Gamma^2 = (1+\delta)^2\gamma^2$. Let $\tilde{M}$ be the covariance after removing all the Γ^2- outliers using Algorithm 3. Then with high probability, $P((1+\delta)\Gamma E(\tilde{M})) \geq 1-\epsilon$; $\epsilon > 0$ and $(1+\delta)\Gamma E(\tilde{M})$ does not contain any $(1+\delta)^{O(1)}\gamma^2$ outliers.*

The above result says that as we continue to remove the Γ^2-outliers, there comes a time when the distribution is rounded to the point that no further rounding is necessary. Dunagan el at. [104] also showed that for $\alpha = \tilde{O}(\frac{d}{\epsilon})$, the points in the α-outlier free set will be such that, along any direction w, their distance from the mean will be at most $\sqrt{\alpha}$ times the standard deviation along w. This in turn shows that it is possible to find a large subset of the original probability distribution where no point is too many standard deviations away from the mean (location parameter) [104]. Thus if we model the sources as reporting values from an underlying probability distribution with the truth at the location parameter, then the most reliable source is the one that is close to the location parameter. By the above results this algorithm will eventually converge to this source. If there are several such sources, then it will converge to one of them.

4.6 Experimental Setup

We ran several experiments on a variety of datasets to test the proposed algorithms. Since TruthCore can be applied to only one attribute at a time, in case the item has multiple attributes, truth finding should be preceded by the task of schema matching. We performed the experiments on a workstation with eight Intel(R) Core(TM) i7-2600 CPU @ 3.40GHz processors with 15.7 GB of RAM. Each experiment used only a single core. We used Python 3 to implement the algorithms. For Schema Matching, we used LIBSVM's [108] implementation of Support Vector Machines. We selected the RBF kernel and performed exhaustive grid search with 10-fold cross-validation to determine the optimal parameter values. The accuracy was measured by the percentage of the predicted values which are within a fixed interval of the ground truth. For Schema Matching, we simply compared the predicted matchings with the manually verified matchings for the accuracy.

4.7 Schema Matching Experiments

To test the algorithm, we need a dataset containing information about multi-attribute items obtained from multiple sources. The schemas provided by the sources are expected

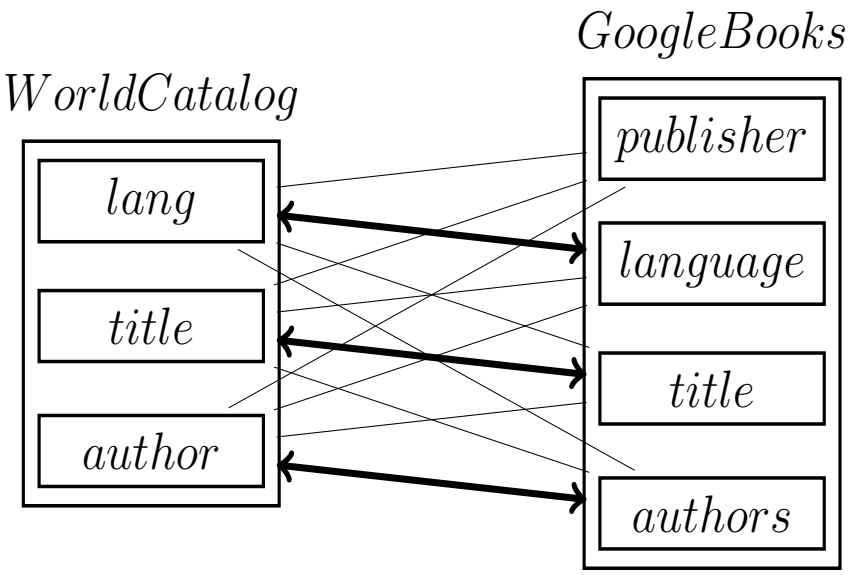


Figure 4.1: Edges of the complete bipartite graph correspond to all possible mappings between attribute names provided by WC and GB for ISBN 9781402773259. Only three edges (drawn as thick arrows) out of the twelve are the valid mappings.

to be different. Given the dataset and a pair of sources, the algorithm outputs a set of mappings between the schemas of the two sources. We compare these mappings with the Gold Standard and as we have already discussed in Section 4.4, in case of a multi-attribute object, truth finding should be preceded by the task of schema matching. So, the type of data determines if schema matching is required.

4.7.1 Dataset

Table 4.1: Conflicts in schemas. Attribute names provided by online book metadata sources.

GoogleBooks	**WorldCatalog**	**OpenLibrary**	**ISBNdb**
authors	author	authors	author_data
title	title	title	title
language	lang	languages	language
publishDate	year	publish_date	NA
publisher	publisher	publishers	publisher_name
pageCount	NA	number_of_pages	physical_ description _text

For testing the algorithm, we created a dataset that contains book metadata for 190 ISBNs obtained from four online sources, namely *Google Books (GB), Open Library (OL), World Catalog (WC) and ISBNdb (IDB)*. The selected sources provide APIs that can be queried with a given ISBN for obtaining the data. The data from each source is obtained as a comma-delimited key/value pair, called JSON. Sometimes the JSON object is nested. Apart from flattening the nested data, we do not perform any other modifications to the keys or the values. Each book record has a multi-attribute schema with the ISBN as the primary key. The attributes include title, author names, date of publication and publisher information among others. Different sources name these attributes differently and hence the underlying schemas from them are different (Table 4.1). Some attribute names are exact, for example, "authors" in GB and OL while some are significantly different ("publishedDate" and "year" in GB and WC respectively). In addition to the conflicts in the attribute names, the sources also provide information for different number of attributes, for different ISBNs (Table 4.2).

Ground Truth. In order to create ground truth data for training and cross validation, we sample a subset of the available 190 ISBNs and manually map and create pairwise matchings between every source pair. We call this the Gold Standard. If we take attribute k_i from source i with k_j from source j and if v_{k_i} and v_{k_j} are the corresponding attribute values as strings, we apply four string similarity functions between v_{k_i} and v_{k_j}, namely, *edit distance*, *TF-IDF similarity*, *exact similarity* and *trigram similarity* which gives a vector of 4 values. This transforms an observation of the considered attribute pair between two sources to a 4-dimensional space. If k_i and k_j are verified to be a matching, we give the vector a class label +1 since it is a valid matching. Otherwise, we give -1 as the class label.

The schema matching algorithm described in Section 4.4 is a supervised learning scheme and requires training and test samples. Each sample corresponds to a labeled edge weight as we described above. There is a large difference in the number of valid and invalid matchings in the training set (Figure 4.1). The total number of training samples for a pair of sources corresponds to the number of edges in the bipartite graph for each ISBN. It can be estimated as the product of the total number of ISBNs and the average number of attributes in each

source. For example, for IDB and GB as the training pair and with 60 ISBNs between them, approximately $60 * 21 * 17 = 21420$ samples are generated in the training set. Since the total number of valid mappings between attributes in two different schemas S_1, S_2 is always $\leq min(|S_1|, |S_2|)$ and the total number of possible mappings is $|S_1| . |S_2|$, we observe only 2% as +1 samples and the rest (almost 98%) as -1 samples in the generated training set. The test set also has similar properties.

Table 4.2: Maximum, minimum and average number of attributes given by the four sources for 190 ISBNs in the Schema Matching Dataset.

Source	max	min	average
IDB	21	21	21
GB	22	13	17
OL	100	11	24
WC	13	1	10

4.7.2 Results and Discussions

We list the results of the schema matching experiments in Table 4.3. It shows a comparison of the number of valid mappings the algorithm was able to predict with the number of matchings we were able to ascertain manually. For (OL, IDB) and (GB, IDB) the algorithm picks 5 and 3 additional valid matchings respectively, compared to the ground truth (See Table 4.5). These matchings are valid because they refer to the same aspect of the physical description of the book but are difficult for humans to notice and were also not represented in the training set. For example, since number of pages in a book is a part of its physical de-

Table 4.3: Schema matching results. 60 ISBNS used for training and 130 used for testing.

