GEM: An Efficient Entity Matching Framework for Geospatial Data

by

Setu Nilesh Shah

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved June 2021 by the
Graduate Supervisory Committee:

Mohamed Sarwat, Chair
Giulia Pedrielli
Dragan Boscovic

ARIZONA STATE UNIVERSITY

August 2021

ABSTRACT

The use of spatial data has become very fundamental in today's world. Ranging from fitness trackers to food delivery services almost all application record users' location information and require clean geospatial data to enhance various features of the application. As spatial data flows in from heterogeneous sources various problems arise. The study of entity matching has been a fervent step in the process of producing clean usable data. Entity matching in an amalgamation of various sub processes including blocking and matching. At the end of an entity matching pipeline we get deduplicated records of the same real world entity. Identifying various mentions of the same real-world locations is known as spatial entity matching. While entity matching received significant interest in the field of relational entity matching, the same cannot be said about spatial entity matching.

In this dissertation, I build an end-to-end Geospatial Entity Matching framework, GEM that explores spatial entity matching from a novel perspective. In the current state-of-the-art systems spatial entity matching is only done on one type of geometrical data variant. Instead of confining to matching spatial entities of only point geometry type, I work on extending the boundaries of spatial entity matching to match the more generic polygon geometry entities as well. I propose a methodology to provide support for three entity matching scenarios across different geometrical data types : point $\times$ point, point $\times$ polygon, polygon $\times$ polygon. As mentioned above entity matching consists of various steps but blocking, feature vector creation, and classification are the core steps of the system. GEM comprises an efficient and lightweight blocking technique, GeoPrune, that uses the geohash encoding mechanism to prune away the obvious non-matching spatial entities. Geohashing is a technique to convert a point location coordinates to an alphanumeric code string. This tech-

nique proves to be very effective and swift for the blocking mechanism. I leverage the Apache Sedona engine to create the feature vectors. Apache Sedona is a spatial database managment system that holds the capacity of processing spatial SQL queries with multiple geometry types without compromising on their original coordinate vector representation. In this step, I re-purpose the spatial proximality operators (SQL queries) in Apache Sedona to create spatial feature dimensions that capture the proximity between a geospatial entity pair. The last step of an entity matching process is matching or classification. The classification step in GEM is a pluggable component, which consumes the feature vector for a spatial entity pair and determines whether the geolocations match or not. The component provides 3 machine learning models that consume the same feature vector and provide a label for the test data based on the training. I conduct experiments with the three classifiers upon multiple large-scale geospatial datasets consisting of both spatial and relational attributes. Data considered for experiments arrives from heterogeneous sources and we pre-align its schema manually. GEM achieves an F-measure of 1.0 for a point $\times$ point dataset with 176k total pairs, which is 42% higher than a state-of-the-art spatial EM baseline. It achieves F-measures of 0.966 and 0.993 for the point $\times$ polygon dataset with 302M total pairs, and the polygon $\times$ polygon dataset with 16M total pairs respectively.

DEDICATION

To my beloved parents for standing by me through thick and thin

ACKNOWLEDGMENTS

I am grateful to my esteemed advisor, Dr. Mohamed Sarwat, for his invaluable guidance, continuous support and patience throughout my Master's thesis study. His immense knowledge and plentiful experience has helped me discover research acumen in myself and embrace it. I would like to thank the higher power of the universe for always looking out for me and giving me opportunities to make my self better everyday. Additionally, I would like to thank Dr. Giulia Pedrielli and Dr. Dragan Boscovic for their valuable feedback and advice. I am also very thankful to my fellow members of the Datasys Lab for helping me overcome various hurdles in the process and making the roller-coaster ride fun. Moreover, I am grateful for my amazing friends who taught me to believe in myself and themselves never stopped believing in me. Last but not the least, I would like to mention that my family members have always been the pillars of life and there are no words for the gratitude I feel towards them. Their prayers and spirit, has given me the strength to finish my Master's study.

TABLE OF CONTENTS

Page

LIST OF TABLES vii

LIST OF FIGURES viii

CHAPTER

1 INTRODUCTION 1

2 LITERATURE REVIEW 5

3 PROBLEM DEFINITION AND BASELINE APPROACHES 16

3.1 Problem Definition 16

3.2 Baseline Approaches 18

3.2.1 Magellan 18

3.2.2 DeepMatcher 20

3.2.3 QuadSky 21

4 SYSTEM ARCHITECTURE 23

4.1 Overview 23

4.2 Spatial Blocking 27

4.3 Feature Vector Creation 31

4.4 Classification 35

5 EXPERIMENTAL EVALUATION 37

5.1 Experimental Setup 37

5.2 Datasets 37

5.2.1 Point × Point 38

5.2.2 Point × Polygon and Polygon × Polygon 39

5.3 Comparison of Classifiers 40

5.4 Evaluation of Blocking 42

5.5 Comparison with Baselines 44

CHAPTER Page

6 CONCLUSION AND FUTURE WORK 48

REFERENCES 49

LIST OF TABLES

Table Page

2.1 State-of-the-art Approaches 13

5.1 Details of the Spatial Datasets 38

5.2 Test Performance Evaluation of GEM Across Various Classifiers and Datasets 40

5.3 Comparison with Baselines for Point-point Case 46

5.4 Comparison with Baselines for Point-polygon Case 46

5.5 Comparison with Baselines for Polygon-polygon Case 46

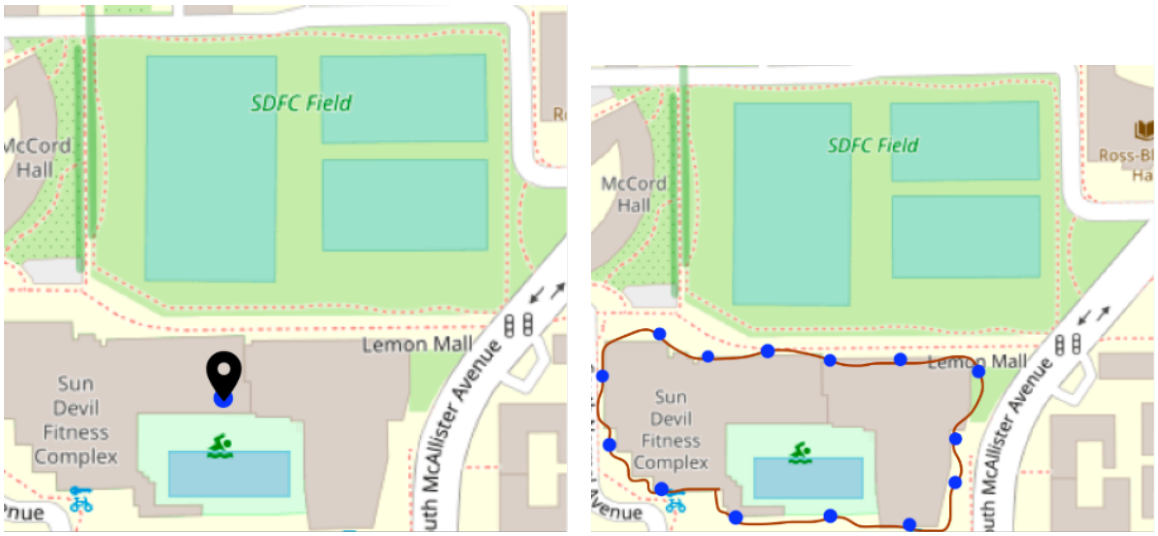
LIST OF FIGURES

Figure		Page
1.1	Point and Polygon Examples of Sun Devil Fitness Complex at Arizona State University	1
2.1	Entity Matching Flow	6
2.2	Workflow of Geobench system	10
2.3	Workflow of Geoalign System	10
3.1	Illustrative Examples of Spatial Entity Matching Across Diverse Geometries	17
3.2	Baseline Relational & Spatial Entity Matching Approaches	19
4.1	System Architecture Overview of *GEM*	23
4.2	Feature Vector Skeleton	32
5.1	Latency Comparisons of the Three Classifiers Across All the Datasets	40
5.2	Evaluation of Random Forest Classifier for Various Values of K in Fodors-zagats Dataset	41
5.3	Comparison of GEM with Baselines Across 3 Spatial Datasets with Different Geometries	45

Chapter 1

INTRODUCTION

The past decade has witnessed a spatial data deluge, thanks to a plethora of applications that are constantly collecting geospatial location data. Examples of such applications include but are not restricted to social media check-ins, fitness trackers, cab services, e-bikes, etc. As geospatial data flows in from different sources, various hurdles arise such as data inconsistency, data redundancy, incorrect or incomplete data, discrepancy between old and new data and much more. To make it useful, such data needs to be curated before being passed to applications. Entity matching (EM) is a prominent data integration technique Elmagarmid *et al.* (2006), the goal of which is to match different mentions of the same real-world entity Meduri *et al.*



(a) Point (b) Polygon

Figure 1.1: Point and Polygon Examples of Sun Devil Fitness Complex at Arizona State University

(2020) across diverse data sources. These real-world entities can be of any type such as personal data, object data, location data or more.