Train	**Test**	**Predicted**	**Manual**
GB, OL	WC, IDB	5	6
GB, WC	OL, IDB	12	7
GB, IDB	OL, WC	9	9
OL, WC	GB, IDB	9	6
OL, IDB	GB, WC	5	6
WC, IDB	OL, GB	8	10

Table 4.4: Missed matchings for listed test source pair

Test	**Missed**
WC, IDB	$lccn \Leftrightarrow lcc\_number$
GB, WC	$language \Leftrightarrow lang$
OL, GB	$description \Leftrightarrow description$
	$genres \Leftrightarrow categories$

Table 4.5: Discovered matchings for listed test source pair

Test	**Discovered**
OL, IDB	$physical\_format \Leftrightarrow edition\_info$
	$subtitle \Leftrightarrow title\_long$
	$number\_of\_pages \Leftrightarrow physical\_description\_text$
	$full\_title \Leftrightarrow title\_latin$
	$publish\_places \Leftrightarrow publisher\_text$
GB, IDB	$subtitle \Leftrightarrow title\_long$
	$pageCount \Leftrightarrow physical\_description\_text$
	$publishedDate \Leftrightarrow edition\_info$

scription, we can assume the mapping between "pageCount" and "physical_description_text" to be valid. We were unable to predict these extra matchings manually because the attribute names were too disparate. Moreover, these attributes were not present in the metadata of the ISBNs which were sampled while creating the Gold Standard. Thus, the algorithm has two advantages over manual matchings. It not only automates Schema Matching but also helps in discovering additional matchings which can be left out while doing it manually. The algorithm also fails to pick some of the valid matchings as shown in (Table 4.4). For the source pairs (WC, IDB), (GB, WC) and (OL, GB), the number of missed matchings are 1, 1 and 2 respectively. The easily noticeable matching from "language" to "lang" between GB and WC is missed because the two sources hold different strings for the same language information. For example, for a book in English, GB has "en" whereas WC has "eng". Thus, in case the same attribute content exists as different strings in two schemas, the algorithm may fail to pick that matching. An interesting case is the missed matching between "lccn" and "lcc_number" for WC and IDB shown in Table 4.4. The "lccn" in WC corresponds to *Library of Congress Control Number* whereas the "lcc_number" in IDB corresponds to

Library of Congress Call Number and there should not be any matching between them. For OL and GB as the test pair, the algorithm fails to pick the following mappings: ("description", "description") and ("genres", "categories"). This is due to the sparsity of the dataset. Out of 190 ISBNs, OL has less than 10 records that contain the attributes "genres" and "description". Moreover, the values of these attributes provided by OL differ significantly from those provided by GB. This also shows that manual schema matching can be prone to errors.

Certain sources can make a dataset difficult by providing composite attributes. These attributes are conglomerations of several atomic attributes. For example, a source can merge multiple attribute values to obtain one single attribute. "physical_description_text" in IDB includes information like physical dimensions, physical weight and number of pages which are given separate attribute names in OL. This adds difficulties even during manual matchings. The natural solution to this is to split the attribute into its subcomponents but automatic detection and unpacking of such merged attributes is itself a hard problem.

4.8 Truth Finding Experiments

We applied our truth finding algorithm on two datasets, the book author dataset and the algorithms quiz dataset. We discuss the datasets, the experiments and the results in detail below.

4.8.1 Book Author Dataset

We create a new dataset which is similar to the Book Author Dataset used in [109]. Since TruthCore is designed to process data obtained only from well-known and reliable sources, we select Google Books, Open Library, World Catalog, ISBNdb and Amazon as our authoritative sources. We obtain author information for 130 ISBNs from these 5 sources. Our Gold Standard contains manually extracted author names from the book covers and includes the original Gold Standard used in [109].

String Alignment and Thresholding for Evaluation. A book can have multiple authors and the sources can provide author names in different order. Even for a single

author, some sources can report different versions of the author's name. For example, the sources can include nicknames (*Bob* for *Robert*), abbreviations of first or middle names (*D. Knuth* for *Donald Knuth*) or titles (*Mr., Ms., Dr.*). Moreover, the strings containing author names can also have words like *by, etc, with, authored* which are not part of any name. Since our algorithm relies on the similarity between the reported values, these values need to be pre-processed to remove any ambiguities. To this end we first break down the strings into tokens, then filter out the tokens which are in the set of unnecessary words. Finally, we apply Kuhn-Munkres Algorithm [110] to align the two strings.

TruthCore, by design, may not produce the exact answer as the output but will give an output in the vicinity of the expected answer. For this purpose, we introduced the idea of a threshold ϵ. We fix ϵ to be equal to 3. The output of TruthCore is considered to be a match if the edit distance with its corresponding ground truth is 3 or less. We use normalized edit distance to compute the MAE. The string representing an author name is first aligned with the string provided by the Gold Standard and it is considered to be a match if the edit distance between the two is ϵ. We compute accuracy as the percentage of the predicted values which are within an edit distance of 3 from their ground truths after filtering and alignment. This method has been used for calculating the accuracy reported below.

4.8.2 Algorithm Quiz Dataset

This dataset was collected during a quiz taken in one of the classes of Advanced Algorithms (COT5405) offered in Fall 2015 by Dr. Piyush Kumar at Florida State University. It contains 30 questions related to the course. 22 students present in the class took the quiz.

Sources are represented by students and each source has an arbitrary label to maintain anonymity. Some of the questions were left blank and we replaced the missing answers to these questions with a default string (character "X" repeated 40 times) to make it highly dissimilar to any other answer. This is chosen as it is most likely to be highly dissimilar to any other normal answer and the distance between the two will be large. This in turn will make the likelihood of this value being treated as an outlier high. An expert marked all the answers as either correct or incorrect. Because of the nature of the quiz, though there is only

Table 4.6: Samples of correct answers to question Q_1, Q_2, Q_9 provided by the students in the Algorithm Quiz Dataset.

Key	**Question**	**Correct answer samples**
Q_1	Which sorting algorithm will you use if you are asked to sort an array of integers which are within the range $(1, 1000)$?	Radix Sort
		Counting sort
		Bucket Sort, use an array $[1000]$ to sort the integers
		count sort
		(first thought is Quicksort), but with constraint use bucket like sort, with 1000 buckets/array elements)(i could do in O(N),just ask how)
Q2	How fast can the median of n numbers be determined?	O(n)
		O(n), using like quick sort approach
		linear time
		linear time: O(n)
Q9	State the max flow - min cut theorem.	finding the maximum flow of graph G corresponds to finding the min cut
		max flow = min cut
		This theorem says this two problems are equal which means by finding the max flow we already have found the min cut and vice versa
		If Sum (e out of s (f(e)) - sum e in to s (f(e)) = value of flow and v(f) = cap(A, B), then the flow is maximum and the cut is minimum.

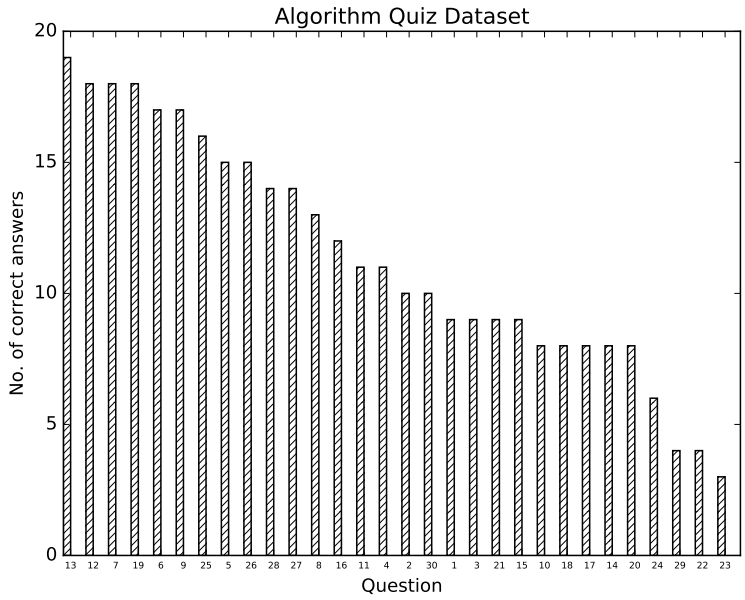


Figure 4.2: Number of correct answers for each question in the quiz. Questions 22, 23 and 29 have the least number of correct

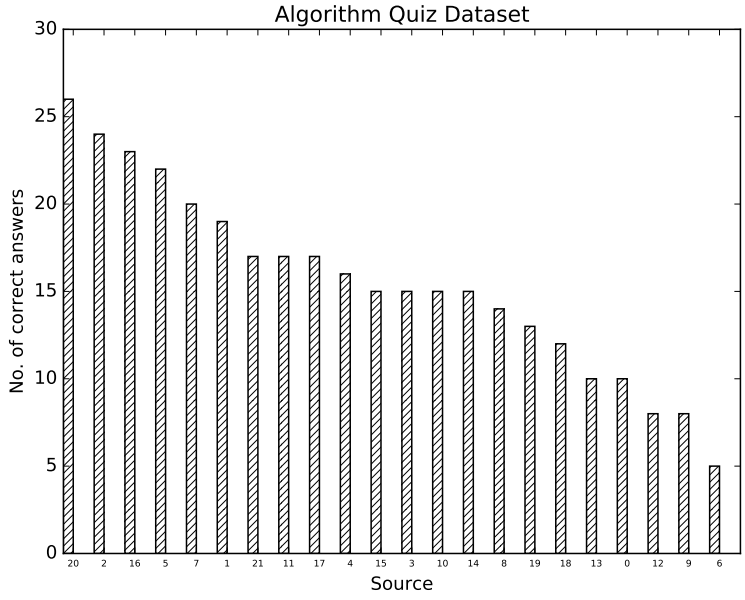


Figure 4.3: Number of correct answers given by each source in the quiz.

Table 4.7: Source confidence computed from the ground truth for Book Author Dataset

Source	Average Confidence
AMZ	0.92
OL	0.85
WC	0.85
IDB	0.82
GB	0.74

one correct answer to a question, there are several different ways to write the same. We put all the possible ways of representing the correct answers to a question in a single group. We used this group to verify the correctness of the answer generated by TruthCore.

An analysis of this dataset shows that the answers are very unstructured and diverse. This makes this dataset interesting and really hard to process automatically. Table 4.6 gives a glimpse of the diversity in the answers. There are questions (*Q1*) which can have multiple correct answers. For some (*Q2*), the same answer can either be described using words or using mathematical expressions and for others (*Q9*), answers can either be concise or verbose. All of these natural/mathematical language elements, syntactic differences and other factors present in the answers makes it harder for the algorithm to find the correct and most appropriate answer. Only 13 questions are answered correctly by the majority, whereas, for the remaining 17, only a minority of the sources give the correct answers (Fig 4.3). Question numbers 23 and 13 have the minimum (3) and maximum (19) number of correct answers respectively. These aspects of the dataset make the task of truth finding difficult.

For all the experiments we compare TruthCore with two baseline methods for truth finding, namely, Random Selection and Voting. In Random Selection, the output is picked at random. In Naive Voting, the most frequently occurring value is chosen as the output.

4.8.3 Results and Discussions

For both datasets, experiments with different string similarity metrics reveal that the choice of a particular similarity metric does not have an impact on the performance of the algorithm (Table 4.8). The average accuracies obtained for all of the distance metrics

Table 4.8: Average and best accuracies for TruthCore with different distance measures in the Book Author Dataset. Though it shows that jaro metric gives the best result, others are not too far away.

Distance	**Average Accuracy (%)**	**Best Accuracy (%)**
dicesorensen	88.3	91.0
jaccard	87.9	93.0
jaro	88.2	93.0
jarowinkler	88.9	92.0
levenshtein	88.2	92.0
ngram	86.4	91.0
ratcliffobershelp	88.1	91.0

Table 4.9: Comparison of TruthCore with the baselines for Book Author Dataset

Method	**Accuracy in %**	**MAE**
Random Selection	79.84	0.1231
Naive Voting	80.23	0.1178
TruthCore (Average Dist)	93.76	0.0337
TruthCore (MDS)	94.61	0.0285

Table 4.10: Effect of choice of sources for truth finding on the Book Author Dataset.

No. of sources	**Sources**	**Accuracy in %**
3	WC, OL, AMZ	93.07
	GB, IDB, WC	85.38
4	WC, OL, AMZ, IDB	93.84
	GB, WC, IDB, OL	86.15
5	WC, OL, AMZ, IDB, GB	94.61

are close to 94%. This could be the result of preprocessing applied on the pair of strings before we compute the similarity. In all likelihood, the pre-processing makes sure that the idiosyncrasies between the different strings representing the author names are removed. As a result any string similarity metric can correctly capture the similarities or differences. Thus we used edit distance as the default string metric and the results for the book author data are shown in Table 4.9.

We observe that TruthCore outperforms both the baselines by a large margin. TruthCore takes into account the distances among the strings containing the author names whereas Voting only considers the frequency of occurrence of each string. Thus, TruthCore is better at estimating the central tendency of the input answers. We also see that the performance of random selection algorithm is comparable to that of Voting. The reason behind it is that even though the input strings contain the same author information, a minor difference ends up making them unique strings. When voting is done on unique strings, each string ends up having a single vote and random selection is required to break the ties. Thus, in essence, voting becomes a random selection procedure.

We also studied the effect of source confidence on TruthCore. We first compute the source confidence for all the five sources. *Source confidence* of a source is the percentage of books for which the source provides author names correctly. Table 4.7 shows the sources ranked according to their confidences and Table 4.10 shows how the presence of high confidence sources positively affects its performance. We get the best average result (93.07%) when the top three sources (AMZ, OL, WC) are included. We get the worst performance (85.38%) if we include only the three worst sources (WC, IDB, GB). Either the inclusion of a low confidence source (GB) or the exclusion of a high confidence source (AMZ) is detrimental to the performance of "TruthCore". This is because of the fact that "TruthCore" is essentially performing a non-parametric estimation of the location parameter of an unknown distribution. Inclusion of sources with higher confidences means that we are using statistics that have lower MSE and hence "TruthCore" will select these as its answer, thereby reducing the error. On the other hand if the statistics are not good in terms of their MSE, then

Table 4.11: Comparison of different versions of TruthCore with the baselines for Algorithm Quiz Dataset.

Method	Accuracy in %
Random Selection	45.00
Naive Voting	45.00
TruthCore (jaccard)	63.33
TruthCore (ngram)	64.00
TruthCore (jarowinkler)	70.00
TruthCore (jaro)	71.00

"TruthCore" can only do as good as the statistics included and hence its accuracy will be adversely affected.

We also observe that, in the best case, TruthCore is able to select correct answers for 23 questions, which is better than any single student in the class. Out of these 23 questions, the student majority was correct for 12 questions and the minority was correct for the remaining 11. With an appropriate distance measure, TruthCore does not miss picking the correct answers when the majority of the sources are correct. Even when the majority of the sources are providing incorrect answers, it was able to pick the correct answer for a majority of the questions. Thus, with an appropriate metric that can handle the answer diversity, "TruthCore" can be incorporated into a semiautomatic grading system to assist a human grader.