EM on spatial data is known as spatial entity matching, which is the task of determining whether the given spatial entities map to the same geolocation Isaj *et al.* (2019). Given a spatial entity pair s_1 and s_2, with n attributes each, an EM system can classify the pair as a *match* or a *non-match*. Entity s_1, for instance, can be represented as a record with the following attributes *name: "Ike's"*, *location: "33.423, -111.939"*, *ratings: "3.7"*, *category: "salad, cafe, bread"* while s_2 respectively can be *"Ike's Love and Sandwich" , "33.43, -111.92" , "4" , "store, sandwich, restaurant"*. This pair should be classified as a *match* since both records refer to the same location (i.e., Ike' s restaurant), despite not having the same values for some of the attributes. EM on relational data (aka. relational EM), has been a fervent area of research for a few decades now Elmagarmid *et al.* (2006); Christen (2012); Barlaug and Gulla (2021). Relational EM predominantly assumes that the attributes in a record pair can be represented as a collection of strings and thus leverages string similarity functions from string matching libraries to derive numerical feature vectors for a record pair Monge *et al.* (1996); Monge and Elkan (1997). Unlike relational data, we cannot simply assume that a string-based representation suffices for spatial EM. Treating spatial coordinates (i.e., latitude and longitude) as strings will result in a significant loss of semantic information and poor matching performance.

Existing works on spatial EM Sehgal *et al.* (2006); Karam *et al.* (2010); Morana *et al.* (2014); Berjawi *et al.* (2014); Barret *et al.* (2019); Isaj *et al.* (2019) match only a single type of geometrical spatial data, i.e., a *point* to a *point*, which falls short in catering to diverse geometric data types that geospatial data comes in, i.e., point, polygon, multi-polygon, and more. Surprisingly, none of the spatial EM work so far recognizes that a *point* can be matched to a *polygon* or that a *polygon* can be

matched to another *polygon*. One such example is shown in Figure 1.1 where a record s_1: (*name: "SDFC", location: "(33.4152, -111.9310)", ratings: "5", category: "gym, pool, football ground"*) representing a *point* is potentially matched to a *polygon* depicted respectively by s_2: (*"Sun Devil Fitness Complex" , "33.4159312, -111.9331601; .....; 33.4158702, -111.9326267; 33.41587, -111.9327201; 33.4158436, -111.9327202", "4.5", "fitness building, ASU, gym"*). While s_1 contains the point coordinates of the fitness center at Arizona State University (Figure 1.1a), s_2 contains the coordinates of a multi-vertex polygon that encompasses the whole building (Figure 1.1b). Although their spatial extent is different, they should be classified as a 'match' since they refer to the same real-world location. A spatial EM system capable of matching diverse geometries is essential for several applications such as (a) unification of check-ins for an event from various social media platforms, (b) information integration about the same place from multiple datasets (e.g., Yelp and Open Street Maps), (c) location disambiguation among proximal, but different spatial entities, (d) matching addresses to the same venue.

In this dissertation, I introduce an end-to-end Geospatial Entity Matching system called GEM, which can match a pair of spatial entities regardless of their geometry types. Note that the entities are represented as records containing both relational and spatial attributes. I also talk about a lightweight blocking mechanism, $GeoPrune$, to prune away the obvious non-matches by leveraging the geohash encoding technique Wikipedia contributors (2020). Our system leverages Apache Sedona TheApacheFoundation (2020a) which is a scalable geospatial data processing engine to create feature vectors for a pair of spatial entities. I re-purpose the spatial query operators in Apache Sedona to create numerical dimensions for spatial attributes that capture the spatial proximity between a geospatial entity pair. In order to create numerical feature dimensions for relational attributes, the Simmetrics library TheA-

pacheFoundation (2004) is utilized. Thus, *GEM* encodes the information about both relational similarity and spatial proximality between a spatial entity pair which is converted into a numerical feature vector, that is passed to a binary classifier to generate a match or non-match label for the entity pair.

Following are our contributions in this thesis:

- An end-to-end Geospatial EM system called *GEM* that can provide seamless support to match a *point* with a *point*, a *point* with a *polygon* and a *polygon* with another *polygon*.
- A new lightweight spatial blocking mechanism called, *GeoPrune*, that uses the geohash encoding technique to prune the obvious non-matching entity pairs.
- To build numerical feature vectors at scale, the system leverages a scalable geospatial data processing engine, namely Apache Sedona, to encode the spatial proximity between the locations in an entity pair as well as the Simmetrics framework to encode the similarity between the relational attributes.
- It supports binary classifier variants of three out-of-the-box ML classifiers, i.e., Random decision forests (RF), Support Vector Machines (SVM), and feed-forward neural network (NN) to be plugged into the classification module of *GEM* that consumes a feature vector for a pair of geospatial locations and classifies them as *matching* or not.

The remainder of this article is structured as follows. Chapter 2 summarizes the related work. The detailed problem definition and a description of the state-of-the-art baselines from both relational and spatial EM that I have adapted towards solving the problem in Chapter 3. The system architecture of *GEM* in Chapter 4, followed by experimental results in Chapter 5. I conclude and provide future scope in Chapter 6.

Chapter 2

LITERATURE REVIEW

Entity matching problems and challenges have received a lot of attention since the beginning of the big data surge. This chapter attempts to give a brief overview of the state-of-the art approaches for entity matching. It starts with providing definitions and understanding of entity matching, then talks about various art approaches for entity matching in relational databases. In the second section there is literature supporting the surge of spatial databases and various challenging work done in this area. Furthermore it briefly discusses the outline of entity matching process and describes the current state of the art systems that claim to support spatial entity matching AKA entity matching or deduplication in spatial data. The next section talks about geohashes and the extent of exploration they have been subjected to.

Relational EM assumes that the attributes to be matched across records are predominantly textual in nature. Thus, it uses string similarity functions (e.g., Jaccard similarity) to compare the pre-aligned attributes of an entity pair, and generate numerical similarity scores which are used as dimensions in a feature vector. Prior works on blocking Papadakis *et al.* (2016) compare one or more of these attribute similarity scores with predefined thresholds to prune away the obvious non-matching pairs. Full feature vector are created only on the surviving post-blocking pairs which are subsequently passed to a heuristic matcher or a binary classifier.

Entity matching is a very vital data integration procedure. It is defined as a core data cleaning activity that aims to find data instances that refer to the same real-world entity Meduri *et al.* (2020)Köpcke and Rahm (2010)Konda *et al.* (2016). Entity resolution, de-duplication, record linkage, data matching, tuple matching are

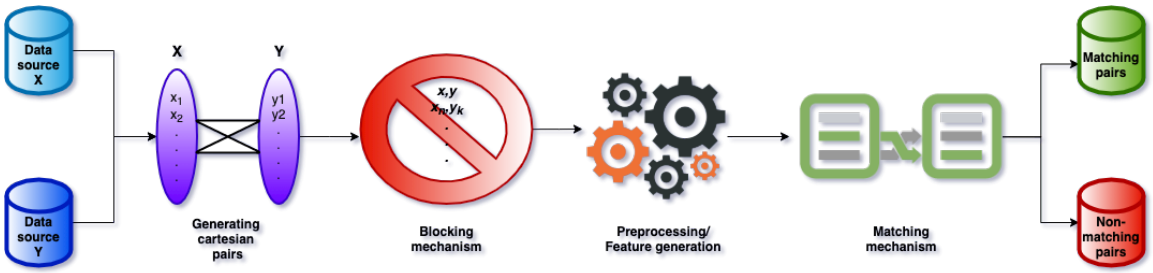


Figure 2.1: Entity Matching Flow

all synonymous with entity matching. The subject of record linkage has received copious attention in the relational database community. Although relational EM has seen a substantial amount of work, the same cannot be said about spatial EM. Some recent work exists for spatial data integration Walter and Fritsch (1999); Balley *et al.* (2004) and testing spatial EM algorithms Morana *et al.* (2014); Barret *et al.* (2019); Berjawi *et al.* (2014). Going further the chapter reviews various techniques, algorithms and machine learning models that claim to achieve state of the art results in the task of entity matching- both with relational data and some with spatial data. I first summarize the work on relational EM, followed by that on spatial EM.

Entity matching scenarios generally consist of 2 tuples X and Y, are put through a set of steps and algorithms to annotate them whether they are the same real-world entity or not i.e. are they a *match* or a *non-match*. The generic entity resolution process has a skeleton of steps like the pairing step, blocking step, and the matching step as shown in the Figure 2.1. This pair of tuples could be product information, location-based information, personal information, restaurant rating information, or any other kind of data information.