4.9 Automating Open Library Catalog Edits

In this section we describe our attempt to fix records in Open Library [111], a very large database of more than 20 million book metadata, using TruthCore. Given a book in the database, we query other reliable sources and gather data about the book. Then we apply TruthCore to this collection of records. If the output of TruthCore is different from the record in the database, the record is likely to be incorrect. In such a case, the record can be replaced with the output of TruthCore.

Table 4.12: Types of edits performed by TruthCore on 50 randomly selected edits on author names while correcting Open Library Catalog.

Edit type	Count
Provide complete author names	38
Provide subset of author names	8
Remove titles from author names	1
Select additional contributors	1
Remove repeated author names	1
Provide all authors but repeat one name	1

4.9.1 Dataset

We downloaded the publicly available Open Library data from [112]. We selected book records for 139996 ISBNs such that GoogleBooks (GB) and World Catalog (WC) contained metadata for these ISBNs. We query GB and WC and create a local database for these ISBNs obtained from three sources: GB, WC, OL. We consider the "author name" attribute. Out of the 139996 ISBNs, OL doesn't have any author information for 14098 ISBNs.

We created the ground truth for 50 randomly selected ISBNs with the data from Florida State University Libraries Catalog Services, manual searches of the books we own, and from web searches of book cover images.

4.9.2 Results and Observations

Out of 139996 records, TruthCore attempts to fix 37011 (26.4%) OL author names. To take a closer look at the quality of edits, we selected a subset of these for which we had the Ground Truth and verified the correctness manually. The catalog service of Florida State University Libraries had records for only 196 out of 37011 edits. We select 50 edits at random from this verifiable set of 196 edits.

Table 4.12 shows the types of edits performed by TruthCore. A total of 42 edits provided the correct list of author names for the books although with minor changes for some. These changes do not alter the core content and the reader is still able to identify the actual author names. For example, for ISBN *9782742782222*, OL's initial *"Daum, Pierre journalist"* was replaced with *"Pierre Daum"*. Here, a word denoting the author's title is dropped. It

also selected other contributors to the book in addition to its authors. OL has *"Carole McNamara"* for ISBN *"9781555953256"* which was edited to *"Carole McNamara ; with essays by Sylvie Aubenas ... [et al.]."*.

Moreover, out of the 50 randomly selected edits, OL had no information for 23 ISBNs. 19 of these missing values were correctly edited by TruthCore. However, there were 8 incorrect edits (Table 4.12). We consider the edits to be wrong when TruthCore picks only a subset of the author names. A closer look at the incorrect edits revealed that none of the three sources (OL, GB, WC) had complete author information for 6 of those ISBNs.

For example, for ISBN *9781841272573*, OL provides none and GB and WC both provide 2 author names (*"J. Andrew Dearman, M. Patrick Graham"*). The book has 3 author names (*"J. Andrew Dearman; M. Patrick Graham; Miller, J. Maxwell"*) listed on its cover. For the other 2 edits, only one of the three sources provides the correct information. For example, OL has no author information for *9781584889731*, WC provides a single author name *"Motoichi Ohtsu"* and GB correctly provides all five *"Motoichi Ohtsu, Kiyoshi Kobayashi, Tadashi Kawazoe, Takashi Yatsui, Makoto Naruse"*. TruthCore fails to pick the correct answer when none of the sources provide the correct answer or when a majority of the sources provide wrong answers.

The results show that we can apply TruthCore in a crowd sourced application with a huge database like Open Library to detect and fix incorrect or incomplete records automatically. If correctness cannot be guaranteed or verified for certain edits, the corresponding records can be flagged so that the concerned authority can have a closer look at them manually.

4.10 Conclusion and Future Work

In this work we have described two algorithms for truth finding from a collection of authoritative sources. The first works in a metric space and the second in Euclidean space. The results of our experiments show that the algorithms can be used for determining the truth from very large databases like the Open Library catalog as well as esoteric datasets like the algorithms quiz data. Moreover the algorithms are agnostic to the data type, which is a major advantage. Going forward, we would like to expand this study along two directions:

first we would like to apply these algorithms on larger datasets and study the results. As the algorithms used in the paper have drawn their inspirations from the study of combinatorial optimization algorithms, we would also like to create a rigorous theoretical framework for explaining truth finding algorithms from the perspective of combinatorial optimization, in the hope of getting stronger theoretical guarantees.

CHAPTER 5

PYTHON APPLICATION PROGRAMMING BOOK

5.1 Introduction

The last five years have seen an exponential growth of the Python programming language [113]. It was the second most popular language on github in 2017 based on the number of pull requests [114]. As of July 4, 2018, almost 25% of the job advertisements on StackOverflow look for Python programming skills [1]. Further, there is a surge in cross-disciplinary machine learning applications like medical imaging, Large Hadron Collider data analytics [115], bioinformatics, protein structure prediction and others. Researchers and programmers tend to use Python for these tasks due to the availability of libraries like scikit-learn [2], scikit-image [3], keras [4], PyROOT [115] and others. These are easy-to-use, documentation-rich libraries with user-friendly APIs.

There are other important factors behind the popularity of Python with beginners [116]. Python is free and can run on any system that has a C compiler. So, anybody can use it without worrying about licensing. With Python, less time is spent on understanding the tooling system before running the first ever Python program (say, a "Hello World!" program). In case of C or C++, learners may lose their enthusiasm while trying to understand program compilation, "#include" statements, main() function and others before they can even print "Hello World!". Due to this low learning curve, Python allows the first time learners to directly dive into interactive programming. Python language itself is known for its clean and readable syntax which is advantageous for beginners. Python also has a vibrant

[1] https://stackoverflow.com/jobs/developer-jobs-using-python
[2] https://scikit-learn.org
[3] https://scikit-image.org
[4] https://keras.io/