Entity matching has been a very popular and heavily-researched subject matter Christen (2012)Elmagarmid *et al.* (2006)Getoor and Machanavajjhala (2012)Papadakis *et al.* (2016). Art work in the area of entity matching on relational databases

ranges from initial intuitive approaches of using similarity values and weighted average of the computed similarity values Jaro (1989)Landau and Vishkin (1989)Monge *et al.* (1996)Monge and Elkan (1997), though many of these initial work assume that the data is well structured and mostly noise-free which is a very ideal scenario far from the nature of obtained data nowadays. Later, there have been articles using supervised and semi-supervised learning methods alongside active-learning techniques Meduri *et al.* (2020)Cochinwala *et al.* (2001)Monge and Elkan (1997)Bansal *et al.* (2004)Domingos (2004)Tejada *et al.* (2001)Tejada *et al.* (2002). Numerous early approaches rely on string similarity scores of one primary attribute pair to infer the possibility of linkage between the given entities Elmagarmid *et al.* (2006). Other approaches considered multiple attributes similarity scores and calculated the result using a weighted scoring function Christen *et al.* (2004); Jaro (1989); Landau and Vishkin (1989); Monge and Elkan (1997); Monge *et al.* (1996). More recent relational EM approaches apply various machine learning(ML) techniques to the matching step. Verykios *et al.* (2000) use unsupervised ML techniques, whereas Cochinwala *et al.* (2001); Bansal *et al.* (2004); Tejada *et al.* (2001); Domingos (2004) use semi-supervised and supervised ML algorithms. End-to-end EM systems which support all the steps in the EM pipeline exist for both supervised learning Konda *et al.* (2016); Christen (2012) and active learning Meduri *et al.* (2020). Mudgal *et al.* (2018) apply deep learning techniques for relational EM. All the above methods and algorithms proposed are for relational databases and they all work with the basic assumption of considering the data fields as string and processing them. In spatial data the most important piece of information are the location coordinates- latitude and longitude and considering them string or text deteriorates their value. Due to this loophole many pioneer entity matching methods that work flawlessly for relational databases or large structured databases might not work as well for spatial data. I describe how

Konda *et al.* (2016) and Mudgal *et al.* (2018) are adapted to spatial EM in Chapter 3.

Data with location information AKA spatial attributes has been around for a long time, but recently since the last decade there has been a surge of spatial data and analysis around it. This is termed as spatial data deluge Sarwat and TheApacheFoundation (2020). Such data includes spatial-temporal data, socioeconomic data, geo-tagged social media, satellite imagery data, data gathered through various mobile devices/sensors and much more. Spatial data has started being a part of almost all the latest trending analysis ranging from news posts about various protests in India, tweets about blast in Beirut,Lebanon ; US presidential elections, COVID-19 data etc.

Since the early 2000s we started having many different apps and platforms collecting spatial data in the forms of geolocation check-ins, taxi trip information, GPS enabled vehicles, GPS and wifi enabled bicycles, traffic light sensors etc. Hence overtime spatial data also became a part of big data and started being used for enhancing personalized user experience. Data deduplication done on spatial objects (only location information) is generally referred to as spatial data integration. Artwork such as Balley *et al.* (2004); Walter and Fritsch (1999) discusses the integration of various forms of spatial data.

Due to easy availability of voluminous data this area has witnessed some great scientific experiments, conclusions and systems over time. As geospatial data continues to grow it becomes very important to have a scalable, reliable and efficient system in place which can help researchers working with large spatial data to program and progress with efficiency and ease. Apache Sedona is a cluster computing system which extends Apache Spark and Spark SQL to support spatial geometrical operations at scale Yu (2020).

Treating attributes like spatial coordinates (latitude and longitude) as strings

can prove to be detrimental for spatial EM. Existing works such as Balley *et al.* (2004); Walter and Fritsch (1999) assume that the spatial objects to be matched only contain spatial information. Unlike them, a spatial data point, also referred to as a spatial entity in this paper, can contain both relational attributes (like name, ratings, description, address, ..) and location coordinates representing diverse geometries. Most of the following works do assume that a spatial entity can contain both relational and spatial attributes. However, they only cater to $point \times point$ matching and do not consider polygons except Ruiz-Lendínez *et al.* (2017). In the area of entity resolution in geospatial data, there has been prior research and experiments attempting entity resolution in spatial data using similarity measures of non-spatial and spatial features and using the weighted average of these values to provide an label Sehgal *et al.* (2006). They work with both spatial and non-spatial attributes and calculator similarity values namely 3 attributes - Location name, Coordinates and Location type. The author then calculates the weighted average of the similarity values and derives a label for the pair X and Y. Sehgal *et al.* (2006) provide a matching algorithm based on distance measure for spatial features and string similarity scores for relational (non-spatial) features. Interestingly, this system solves the complementary problem of discovering non-matches instead of matches via a distance function. This approach works with an assumption that the data is clean, well populated and absolutely error free, which is a strong assumption to work with given the current nature of available data. In a scenario where tuple A : 'Name: Tazzo Cars, Coordinates: 33.427, -111.917, Type: Car repair shop' and tuple B:'Name: Tazzo Bar, Coordinates: 33.435, -111.924, Type: sports bar and food' ; according to weighted similarity this tuple pair will pass the threshold be classified as 'same' but in reality this is 'not same'. Hence this approach there is a high possibility of increased false positives and false negatives because a single combination of weights and a rigid threshold cannot do justice to

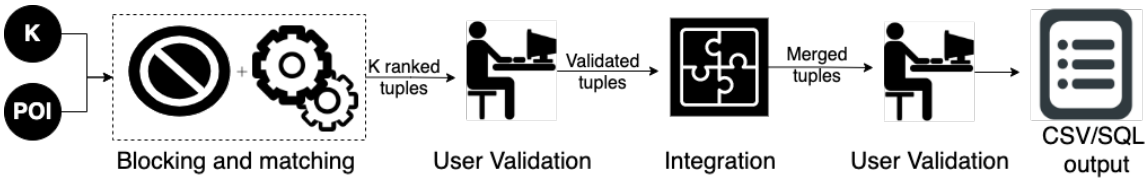


Figure 2.2: Workflow of Geobench system

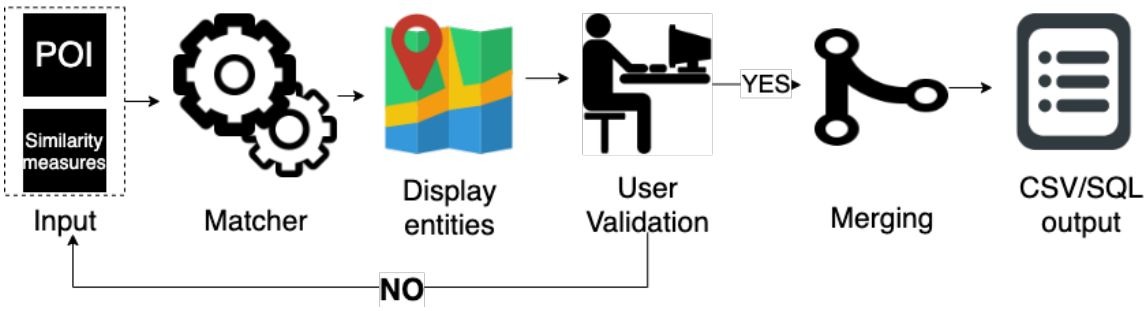


Figure 2.3: Workflow of Geoalign System

each and every tuple for various other data sources.

Some other research approaches work around on the point of interest(POI) such as restaurants, hotels, parks etc Morana *et al.* (2014)Barret *et al.* (2019). Figures 2.2 and 2.3 depicts the workflows of the two systems. Interestingly there has been substantial research and advancement in the area of available tools for testing spatial data matching algorithms but a lack of benchmark does not facilitate accurate comparison between various spatial matching algorithms Morana *et al.* (2014). Morana *et al.* (2014); Barret *et al.* (2019) build benchmark tools with usable interfaces to support spatial EM techniques. Morana *et al.* (2014) segregate locations into various types based on which they find points of interest (POIs) similar to the current location. Despite being a testing tool, Morana *et al.* (2014) describe an EM flow that has a blocking function with predefined threshold values corresponding to each location 'type' and a matching function that can handle both relational and spatial attributes.

Geobench assists users in doing so and the integration of corresponding spatial

entities which you can download as CSV files for future use. They work with an assumption that a specific set of attributes(classified as primary and secondary) have to be present in a contestant dataset for it to be eligible for testing through the system. They have a blocking and matching mechanism in place as a part of the process. The blocking mechanism works with a pre-decided fixed radius threshold value depending on the type of POI and for matching they calculate similarity scores on some of the attributes and rely upon a weighted score for the decision. This system involves a lot of manual intervention. It starts with taking input for POI and a value K, using Geonames, Here and Google Maps data they perform blocking and matching. The displayed top K results need to be validated by the user and then they go ahead and integrate the results using Levensteins score of the 'Title' attribute. Again post this there needs to be user validation and then the final tuple is represented with integrated information from all available sources about the POI entered by the user.

On the other hand, Barret *et al.* (2019) build a dedicated testing tool which offers a host of similarity metrics and parameter settings that can be fine-tuned as per the user preferences. GeoAlign allows users to customize and fine tune the combination function. The graphical user interface assists the visualization and merging of detected correspondencesBarret *et al.* (2019). Four cartographic providers namely - Open Street Map, Geonames, Here and Bing Maps are available in the system. GeoAlign provides various categories of similarity metrics for different attributes and settings to fine tune their weight according to the user's need. The user is expected to enter the POI and tune their similarity values after which GeoAlign processes and generates best possible results on the graphical map interface.

The GeoBench system works with some very rigid assumptions. Both the systems work with a defined set of data sources which ceases the possibility of trying the system on new datasets. Moreover GeoBench and GeoAlign are very productive research

contributions in the field of tools and systems for testing existing spatial algorithms in contrast to experimentally proving a fresh algorithm and its performance to be effective for spatial entity deduplication.

Isaj *et al.* (2019) build an end-to-end spatial EM system that has a spatial blocking step and a matching step. They employ a modified quadtree algorithm as the blocking mechanism and a heuristic of multiple skylines to classify the pairs. It comprises a blocking algorithm and a classification technique for spatial entity linkage. The blocking algorithm has the concept and complexity of the quadtree algorithm at the core of it Isaj *et al.* (2019). The variations proposed to the traditional quadtree algorithm to make it more compatible with spatial scenarios are as follows:

- Two point will only be considered if the distance between given 2 points has to be less than the diagonal of the rectangular quadtree child

- One point can be a part of more than one children of the quadtree; modifying the recursive working of quadtree to accommodate more points that are closer rather than splitting them randomlyIsaj *et al.* (2019)

Further they compute similarity scores of various attributes and using those similarity values generate a label for a particular pair. These pairs are represented as points in a space of n dimensions with 4 string similarity values, where n is the number of attributes. The labelling technique, SkyEx, uses Pareto optimality principle combined with a multiple skyline approach. They work with the assumption that the true positives or the positive matches are a minority and hence lie as outliers on the graph. Hence they propose to explore k skyline to ensure that all the possible true positives have been recorded. While the technique is a unique combination of functions, there is not much clarity provided on the determination of k and how experimental results varied with various values of k. Furthermore the blocking tech-

Approaches	Support for spatial data				
	Spatial Blocking	**Different geometry support**	**Feature generation**	**Training**	**Inference**
Magellan Konda *et al.* (2016)	No	N/A	No	No	No
Deepmatcher Mudgal *et al.* (2018)	No	N/A	No	No	No
GeoBench Morana *et al.* (2014)	Yes	Point	Yes	N/A	No
GeoAlign Barret *et al.* (2019)	No	Point	Yes	N/A	No
QuadSky Isaj *et al.* (2019)	Yes	Point	Yes	N/A	Yes

Table 2.1: State-of-the-art Approaches

nique proves to be very computationally heavy and did not scale well when exposed to large data. While carrying out experiments on the QuadSky system with various data sources, we noticed that the system needs a weighted value of the similarity scores, termed as 'preference function' to carry out the labelling step. Moreover we observed that the system exhibits large amounts of FP and FN which compromises the F1-score in case of large noisy data, probably because the systems relies heavily on similarity score and in case of inaccurate score the preference function gets compromised which results in poor performance. Also in scenarios where the diagonal length threshold is manipulated the system proves to be performing well only for a certain value which contradicts the establishment of the threshold as the density of the area does not affect the length in turn casting no shadow on the overall performance. Since their system can only support point matching, we describe how we adapt it towards matching polygon pairs in Chapter 3. Ruiz-Lendínez *et al.* (2017) build a system that matches two *polygons*, but it is not an EM system. They treat spatial data in an image format rather than the original coordinates vector. Unlike these techniques, we treat spatial geometries in their native format and leverage a spatial DBMS like Apache Sedona to enable the EM task.

A comparative study of various state of the art algorithms has been shown in

Table 2.1. It depicts the performance of these systems over various crucial features that a good spatial algorithm should possess. First up we have the most important measure of efficiency with spatial data, next we talk about the scalability of the system and blocking technique. As we talked about the increasing size of data and data deluge it is very important for a spatial system to be scalable. The blocking technique is a very crucial step of the entity matching pipeline. An easy technique which is low in its implementation complexity can give a scientist lots of freedom to experiment and increase the performance of the algorithm's training step. We talk about various systems but sadly not many have a good blocking algorithm. We then talk about how compatible are the existing algorithms with the various kinds of the geometric shapes in which spatial data is available, namely, points, polygons, lines. We observe that all the available art can only deal with a node or point and fails for enclosed shapes. Lastly we compare the complexity of the two large but principal steps of any algorithm - training and inference. The legend for these columns are - high, low and medium. When we say the step complexity is high we mean that it is difficult to break down or understand the inner workings which leads to less to none scope of improving the algorithm. Moreover high complexity also denotes that these systems take long hours even days to execute over chunks of comparatively small data.

The geohashing technique of converting point coordinates into a 12 character alphanumeric code has been well-known for its proximity searching use case GeohashPubnub (2020). Research has been done on how geohashing is a flexible and efficient way of converting latitude/longitude coordinates to string codes and vice versaMoussalli *et al.* (2015). Geohash code technique has been used for quite a few applications like object tagging along with universally unique identifiers for gaining spatio-temporal identifiers for all the data availableBalkić *et al.* (2012), indexing for

spatial data management in distributed memoryLiu *et al.* (2014). Geohashing has also been used as an encoding and decoding technique used to generate frequent itemsets in demographic prediction experiments Roy and Pebesma (2017). Astonishingly enough, geohashing technique has not been explicitly addressed or experimented as a potential classifying technique or blocking mechanism for spatial entity linkage.

In summary, note that there are numerous approaches to solve the relational EM problem but they need to be adapted towards solving the spatial EM problem. Spatial data needs to be handled distinctively, and hence there is a need for spatial EM systems. Most of the existing spatial data integration tools work on the testing of a spatial entity linkage algorithm. However, as depicted in Table 2.1, all the current EM artwork in the spatial community only supports the matching of a *point* to another *point*. Although some of the existing works on spatial EM can tackle both relational and spatial attributes, they miss out on catering to the diverse spatial geometry which includes polygons. *GEM* is a system that enables this vision of matching diverse spatial geometries while accommodating both relational and spatial attributes. I categorize the distinctive features of various state-of-the-art EM systems in Table 2.1. In the next chapter, I will introduce the problem statement and describe the state-of-the-art baselines I implemented for my thesis.

Chapter 3

PROBLEM DEFINITION AND BASELINE APPROACHES

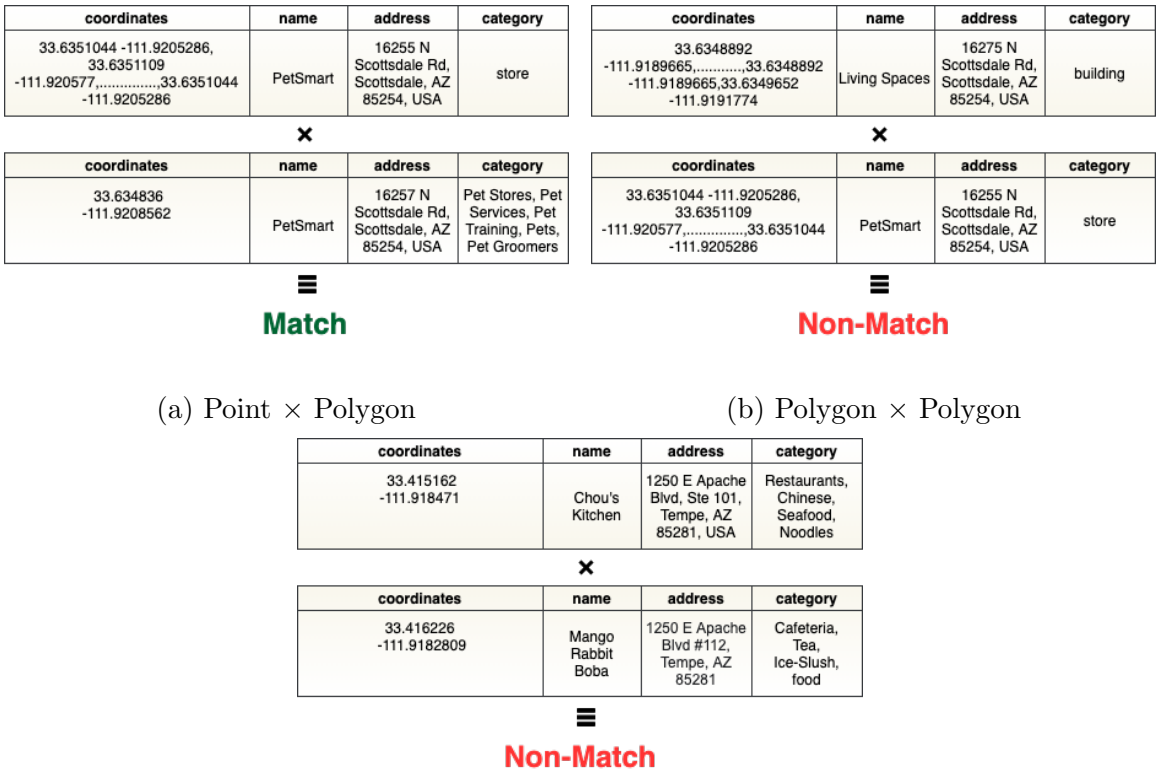
In the first half of this Chapter, I define our problem. The later half, discusses non-trivial details of how I implemented a few state-of-the-art baselines from both relational and spatial EM as shown in Figure 3.2.