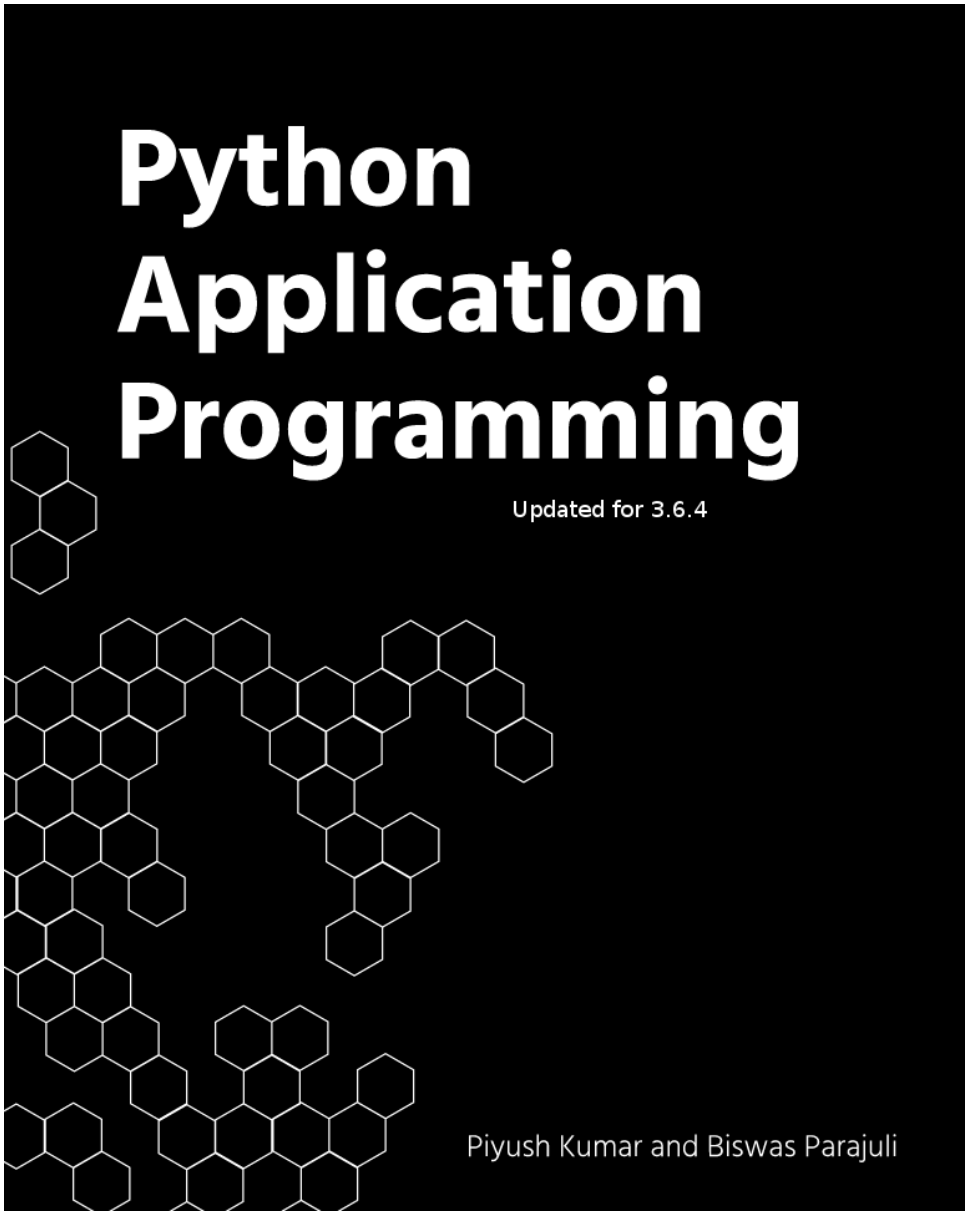


Figure 5.1: Cover page of our book on Python.

community and the web is rich in learning resources, tutorials and free books. Moreover, many universities are offering introductory programming courses based on Python [117].

We also used Python quite extensively to implement our research projects that deal with machine learning concerning computer vision. It was our go-to language due to the availability of advanced, up-to-date, well-documented machine learning and computer vision libraries. In the course of implementing the projects, we ourselves learnt about how to learn Python. Although there already existed power point slides for the Python programming course offered by the CS department at Florida State University, we compiled additional Python materials that reflected our learning process. Eventually, our efforts culminated into a book titled "Python Application Programming" which contains materials refined through years of teaching, research experiences and feedback received from students.

5.2 Goal and Requirements

Our goal is to provide an efficient learning trajectory to beginners as well as intermediates of Python programming. Books on programming usually tend to be verbose. In our book, we take a non-traditional approach to learning Python. We try to strike a balance between verbosity and pace of learning. We summarize the important features of the language with self-explanatory examples while keeping descriptions to a minimum. We customize the content such that readers who have prior programming knowledge but are new to Python will find the book easy to read while experienced programmers might find it useful in preparing for programming interviews. We had our own set of requirements while writing this book. We also had to deal with a few constraints which resulted in need of additional infrastructure.

5.2.1 Online and Print

We aimed for two versions of the book, namely, online and print. To save time and for efficiency, we enforced a single point of edit from which multiple versions of the book can be generated. When we started out with this project, Jupyter Notebook wasn't yet known. So, we decided to edit the book content in HTML and later convert it to PDF. Initially, we wrote a script which renders the book in a browser and takes a snapshot for every page. We

wanted to combine these snapshots to get a pdf file but it didn't give us more control over the layout. Thus, we decided to write a custom translator which parses the HTML and generates Latex. This way, we were able to fine tune how each HTML element should be rendered in a pdf file. We put the HTML version of the book on the web at `https://pybook.rocks`.

5.2.2 Automate Tests of Snippets

The book is interspersed with Python scripts which should be properly vetted. For this, we wrote a suite of unit tests and a script which automatically shows the failing scripts. This way we can pin point the problematic scripts fast.

5.2.3 Catch up with Python's version change

It is a known fact that the version of Python changes quite frequently. The newer versions can have additional features or they might discontinue pre-existing features. Because of this, some of the scripts in the book may not work. Moreover, some scripts display Python versions which also need to be updated. To address these issues, we create a new virtual environment for the current Python and run tests (as described above). We also re-generate the scripts which display the version of Python.

5.3 Book Content

The book contains 10 chapters in total along with their associated exercises. We describe the individual chapters below, in brief:

5.3.1 Chapters

Getting Started. This is an introductory chapter which covers basic tooling for getting started with writing and executing Python programs. Any learner should be able to dive directly into the relevant effort-feedback loop. If a student wastes hours on the internet or by going through bulky documentation while trying to set up a programming environment before even getting a chance to play with the Python language, their motivation can fade

Figure 5.2: Table of contents for chapter *Getting Started*.

away fast. We try to include the right pieces of information for getting started so that the students don't have to look for other resources and their interest remains intact.

We first introduce the Python installation process. Students can have access to and be familiar with computers with a variety of Operating Systems which results in the programming environment and the set up being different. Further, the system related expertise may vary a lot among the students. So, we list the installation procedure for the most commonly used computers, namely, Windows, Linux and Mac.

A basic programming environment includes a code editor with which the students can write the program, a compiler or an interpreter that runs the written program and a file system so that the program can be saved for future usage. To this end, we recommend a bunch of code editors along with their advantages. We give examples of how to use the command line on a Unix-like system. We give a more detailed view of Jupyter Notebook which is a browser based environment and is best suited for students new to Python.

Our aimed readers also include the ones who are already familiar with other programming languages like C, C++ or Java. We describe the visual anatomy of a Python program to introduce them to Python's peculiar syntax. After this, we directly get into the basics of Python programming: syntax, names, loops, conditionals, printing, core data types and others.

Figure 5.3: Table of contents for chapter *Data Types and Operations*.

Data Types and Operations. This chapter introduces the power of Python in the form of simple data types and data structures so that students can move on from simple programs involving simple loops and single valued names to more complex programs that can handle real world data and simulate applications. The covered data types are *int*, *float*, *string*, *None* and others. We show how objects of these simpler data types can be grouped to get a collection of objects. We cover data structures like *tuple* as an ordered but immutable collection, *list* as an ordered collection, *set* as an unordered collection, and *dict* as a collection of objects mapped with unique keys. We also elaborate on the differences between and the properties of mutable and immutable objects since this is one of the common mistakes students make when they start out with Python.

Students first learn the concept behind each data structure. Then, they learn the syntax with the help of included examples. Finally, there are examples that show the usage of a subset of built-in methods which are most frequently used for each data structure. Students with prior programming experience in other languages are shown how a multiline program in C, C++ or Java can be shrunk to a few lines of Python with list comprehension and utilities like *zip()*, *enumerate()*, *range()*, *sorted()* and others.

Functions. The next stage after learning how to write Python statements that work on objects or collections of objects is to learn to group statements that are used repetitively in a program. We start with the syntax of defining functions and the importance of indentation

Figure 5.4: Table of contents for chapter *Functions*.

to define a code-block. We state that in Python, functions are objects pointed to by a name but that can be called to run the statements that lie within their respective code blocks. We show various ways of how objects can be passed into functions known as argument passing. We describe another type of functions, the *lambda* functions, which don't have to be associated with a name but can be created and passed as arguments on the fly. We give examples of these anonymous functions while covering the *key* argument in *sorted()*, *min()* and *max()*.

In addition, we cover scoping rules within functions, exception handling, closures, enforcing checks with *assert* and others. On completing the exercises, students learn to write refactored functions.