3.1 Problem Definition

In this thesis, I attempted to solve the spatial EM problem. Consider two spatial datasets S_{left} and S_{right}. Tuples in these datasets are well-defined spatial entities that have both spatial features like *coordinates* and relational (non-spatial) attributes. The EM task here is to *match* spatial entities from S_{left} to that of S_{right}. Datasets S_{left} and S_{right} can have spatial data of either geometry type: *point* or *polygon*. We assume that the schema of the datasets, S_{left} and S_{right}, is pre-aligned. T_{point} is a spatial data tuple of geometry type *point* which has one pair of coordinates (i.e. latitude and longitude), and $T_{polygon}$ is a spatial data tuple of geometry type *polygon* which has multiple coordinate pairs. We consider all the possible scenarios that can occur within entity matching of multiple geometry types: *point* and *polygon*. *GEM* my EM system provides support for the following entity matching scenarios:

- $S_{left} \times S_{right}$, where $S_{left} \in T_{point}$ and $S_{right} \in T_{point}$
- $S_{left} \times S_{right}$, where $S_{left} \in T_{point}$ and $S_{right} \in T_{polygon}$
- $S_{left} \times S_{right}$, where $S_{left} \in T_{polygon}$ and $S_{right} \in T_{polygon}$

Figure 3.1 provides an example for each of these cases. Consider an entity pair s_1 and s_2, where $s_1 \in S_{left}$ and $s_2 \in S_{right}$. The pair contains one spatial attribute:



(a) Point × Polygon (b) Polygon × Polygon

(c) Point × Point

Figure 3.1: Illustrative Examples of Spatial Entity Matching Across Diverse Geometries

coordinates and three relational attributes: *name*, *address* and *category*. s_1 and s_2 in fig 3.1a are classified as a *match* based on the *coordinates* and the similarity between *name and address* . In contrast, the pair in Figure 3.1b is marked as a *non-match* because all the attributes are different. Even though *address* and *category* are similar in Figure 3.1c, the entity pair is classified as a *non-match* because of different *name* and *coordinates*. Also, Figure 3.1 depicts the three possible matching scenarios w.r.t. diverse spatial geometries - *point* × *point*, *point* × *polygon* and, *polygon* × *polygon*.

Since a few decades relational EM has been a hot research topic, spatial EM has come to the limelight very recently. As mentioned in section 2 the current spatial EM

systems are confined to work with only the *point* geometry type. It is surprising that the vast EM community hasn't considered matching spatial data of other geometries. While there is no prior artwork on *point* × *polygon* spatial matching, Ruiz-Lendínez *et al.* (2017) attempts matching a *polygon* with *polygon* using spatial image data. This system considers the spatial coordinates of different geometries and process them using Apache Sedona. Details about the working of GEM has been deferred to Chapter 4. A system that matches points with polygons and polygons with polygons can be useful in disambiguating various mentions of an event across different datasets. This is an end-to-end spatial EM system that supports entity matching across multiple spatial geometries. In the next half of this chapter I discuss various implementation details and parameters for the current state-of-the-art EM systems, both- relational and spatial that I consider as baselines and evaluate my system against.

3.2 Baseline Approaches

Among the approaches we describe below, Magellan and DeepMatcher are relational EM systems, whereas QuadSky is a spatial EM system. Figure 3.2 highlights their system workflow.

3.2.1 Magellan

Magellan Konda *et al.* (2016) is relational EM system consisting of all the expected steps in an end-to-end EM pipeline such as blocking, feature vector generation, model training and matching. Each of these building blocks are pluggable with multiple options to pick from. Among the various options for each component, I pick the default or best working option as highlighted by Konda *et al.* (2016). I use the overlap blocker

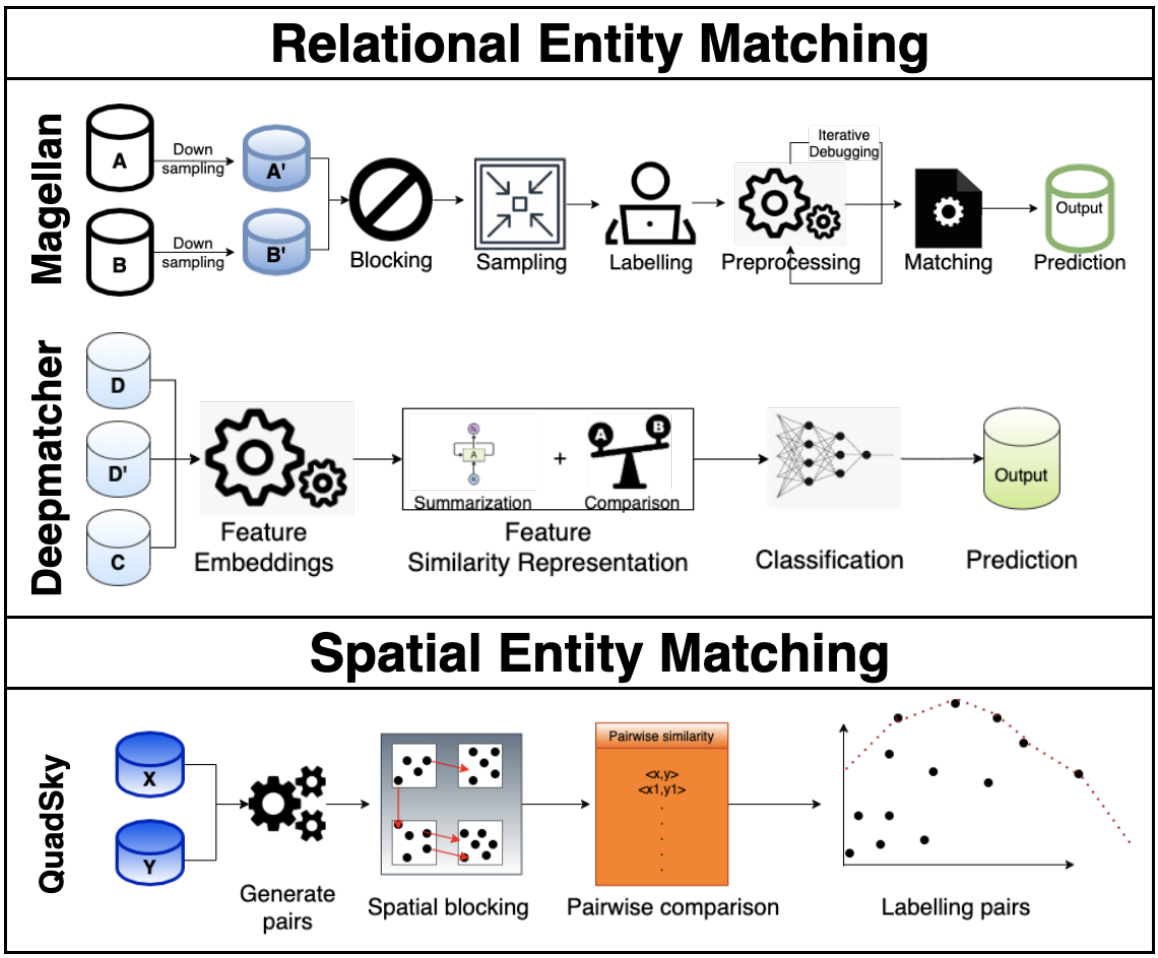


Figure 3.2: Baseline Relational & Spatial Entity Matching Approaches

among the various blocking options. Data labeling in this system is done manually, but as the post-blocking set grew to the order of a few millions I employed our semi-manual labelling technique. I defer experimental details to Chapter 5.5. As mentioned before, Magellan employs various string similarity functions like Jaro distance, Cosine similarity for feature vector generation. Hence, it treats the spatial attributes as text and calculates string similarity on them. It provides two types of matchers - learning-based and rule-based classifiers. Although rules are interpretable, rule-based classifiers are significantly more expensive to train than the learning-based classifier. Therefore, I choose Random Decision Forests as the matching component for Mag-

ellan. Additionally Random Forests are one of the three classifiers that I provide in the pluggable classification component of *GEM* and hence it would be an exciting experiment to compare the perfomace of *GEM* against relational EM state-of-the-art system Magellan Konda *et al.* (2016) I feed Magellan with pre-aligned spatial data sources.

3.2.2 DeepMatcher

DeepMatcher Mudgal *et al.* (2018) is a deep learning (DL) based approach for relational EM, which was specifically designed towards handling long text in the attributes. It thus focuses on deriving a suitable representation for the long textual attributes in the input entity pair using Recurrent Neural Networks (RNNs) and word embeddings generated using FastText FacebookOpenSource (2020) library, instead of the conventional string similarity functions. The system has three steps:

- feature embedding: Converts attributes of two data tuples into sequence of word embeddings.
- feature similarity representation: captures the similarity of the two entities and puts them into a feature vector
- classification: performs classification using a multi-layer neural network

The learned embeddings for entity pairs are passed as input to a multi-layer perceptron (MLP) which derives their classification outcome. Since DeepMatcher is a classifier and not on end-to-end system like Magellan, it does not have a blocking step or a labeling step where a human is asked to generate the labels for training. Since DeepMatcher and Magellan are a part of the same overarching EM project, I resort

to using Magellan's overlap blocker to generate post-blocking pairs for DeepMatcher. I pre-generate the training, validation and test sets for DeepMatcher and input them along with the raw text files to enable the feature vector generation and subsequent classification. Similar to Magellan, one major limitation while adapting relational EM to spatial EM is that spatial attributes are treated as strings, that can lead to substandard performance which I empirically validate in Chapter 5.5. Additionally, in DeepMatcher the feature vector creation is done as a part of the package, so it prohibits equipping or amending the feature vector with spatial information, that can assist the model to perform adequately on spatial data.

3.2.3 QuadSky

Isaj *et al.* (2019) build QuadSky as an end-to-end spatial EM system that matches spatial entities comprising both spatial and relational attributes. QuadSky consists of spatial blocking, feature vector generation and classification as the steps in its EM pipeline similar to GEM. However, QuadSky proposes skyline-based heuristics instead of an ML classifier to label the spatial entity pairs as matching or not. Also, unlike traditional EM systems that have a left and a right dataset, QuadSky takes a single dataset as input upon which Cartesian product is computed to generate the pool of candidate entity pairs that need be matched. We union S_{left} and S_{right} into a single dataset and feed it to QuadSky. For example is if S_{left} and S_{right} have 300 and 400 tuples respectively, I create a new dataset S_{merged} with $300+400 = 700$ tuples and perform a self product of those 700 tuples. Hence ideally if the data was considered in two data sets the number of total pairs will be equal to the Cartesian product of $S_{left} \times S_{right}$ which is equal to $300 \times 400 = 120,000$, whereas for the QuadSky system

the total number of pairs will be 244,650.

The spatial system leverages the quadtree index in its blocking step to prune away the obvious non-matches. It creates feature vectors using four string similarity scores for relational attributes and spatial distance for *coordinates*. It uses a heuristic based on multiple skylines to discover the matches. QuadSky only matches *point* geometries. To adapt it for *polygon* matching, I reduce the polygon to a point with its *centroid* coordinates. Doing so leads to poor EM performance which we will show in Chapter 5.5. Note that I could not compare GEM to GeoBench or GeoAlign due to their restrictive web interfaces and lack of source code.

Chapter 4

SYSTEM ARCHITECTURE

This section presents the details of *GEM* and how it can match spatial entities with diverse geometry types.

4.1 Overview

Spatial entity matching is not as explored as relational entity matching; most of the state-of-the-art spatial EM systems only match tuples of geometry type *point*. Astonishingly, no investigation has been done in the area of EM of a polygon to another polygon or matching a point to a polygon geometry type tuple. Deduplicating data across various datasets, mapping social media check-ins from multiple platforms without the barrier of geometry type are just some of the many applications of this fundamental functionality. Analyzing the various state of the art entity matching algorithms, I understood that all the existing systems either read the spatial attributes as strings (relational EM) or need to be fed the polygon spatial coordinates in the form of a point, i.e., the centroid of the polygon (spatial EM). I propose a unique way to handle spatial EM on multiple spatial geometries with the assistance of Apache Sedona TheApacheFoundation (2020a); Yu *et al.* (2015, 2019). Apache Sedona is a

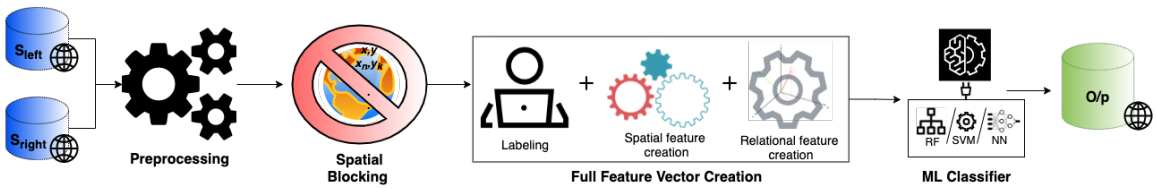


Figure 4.1: System Architecture Overview of *GEM*

cluster computing system that can load and process spatial SQL queries on big spatial data with efficiency and resilience. Furthermore, the expeditious and scalable nature of the system makes the feature vector creation process entirely scalable and swift.

Let us assume that we have two data sources S_{left} and S_{right} whose schema was manually pre-aligned, i.e., the matching attributes from the data sources were mapped to each other with the help of domain experts. It is required that the spatial coordinates of each entity pair are non-null, although the non-spatial attributes are allowed to have null values. The spatial blocking step prunes away the obviously non-matching pairs from the pool of Cartesian product pairs. Figure 4.1 gives an overview of the *GEM* system architecture. It consists of a pre-processing step which performs a sanity check to ensure that the entity pairs do not have unaligned attributes or null spatial coordinates. The clean pairs from the Cartesian product are then passed to a spatial blocking step which filters out the obviously non-matching pairs. This is required because, it avoids the feature vector generation and classification expenditure on such obvious mismatches. Furthermore, for spatial blocking, the geohashing technique is employed that works with the spatial features of the data. Geohashing is a flexible and efficient way of encoding coordinate information into a string of 12 characters Moussalli *et al.* (2015). Procedural details of spatial blocking will be explained in Chapter 4.2.

The post-blocking pairs, $S_{blocking}$, is passed to the feature vector creation module, which is divided into three parts as shown in Figure 4.1 - labeling, spatial feature creation and relational feature creation. While spatial feature creation uses Apache Sedona TheApacheFoundation (2020a) to infer the spatial proximity among the entities, relational feature creation augments the spatial features with relational features created using the Simmetrics TheApacheFoundation (2004) library to create the full feature vector. The ability to preserve and utilize the location information provided

in the data makes spatial EM different from relational EM; this is witnessed the most in the feature vector creation. Note that we are not reducing the polygon's coordinate information into a mere point because Apache Sedona is exceptionally equipped to handle polygons with multiple coordinates. We provide set $S_{blocking}$ and datasets S_{left} and S_{right} in their original form to Sedona and compute various spatial proximal queries TheApacheFoundation (2020b) depending on the type of spatial geometry EM scenario

The labelling of pairs from $S_{blocking}$ is done in a semi-manual fashion to create the ground truth labels for the feature vectors. I work on manually generating Boolean DNF (Disjunctive Normal Form) rules for matching through trial-and-error method on various samples of the data. The predicate rules are a conjunction of multiple relational and spatial conditions. These rules and their frequency can vary from dataset to dataset. An example rule for *point* $\times$ *point* dataset could look like '(Euclidean distance between 2 points $\leq 3meters \bigwedge$ Jaccard similarity between *name* attribute $\geq 0.9 \bigwedge$ Cosine similarity between *address* attribute $\geq 0.95) \bigvee$ (Jaccard similarity between *address* $\geq 0.95 \bigwedge$ Levenstein distance between *name* ≥ 0.98'). The outcome of such rules is evaluated on each pair and helps determine its label. These rules are approximate as they are based on manual examination of several samples, which is why extensive manual effort was spent to verify and correct the mis-labelled pairs. This label correction step is done by sampling an ambiguous space of entity pairs whose labels are likely to go wrong. Ambiguous ranges are defined based on a similarity score that is neither too less, nor too high (for instance, overall attribute similarity ≥ 0.75 but ≤ 0.9).

Simmetrics library comprises around 21 string similarity functions such as Jaccard similarity, Cosine similarity and others, each of which is applied to the pair of relational attributes to generate similarity scores. These scores are used as non-spatial

feature dimensions in the numerical feature vector. Apache Sedona TheApacheFoundation (2020a) is a cluster computing system that provides a host of spatial SQL operators to support distributed query processing over big spatial data and can support diverse spatial geometries. In this work, seven such query operators from Apache Sedona are utilized that can compute spatial proximality between a pair of spatial geometries. These include ST_Contains, ST_Intersects, ST_Within, ST_Equals, ST_Crosses, ST_Touches and ST_Overlaps, each of which consumes a pair of spatial geometries and outputs a Boolean value indicating whether the spatial proximity constraint met or not. Spatial proximity can be understood as the equivalent metric for spatial similarity, analogous to how string similarity is used in the context of relational EM. The Boolean output from these proximal evaluation are used as numeric dimensions (1/0 for constraint met or not) in the feature vectors. Spatial feature dimensions augmented with non-spatial feature dimensions produce the final feature vector that is then passed to the machine learning models for training and evaluation. More details about feature vector creation are provided in Section 4.3.