Modules and Standard Library. Programs written on the Python interpreter are lost when we close the interpreter. The programs should be saved for future use. Haphazard file locations hamper the search when we want to reuse the saved programs. So, this chapter provides instructions and examples that show how one can save Python programs as modules. It introduces Python's import mechanism for reusing existing Python code as libraries. Students also learn to create Python packages which are directories containing multiple modules. This chapter teaches code re-organization which is extremely important when the program becomes more complex.

Figure 5.5: Table of contents for chapter *Modules and Standard Library*.

Figure 5.6: Table of contents for chapter *Text Processing*.

In this chapter, we also introduce some of the most commonly used Python modules and libraries. Some programs run in the terminal which need commandline arguments. For this, we cover the *sys* module. Some programs need to interact with the operating system interfaces like the filesystem, for example, to read files from a directory or create new files for which we cover the modules like *os*, *glob* and *shutil*. To get the time taken for executing a program or the current date and time, we cover *time* and *datetime* modules. We show access to the shell interface with the *subprocess* module.

Text Processing. This chapter is related to one of the most important applications that mostly arises in research or data analysis. This application oriented chapter shows how

we can process texts that come in different formats, for example, plain text, CSV (Comma Separated Values) format, HTML, XML and JSON, and parse and extract the information we need. We cover modules *lxml* and *XPath* to handle and parse HTML or XML formatted text. For dealing with JSON data, we show examples that use the module *json*.

Often we need to search for patterns in a given dump of text where the text can either be structured or unstructured. We elaborately explain the importance of regular expressions for searching in texts with Python snippets that use the standard *re* module. We also cover databases like *sqlite3* as a means to store processed textual data.

Figure 5.7: Table of contents for chapter *Functional Programming*.

Functional Programming. Here, we demonstrate with examples Python's functional style of programming with functions like "map()", "filter" and "reduce()". We also cover iterators and generators to process container objects while keeping the memory footprint low.

Object Oriented Programming. This chapter introduces "class" and explains object oriented programming in Python. We cover the special methods like "__init__()", class attributes and the differences between bound and unbound methods. Other topics include inheritance, scoping rules within classes and ways of customizing classes so that Python builtins like "len()", "iter()", "str()" can be used on their instances.

Figure 5.8: Table of contents for chapter *Object Oriented Approach*.

Figure 5.9: Table of contents for chapter *Testing, Debugging and Tuning*.

Here we discuss the internals of Python objects, for example, how Python selects the correct class method in case of multiple inheritance, constructs a new object and performs garbage collection.

Testing, Debugging and Tuning. We cover topics related to locating bugs in the program and also ways to profile and make a program run faster. We show how to write custom exception classes to handle non-standard exceptions. The chapter encourages students to breakdown a program into almost-atomic units so that each unit can be tested separately with a test of its own. We show usage of the modules: (1) "doctest" and "unittest" to effectively run the tests (2) "logging" to create large scale logs (3) "pdb" to locate bugs by

Figure 5.10: Table of contents for chapter *Algorithms in Python*.

iterating through the program statements during run-time (4) "cProfile" to get statistics on execution frequency and duration of program components. We also discuss "Cython" which interfaces Python code with external C code to make the code run faster.

Algorithms in Python. This chapter demonstrates that efficient algorithms can outweigh other system related optimization like multiprocessing. We consider the closest pair problem in two-dimensions and compare/visualize time complexities of a grid-based algorithm, a divide-and-conquer algorithm and a brute force solution. We discuss the object-oriented interface of "matplotlib" to plot the run-times for different input sizes. We also show the usage of module "networkx" to handle real world graph structured data.

The first three chapters are aimed towards beginners. Advanced users can use them to refresh their memories. The other chapters are designed in the context of applying Python programming to real world problems. There are other topics that we are planning to cover: (a) matrix operations with numpy and pandas (b) web programming in Python.

REFERENCES

[1] B. Parajuli, P. Kumar, T. Mukherjee, E. Pasiliao, and S. Jambawalikar, "Fusion of aerial lidar and images for road segmentation with deep cnn," in *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2018, pp. 548–551.

[2] G. Máttyus, W. Luo, and R. Urtasun, "Deeproadmapper: Extracting road topology from aerial images," in *International Conference on Computer Vision*, vol. 2, no. 4, 2017.

[3] D. Costea and M. Leordeanu, "Aerial image geolocalization from recognition and matching of roads and intersections," *arXiv preprint arXiv:1605.08323*, 2016.

[4] G. Siegle, J. Geisler, F. Laubenstein, H.-H. Nagel, and G. Struck, "Autonomous driving on a road network," in *Intelligent Vehicles' 92 Symposium., Proceedings of the*. IEEE, 1992, pp. 403–408.

[5] G. Miller, "The huge, unseen operation behind the accuracy of google maps," https://www.wired.com/2014/12/google-maps-ground-truth/.

[6] G. Moody, *OpenStreetMap Should Be a Priority for the Open Source Community*. [Online]. Available: https://www.linuxjournal.com/content/openstreetmap-should-be-priority-open-source-community

[7] F. Bastani, S. He, S. Abbar, M. Alizadeh, H. Balakrishnan, S. Chawla, S. Madden, and D. DeWitt, "Roadtracer: Automatic extraction of road networks from aerial images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4720–4728.

[8] R. Ghurchian, S. Hashino, and E. Nakano, "A fast forest road segmentation for real-time robot self-navigation," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 1, Sept 2004, pp. 406–411 vol.1.

[9] J. Trinder and M. Salah, "Aerial images and lidar data fusion for disaster change detection," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 4, 2012.

[10] N. J. Yuan, Y. Zheng, and X. Xie, "Segmentation of urban areas using road networks," Tech. Rep., July 2012. [Online]. Available: https://www.microsoft.com/en-us/research/publication/segmentation-of-urban-areas-using-road-networks/

[11] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[12] R. Bajcsy and M. Tavakoli, "Computer recognition of roads from satellite pictures," *IEEE Transactions on Systems, Man, and Cybernetics*, no. 9, pp. 623–637, 1976.

[13] M. A. Fischler, J. M. Tenenbaum, and H. C. Wolf, "Detection of roads and linear structures in low-resolution aerial imagery using a multisource knowledge integration technique," in *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms*, M. A. Fischler and O. Firschein, Eds. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987, pp. 741–752. [Online]. Available: http://dl.acm.org/citation.cfm?id=33517.33576

[14] U. Bhattacharya and S. K. Parui, "An improved backpropagation neural network for detection of road-like features in satellite imagery," *International Journal of Remote Sensing*, vol. 18, no. 16, pp. 3379–3394, 1997.

[15] Z. Zhang, Q. Liu, and Y. Wang, "Road extraction by deep residual u-net," *CoRR*, vol. abs/1711.10684, 2017. [Online]. Available: http://arxiv.org/abs/1711.10684

[16] W. Wang, N. Yang, Y. Zhang, F. Wang, T. Cao, and P. Eklund, "A review of road extraction from remote sensing images," *Journal of Traffic and Transportation Engineering (English Edition)*, vol. 3, no. 3, pp. 271 – 282, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2095756416301076

[17] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 3212–3217.

[18] X. Hu, C. V. Tao, and Y. Hu, "Automatic road extraction from dense urban area by integrated processing of high resolution imagery and lidar data," *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences. Istanbul, Turkey*, vol. 35, p. B3, 2004.

[19] J. Zhao, S. You, and J. Huang, "Rapid extraction and updating of road network from airborne lidar data," in *2011 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*. IEEE, 2011, pp. 1–7.

[20] J. Zhao and S. You, "Road network extraction from airborne lidar data using scene context," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on*. IEEE, 2012, pp. 9–16.

[21] V. Mnih and G. Hinton, "Learning to detect roads in high-resolution aerial images," *Computer Vision–ECCV 2010*, pp. 210–223, 2010.