Three classifiers are employed to match the feature vectors: Random Forests, SVM, and feed-forward neural networks. This is possible because the matcher (classifier) in *GEM* is a pluggable component that can support any binary classifier. This allows for plug and play of ML models in a seamless fashion for extensive experimental adaptation. All the models are trained on 80% of the feature vectors and are evaluated on the remaining 20%. Further in this chapter, I talk in detail about the system and highlight how every step will adapt, and procedure will take place in case of each of the three spatial geometry entity matching scenarios stated in Chapter 3.

4.2 Spatial Blocking

Blocking function determines how many spatial entity pairs within the Cartesian product can be pruned away. Intuitively, this implies that there is an implicit tradeoff between the pruned pairs and matching quality. If too many pairs are pruned out, there may be several matching pairs among them which were deemed as non-matching and this will lead to an increase in False Negatives. Conversely, if the blocking function is too conservative and prunes too few pairs, there will still be several pairs remaining among the post-blocking pairs and although this may not deteriorate the quality of the classifier, it will have an adverse affect on the feature vector creation, training and test latencies of the classifier. Therefore, it is important the blocking function can prune just enough pairs to speed up the execution of the matching phase.

Another issue here is the choice of the blocking function. Given that there are several spatial SQL operators in Apache Sedona, we could have picked a combination of their Boolean evaluations as the blocking function. However, combining the outcomes of several Boolean predicates (such as ST_Contains, ST_Intersects, ..) is a non-trivial exercise and has an exponential search space in terms of the number of combinations of the spatial SQL operators. Therefore, instead I resort to using geohash which is a lightweight indexing technique, as a blocking function. I call the blocking function *GeoPrune*. It is intuitive knowledge that the granularity of the blocking function can substantially influence evaluation measures, the quality of resultant matching pairs, and the standard of eliminated pairs. Nevertheless, what is equally important is that the spatial blocking function needs to be lightweight, supple, and inexpensive in terms of processing. The most intuitive and greedy approach would be to relay all the p possible pairs and perform a combination of various spatial proximal queries like ST_Touches, ST_Overlaps, ST_Contains, ST_Intersects, and others avail-

able in Apache Sedona. The resultant pairs that survive the processing will count towards the post-blocking candidate set C. This method indeed preserves the essence of the system (using spatial features for blocking) but **1.** SQL query computation is inherently more expensive on the processor than string matching (Geohash code). Furthermore, the different proximal query results need to be combined with the help of boolean functions before they can be embedded to the feature vector. For instance using the seven spatial operators for a *point* $\times$ *polygon* tuple pair then one possible Boolean combination is $ST\_Touches \cap ST\_Overlaps \cup ST\_Contains \cup ST\_Intersects \cap ST\_Within \cup ST\_Crosses \cup ST\_Equals$ **2.** It is practically impossible for humans to compute an efficient Boolean relation, in comparison to what an ML model can enumerate. The Boolean relations that we, as humans, can compute are far inferior to what a machine learning model can enumerate. the proximal query Undoubtedly this approach will yield a more decadent post-blocking candidate set with a higher ratio of negative:positive pairs. However, the arguments mentioned above propelled us to adopt an alternative method.

Algorithm 1 describes how GeoPrune works. It consists of two steps - geohash computation and blocking. Geohash is an existing geocoding system that encodes the geographic location of a point, i.e., latitude and longitude Wikipedia contributors (2020). It is a short string of alphanumeric characters which depicts an area on the map. Geohash assumes that the globe is partitioned into a hierarchical grid of 32 cells and annotates them using Base-32 encoding. The first character of the code uniquely identifies a coarse-grained region (cell) on the grid. Each cell is hierarchically divided into 32 cells of a finer granularity (fanout=32) with a maximum allowed tree height of 12, i.e., at most 12 levels can exist in the hierarchy. Hence, a geohash code for a location can have maximum of 12 characters. The more the characters in a geohash, the more precise is the location GeohashPubnub (2020).

Algorithm 1: GeoPrune (Spatial blocking)

Input 1: Two spatial datasets S_{left} and S_{right}

Input 2: blocking threshold k

Output: Post blocking candidate set $S_{blocking}$

```
1  geoPrune:
2      S'_left ← computeGeohash(S_left)
3      S'_right ← computeGeohash(S_right)
4      S_blocking ← blocking(S'_left, S'_right, k)
5  return S_blocking
6  def computeGeohash(S_data):
7      if S_data geometry type Polygon then
8          // compute centroid
9          S_data ← centroid(polygon's coordinates)
10     end if
11     foreach tuple t in S_data do
12         // point coordinates for Point
13         // centroid coordinates for Polygon
14         S'_data ← geohash code using coordinates
15     end foreach
16 return S'_data
```

Algorithm 2: GeoPrune (Spatial blocking)

```
17 def blocking(S'_left, S'_right, k):
18     foreach tuple i ← S'_left do
19         foreach tuple j ← S'_right do
20             if initial k characters of i's and j's geohash codes are same then
21                 // qualify pair to post blocking
22                 S_blocking ← (i, j)
23             end if
24         end foreach
25     end foreach
26 return S_blocking
```

Consider a spatial entity pair (s_1, s_2) with geohash codes G_{s_1} and G_{s_2} respectively. We determine a granularity, k, up until which we will be matching the geohash codes of a given pair. For instance, if k=5 then, every spatial entity pair whose initial 5 out of 12 characters in the geohash codes G_{s_1} and G_{s_2} are same, will qualify to the post-blocking candidate set $S_{blocking}$ as shown in Algorithm 1. For example, (G_{s_1}, G_{s_2}): (**9tbqh**gn36wp7, **9tbqh**1ma6xz5) will qualify, in contrast to another pair (G_{s_1}, G_{s_2}): (**9tbqh**gn36wp7, **9tbq4**1ma6xz5) whose prefixes are different. This granularity measure, k, is different for different datasets. As mentioned before, if the value of k is too high (close to 12), we prune away too many pairs, thereby inducing too many False Negatives. On the other hand, a lower value of k (close to 1) prunes too few pairs, which in turn results in inefficiency and a slower execution of the EM pipeline. The discussion about a precise setting for k is deferred to Chapter 5.

Computing the geohash code is fairly straightforward for *points*. In the case of

polygons, the geohash code of the *centroid* is used for spatial blocking as shown in Algorithm 1. Reducing the polygon to a point during blocking is a convenient approximation which will not lead to a high inaccuracy, as compared to doing so during the actual matching phase. This is because, blocking applies this approximation only to filter out the obvious non-matches. For example if the centroids of two polygons are reasonably far, this can be detected by $GeoPrune$, that prunes away such polygon pairs. While just because two centroids are reasonably close, deeming them to be matching can lead to inaccuracies. Note that if a spatial entity pair qualifies blocking, it means that this pair is not an obvious mismatch. Whether this pair is a match or not can only be determined after creating its feature vector and using a sophisticated classifier to predict its label. Concentrating the polygon's coordinate information in a centroid will help make the spatial blocking function lightweight, swift and efficient. Moreover, it has been observed empirically that shrinking the polygon to a centroid does not greatly affect the accuracy as far as it is confined to the spatial blocking step and not done for feature vector creation or classification. Once I perform the spatial blocking step, the candidate set, $S_{blocking}$, of post-blocking pairs is ready. Next the semi-manual labelling of the post blocking pairs takes place followed by creation feature vectors for the pairs in $S_{blocking}$ and then classification as a *match* or *non-match* using a binary classifier.