[22] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3431–3440.

[23] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2015, pp. 234–241.

[24] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2018.

[25] F. J. Huang, Y.-L. Boureau, Y. LeCun *et al.*, "Unsupervised learning of invariant feature hierarchies with applications to object recognition," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE, 2007, pp. 1–8.

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[27] G. Cheng, Y. Wang, S. Xu, H. Wang, S. Xiang, and C. Pan, "Automatic road detection and centerline extraction via cascaded end-to-end convolutional neural network," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 6, pp. 3322–3337, 2017.

[28] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[29] C. Hazirbas, L. Ma, C. Domokos, and D. Cremers, "Fusenet: incorporating depth into semantic segmentation via fusion-based cnn architecture," in *Asian Conference on Computer Vision*, November 2016.

[30] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik, "Learning rich features from rgb-d images for object detection and segmentation," in *European Conference on Computer Vision*. Springer, 2014, pp. 345–360.

[31] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard, "Multimodal deep learning for robust rgb-d object recognition," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on.* IEEE, 2015, pp. 681–687.

[32] S. Gupta, P. Arbelaez, and J. Malik, "Perceptual organization and recognition of indoor scenes from rgb-d images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 564–571.

[33] C. Becker, N. Häni, E. Rosinskaya, E. d'Angelo, and C. Strecha, "Classification of aerial photogrammetric 3d point clouds," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. IV-1/W1, pp. 3–10, 2017. [Online]. Available: https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-1-W1/3/2017/

[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[35] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" in *Advances in Neural Information Processing Systems*, 2018, pp. 2488–2498.

[36] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *International Conference on Machine Learning*, 2013, pp. 115–123.

[37] *Tallahassee-Leon County Geographic Information Systems.* [Online]. Available: http://www.tlcgis.org/

[38] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[39] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," *CoRR*, vol. abs/1411.4734, 2014. [Online]. Available: http://arxiv.org/abs/1411.4734

[40] M. A. Rahman and Y. Wang, "Optimizing intersection-over-union in deep neural networks for image segmentation," in *Advances in Visual Computing*, G. Bebis, R. Boyle, B. Parvin, D. Koracin, F. Porikli, S. Skaff, A. Entezari, J. Min, D. Iwai, A. Sadagic, C. Scheidegger, and T. Isenberg, Eds. Cham: Springer International Publishing, 2016, pp. 234–244.

[41] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[42] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[43] D. Delling, A. V. Goldberg, M. Goldszmidt, J. Krumm, K. Talwar, and R. F. Werneck, "Navigation made personal: Inferring driving preferences from gps traces," in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2015, p. 31.

[44] J. Biagioni and J. Eriksson, "Inferring road maps from global positioning system traces: Survey and comparative evaluation," *Transportation research record*, vol. 2291, no. 1, pp. 61–71, 2012.

[45] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady, "Preserving privacy in gps traces via uncertainty-aware path cloaking," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 161–171.

[46] K. Zhao, J. C. Suarez, M. Garcia, T. Hu, C. Wang, and A. Londo, "Utility of multi-temporal lidar for forest and carbon monitoring: Tree growth, biomass dynamics, and carbon flux," *Remote Sensing of Environment*, vol. 204, pp. 883–897, 2018.

[47] Z. Azizi, A. Najafi, and S. Sadeghian, "Forest road detection using lidar data," *Journal of forestry research*, vol. 25, no. 4, pp. 975–980, 2014.

[48] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[49] T. K. Dey and P. Kumar, "A simple provable algorithm for curve reconstruction," in *In Proc. 10th ACM-SIAM Sympos. Discrete Algorithms*. Citeseer, 1999.

[50] N. Amenta, M. Bern, and D. Eppstein, "The crust and the β-skeleton: Combinatorial curve reconstruction," *Graphical models and image processing*, vol. 60, no. 2, pp. 125–135, 1998.

[51] D. Chen, L. J. Guibas, J. Hershberger, and J. Sun, "Road network reconstruction for organizing paths," in *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2010, pp. 1309–1320.

[52] S. Karagiorgou and D. Pfoser, "On vehicle tracking data-based road network generation," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*. ACM, 2012, pp. 89–98.

[53] T. Zhang and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Communications of the ACM*, vol. 27, no. 3, pp. 236–239, 1984.

[54] R. Ramamurthy and R. T. Farouki, "Voronoi diagram and medial axis algorithm for planar domains with curved boundaries i. theoretical foundations," *Journal of Computational and Applied Mathematics*, vol. 102, no. 1, pp. 119–141, 1999.

[55] M. Aanjaneya, F. Chazal, D. Chen, M. Glisse, L. Guibas, and D. Morozov, "Metric graph reconstruction from noisy data," *International Journal of Computational Geometry & Applications*, vol. 22, no. 04, pp. 305–325, 2012.

[56] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[57] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 801–818.

[58] G. Zhao, J. Wang, and Z. Zhang, "Random shifting for cnn: a solution to reduce information loss in down-sampling layers," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 3476–3482. [Online]. Available: https://doi.org/10.24963/ijcai.2017/486

[59] T. M. Chan, "Closest-point problems simplified on the ram," in *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '02. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002, pp. 472–473. [Online]. Available: http://dl.acm.org/citation.cfm?id=545381.545444

[60] M. Bern, D. Eppstein, and J. Gilbert, "Provably good mesh generation," *J. Comput. Syst. Sci.*, vol. 48, no. 3, pp. 384–409, Jun. 1994. [Online]. Available: http://dx.doi.org/10.1016/S0022-0000(05)80059-5

[61] D.-T. Lee and B. J. Schachter, "Two algorithms for constructing a delaunay triangulation," *International Journal of Computer & Information Sciences*, vol. 9, no. 3, pp. 219–242, 1980.

[62] A. Saalfeld, "Topologically consistent line simplification with the douglas-peucker algorithm," *Cartography and Geographic Information Science*, vol. 26, no. 1, pp. 7–18, 1999.

[63] T. Erlebach, K. Jansen, and E. Seidel, "Polynomial-time approximation schemes for geometric intersection graphs," *SIAM Journal on Computing*, vol. 34, no. 6, pp. 1302–1323, 2005.

[64] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 6, pp. 679–698, 1986.

[65] S. Tsogkas and S. Dickinson, "Amat: Medial axis transform for natural images," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[66] J. Reid, L. Sutherland, B. Ray, A. Daleiden, P. Jenior, J. Knudsen *et al.*, "Median u-turn intersection: informational guide." United States. Federal Highway Administration. Office of Safety, Tech. Rep., 2014.

[67] S. Funke and E. A. Ramos, "Reconstructing a collection of curves with corners and endpoints," in *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '01. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2001, pp. 344–353. [Online]. Available: http://dl.acm.org/citation.cfm?id=365411.365477

[68] D. Koutra, A. Parikh, A. Ramdas, and J. Xiang, "Algorithms for graph similarity and subgraph matching," in *Proc. Ecol. Inference Conf.*, 2011.

[69] A. Sanfeliu and K.-S. Fu, "A distance measure between attributed relational graphs for pattern recognition," *IEEE transactions on systems, man, and cybernetics*, no. 3, pp. 353–362, 1983.

[70] C. Wiedemann, C. Heipke, H. Mayer, and O. Jamet, "Empirical evaluation of automatically extracted road axes," *Empirical evaluation techniques in computer vision*, pp. 172–187, 1998.

[71] J. D. Wegner, J. A. Montoya-Zegarra, and K. Schindler, "A higher-order crf model for road network extraction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 1698–1705.

[72] W. J. Stein and T. R. Neuman, "Mitigation strategies for design exceptions," CH2M HILL, Inc., Mendota Heights, Minnesota, Tech. Rep. FHWA-SA-07-011, July 2007.

[73] X. Yin, J. Han, and P. S. Yu, "Truth discovery with multiple conflicting information providers on the web," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 6, pp. 796–808, June 2008.

[74] X. L. Dong, L. Berti-Equille, and D. Srivastav, "Integrating conflicting data: the role of source dependence," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 550–561, Auguest 2009.

[75] X. L. Dong, L. Berti-Equille, and D. Srivastava, "Integrating conflicting data: the role of source dependence," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 550–561, 2009.

[76] T. Rekatsinas, X. L. Dong, L. Getoor, and D. Srivastava, "Finding quality in quantity: The challenge of discovering valuable sources for integration." in *CIDR*, 2015.

[77] B. Zhao and J. Han, "A probabilistic model for estimating real-valued truth from conflicting sources," *Proc. of QDB*, 2012.

[78] E. L. Lehmann, G. Casella, and G. Casella, *Theory of point estimation*. Wadsworth & Brooks/Cole Advanced Books & Software, 1991.

[79] M.-F. Balcan, A. Blum, and N. Srebro, "A theory of learning with similarity functions," *Machine Learning*, vol. 72, no. 1-2, pp. 89–112, 2008.

[80] X. L. Dong, E. Gabrilovich, K. Murphy, V. Dang, W. Horn, C. Lugaresi, S. Sun, and W. Zhang, "Knowledge-based trust: Estimating the trustworthiness of web sources," *Proceedings of the VLDB Endowment*, vol. 8, no. 9, pp. 938–949, 2015.

[81] X. Li, X. L. Dong, W. M. K. Lyons, and D. Srivastava, "Truth finding on the deep web: is the problem solved?" in *Proceedings of the 39th international conference on Very Large Data Bases*, 2013, pp. 97–108.

[82] X. Li, X. L. Dong, K. B. Lyons, W. Meng, and D. Srivastava, "Scaling up copy detection," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. IEEE, 2015, pp. 89–100.

[83] H. Xiao, J. Gao, Z. Wang, S. Wang, L. Su, and H. Liu, "A truth discovery approach with theoretical guarantee," 2016.

[84] M. Wan, X. Chen, L. Kaplan, J. Han, J. Gao, and B. Zhao, "From truth discovery to trustworthy opinion discovery: An uncertainty-aware quantitative modeling approach," 2016.

[85] H. Xiao, J. Gao, Q. Li, F. Ma, L. Su, Y. Feng, and A. Zhang, "Towards confidence in the truth: A bootstrapping based truth discovery approach," 2016.

[86] P. Buneman, S. Davidson, and A. Kosky, "Theoretical aspects of schema merging," in *Advances in Database Technology—EDBT'92*. Springer, 1992, pp. 152–167.

[87] R. A. Pottinger and P. A. Bernstein, "Merging models based on given correspondences," in *Proceedings of the 29th international conference on Very large data bases-Volume 29*. VLDB Endowment, 2003, pp. 862–873.

[88] L. Qian, M. J. Cafarella, and H. V. Jagadish, "Sample-driven schema mapping," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 73–84.

[89] M. A. Hernández, R. J. Miller, and L. M. Haas, "Clio: A semi-automatic tool for schema mapping," *ACM SIGMOD Record*, vol. 30, no. 2, p. 607, 2001.

[90] Z. Bellahsene, A. Bonifati, E. Rahm *et al.*, *Schema matching and mapping*. Springer, 2011, vol. 20.

[91] D. W. Embley, L. Xu, and Y. Ding, "Automatic direct and indirect schema mapping: experiences and lessons learned," *ACM Sigmod Record*, vol. 33, no. 4, pp. 14–19, 2004.

[92] V. J. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, 2004.

[93] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[94] Z. He, S. Deng, and X. Xu, "An optimization model for outlier detection in categorical data," in *Advances in Intelligent Computing*. Springer, 2005, pp. 400–409.

[95] B. Scholkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[96] E. M. Knorr and R. T. Ng, "A unified notion of outliers: Properties and computation." in *KDD*, 1997, pp. 219–222.

[97] J. Matoušek, *Lectures on discrete geometry*. Springer New York, 2002, vol. 108.

[98] V. V. Vazirani, *Approximation algorithms*. Springer Science & Business Media, 2013.

[99] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM review*, vol. 38, no. 1, pp. 49–95, 1996.

[100] R. A. Johnson, D. W. Wichern *et al.*, *Applied multivariate statistical analysis*. Prentice hall Englewood Cliffs, NJ, 1992, vol. 4.

[101] T. Yang, J. Liu, L. McMillan, and W. Wang, "A fast approximation to multidimensional scaling," in *IEEE workshop on Computation Intensive Methods for Computer Vision*, 2006.

[102] M. Onderwater, "Outlier preservation by dimensionality reduction techniques," *International Journal of Data Analysis Techniques and Strategies*, vol. 7, no. 3, pp. 231–252, 2015.

[103] S. Chatterjee, B. Neff, and P. Kumar, "Instant approximate 1-center on road networks via embeddings," in *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2011, pp. 369–372.

[104] J. Dunagan and S. Vempala, "Optimal outlier removal in high-dimensional," in *Proceedings of the thirty-third annual ACM symposium on Theory of computing*. ACM, 2001, pp. 627–636.

[105] A. Blum, A. M. Frieze, R. Kannan, and S. Vempala, "A polynomial-time algorithm for learning noisy linear threshold functions." in *FOCS*. IEEE Computer Society, 1996, pp. 330–338. [Online]. Available: http://dblp.uni-trier.de/db/conf/focs/focs96.html#BlumFKV96

[106] V. D. Milman and A. Pajor, "Isotropic position and inertia ellipsoids and zonoids of the unit ball of a normed n-dimensional space," in *Geometric aspects of functional analysis*. Springer, 1989, pp. 64–104.

[107] R. Kannan, L. Lovász, and M. Simonovits, "Isoperimetric problems for convex bodies and a localization lemma," *Discrete & Computational Geometry*, vol. 13, no. 3-4, pp. 541–559, 1995.

[108] C. Chang and C. Lin, "Libsvm: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 27:1–27:27, 2011.

[109] X. Yin, J. Han, and P. S. Yu, "Truth discovery with multiple conflicting information providers on the web," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '07. New York, NY, USA: ACM, 2007, pp. 1048–1052. [Online]. Available: http://doi.acm.org/10.1145/1281192.1281309

[110] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[111] "Open library," https://openlibrary.org/.

[112] "Open library data dumps," https://openlibrary.org/developers/dumps.

[113] D. Robinson, *The Incredible Growth of Python*. [Online]. Available: https://stackoverflow.blog/2017/09/06/incredible-growth-python/

[114] *GitHub Octoverse 2017 — Highlights from the last twelve months*. [Online]. Available: https://octoverse.github.com/

[115] S. Witowski, "Python at cern," Tech. Rep., 2017.

[116] T. Jenkins, "The first language-a case for python?" 2004.

[117] P. Guo, *Python Is Now the Most Popular Introductory Teaching Language at Top U.s. Universities*. [Online]. Available: https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext

BIOGRAPHICAL SKETCH

Biswas Parajuli is a PhD student in the Department of Computer Science at Florida State University (FSU). His research focus is on machine learning and computer vision. He studies object identification and segmentation techniques in remotely sensed imagery and Lidar. He is especially interested in applying vision related algorithms for solving real world problems.

Biswas holds a Masters in Science degree (2015) in Computer Science from FSU and a Bachelor in Technology in Computer Science and Engineering (2010) from the National Institute of Technology, Durgapur, India. Prior to joining FSU, he worked as a Lecturer at Gandaki College of Engineering and Science, Pokhara, Nepal from 2010 to 2011. He has been supported by the Government of India, FSU, National Science Foundation and US Air Force for his education and research.