4.3 Feature Vector Creation

Once the set of post-blocking pairs, $S_{blocking}$, is obtained I generate the ground truth in a semi-automated fashion (refer Section 4.1) by using human intervention for the datasets that do not have their own ground truth. Thereby, we have a labelled post-blocking candidate set $S_{labelled}$, and datasets S_{left} and S_{right} with pre-aligned schema. This section discusses the creation of feature vectors (FV) for the spatial

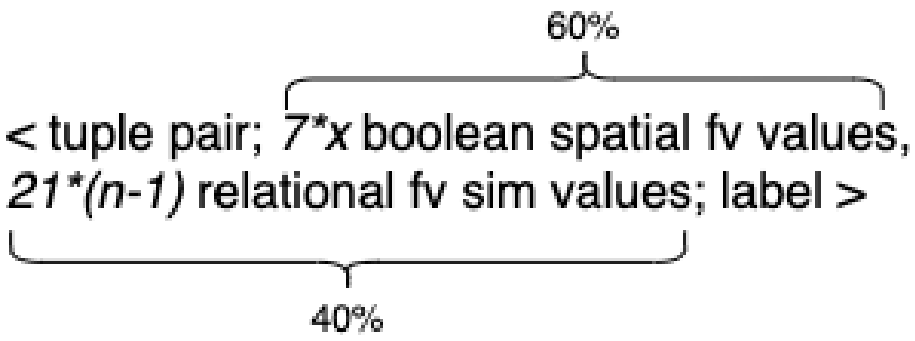


Figure 4.2: Feature Vector Skeleton

entity pairs in $S_{labelled}$. This step is the most salient functioning of the entire system as it generates the spatial dimensions in a geometry-aware fashion. The prime focus of the proposed EM algorithm is to use the spatial information of various geometries in their genuine form and minimize information loss; in this step, it is fulfilled the most. Apache Sedona offers support for diverse geometries including polygons with multiple coordinates; hence I do not have to resort to approximations such as reducing the polygon coordinates to a point. Instead, I apply the seven spatial SQL operators individually to the geometry pair in order to derive their spatial proximity. The seven Boolean dimensions are each depicted in the full feature vector as numbers 1 for qualifying the operator's constraint and 0 for not qualifying. Note that the spatial operators such as ST_Contains, ST_Within etc., are fully aware of the geometry type of the objects whose proximity is being estimated. These spatial operators provide proximality approximation information like polygon contains point, point intersects polygon, polygon touches polygon, and much more with the help of various proximal SQL queries offered by Apache Sedona. Similarity scores like Jaccard similarity, Cosine similarity and others preserves the similitude of attributes, proximity measures i.e. the output of the proximal SQL queries like ST_Contains, ST_Within and more, preserves the spatial knowledge of an entity tuple pair. These spatial dimensions

are further augmented with the relational attribute similarity evaluations obtained using the Simmetrics library, to generate the full feature vectors as shown in Algorithm 3. As mentioned earlier, we concentrate the polygon to a centroid only for the spatial blocking step because we have observed empirically that it does not affect the performance of system.

Let us assume that we are provided with a schema pair of data sources S_{left} and S_{right} which has a total of n attributes, out of which one attribute is spatial, i.e., it consists of coordinates. The other *n-1* are relational (non-spatial) attributes. As illustrated in Algorithm 3 the feature vector is a concatenation of the spatial and relational dimensions. The spatial attribute i.e. *coordinates* is processed through all the proximality operators TheApacheFoundation (2020b) that Apache Sedona offers. The output of these seven operators is represented as Boolean dimensions in the feature vector. We put a '1' if the proximal operator evaluates to TRUE and '0' if it evaluates to FALSE. The relational attribute similarity evaluation is done by computing 21 string similarity scores like Jaccard similarity, Cosine similarity, Levenshtein distance and more using the Simmetrics TheApacheFoundation (2004) library and normalizing the values to lie between 0 and 1.

In order to assert the dominance of the spatial information and to bias the classifier label more towards the spatial dimensions in the feature vector, I replicate the seven Boolean spatial feature dimensions by a replication constant x. This creates a ratio of 60:40 for spatial:relational values in the final feature vector as shown in Figure 4.2. I also append the ground truth to the feature vector. For example consider we have 4 attributes, n = 4, among which 1 is spatial (coordinates) and 3 are relational (name, address, category). In the feature vector, we have $21 \times (n-1) = 63$ similarity values for relational attributes and 7 (# proximality operators) values from spatial attributes. To assert spatial dominance, I replicate each of the 7 operator outputs approximately

Algorithm 3: Feature Vector creation

Input 1: Two spatial datasets S_{left} and S_{right}

Input 2: labelled post-blocking set $S_{labelled}$

Output: Set of feature vectors S_{fv}

```
1  fullFVCreation:
2      S_fv ← spatialFeatureCreation(S_left, S_right, S_labelled) +
         relationalFeatureCreation(S_left, S_right, S_labelled)
3  return S_fv
4  def relationalFeatureCreation(S_left, S_right, S_labelled):
5      load non-spatial attributes of pairs in S_labelled from S_left and S_right
6      compute 21 different string similarity metrics
7      normalized attribute similarity scores
8      S_fv ← relational feature values
9  return S_fv
10 def spatialFeatureCreation(S_left, S_right, S_labelled):
11     load S_left, S_right and S_labelled in Apache Sedona
12     execute all spatial proximal opreators on coordinates
13     convert output to boolean features
14     mutiply boolean features by replication factor x to assert 60:40
15     S_fv ← spatial feature values
16 return S_fv
```

$x = 14$ times. Hence in the final vector we will $7 \times 14 = 98$ spatial attributes values, 63 relational similarity values and 1 dimension for 1/0 (match/non-match) label as shown in Figure 4.2.

Information about datasets, spatial blocking threshold k, experimental results, and various other details are deferred to Chapter 5.

4.4 Classification

Classification is the final step of our spatial EM pipeline. We offer three types of classifiers that are easily pluggable into GEM. They all take the same resource input F_v, i.e., the feature vector shown in Figure 4.2 and provide a prediction output of 1 for matching tuple pair and 0 for non-matching pair. We borrow the implementations of the classifiers' supervised variant for linear classifier: Support Vector Machine (SVM), tree-based classifier: Random Decision Forests (RF), and non-linear classifier: Neural Nets (NN) from Meduri *et al.* (2020). The NN implementation adopts a simple feed forward neural network with a single hidden layer and uses Apache SystemDS TheApacheFoundation (2015) for execution. I train the NN for 100 epochs. Regularization techniques like batch normalization and dropout regularization assisted in making the neural network stable. A feature vector, F_v from S_{fv}, holds information about a tuple pair's spatial and non-spatial features and its ground-truth label which is held out from the classifier at test time. Random decision trees and SVM use the Weka WaikatoUniversity (2020) library for implementation. We use an ensemble of 20 decision trees in our Random forest classifier.

Entity matching is known to suffer from label skew (fewer available positive labels or matches as compared to negative labels or non-matches). Hence, it becomes imperative to have a cost-sensitive classifier that can handle the bias well to produce more meaningful and precise results. Therefore, the SVM implementation considers

a 2×2 dimensional cost matrix similarly to Meduri *et al.* (2020), in order to record the penalties associated with the detection of True Positives, True Negatives, False Positives and False Negatives as the four entry values in the matrix. These penalties are used at training time to inform the classifier that it needs to pay more attention to avoiding a specific type of mis-classification errors. While True Positives and True Negatives are not penalized, I assign a higher penalty to False Negatives than False Positives to encourage finding more matches, as they are fewer in number. Cost-sensitive classifiers have proved to work better in EM pipelines than normal classifiers Ji and Carin (2007); Turney (1994); Parambath *et al.* (2014). The cost matrix trains the classifier in a skew-aware manner, thereby enhancing its accuracy.

Chapter 5

EXPERIMENTAL EVALUATION

This section discusses the experimental results. I start by providing details about the setup and datasets. Then, discuss the performance of various classifiers in GEM, evaluate the *GeoPrune* blocking mechanism w.r.t. variation in k (the spatial blocking threshold), and then compare GEM against the baselines from Section 3.2.

5.1 Experimental Setup

The experiments were conducted on an Intel Xeon E5-2687WV4 CPU (12 cores, 3.0 GHz per core) machine with 100 GB RAM and a 4 TB hard drive. I used Java 1.8, Scala 2.12 and Python 3.7 for implementation. I also installed Apache Sedona 0.1.0 along with Apache Spark 3.0.1 and Apache Hadoop 2.7.2. To calculate the geohash codes of a location, the *pygeohash* package PyPi (2016) was utilized.

5.2 Datasets

All the datasets assume that the spatial attribute (coordinates) is not null. Table 5.1 shows the left and a right tables, # pairs in the Cartesian product and # post-blocking pairs in each dataset. There are four restaurant datasets for the *point*$\times$*point* scenario. None of the datasets originally had location coordinates; so I auto-generated them from the given address using the 'googlemaps' package PyPi (2021). The left and right datasets of Zomato1-Yelp1 (R1), Zomato2-Yelp2 (R2), and Yelp-Yellow Pages (R3) are borrowed from Konda *et al.* (2016)'s data repository and Fodors-Zagats (FZ) from Tejada (2003). In the *point* $\times$ *polygon* scenario, there is an Yelp-OSM dataset in which Open Street Map (OSM) OSM-Data (2021) data contains polygon coordinates

Dataset		#left	#right	#total pairs	#post-blocking
Point-Point	Fodors-Zagats (FZ)	533	331	176.4k	1314
	Zomato1-Yelp1 (R1)	3013	5882	17.7M	3063
	Zomato2-Yelp2 (R2)	7689	4055	31.1M	6864
	Yelp-Yellow Pages (R3)	9947	28787	286.3M	30912
Polygon-Point	Yelp-OSM	4979	60803	302.7M	229377
Polygon-Polygon	AZ-Maricopa	4979	3357	16.7M	35864

Table 5.1: Details of the Spatial Datasets

and the Yelp data Kaggle (2021) contains the point coordinates for various business establishments in Arizona. In the case of *polygon* × *polygon*, the left dataset contains polygon coordinates for buildings in Arizona and the right dataset contains polygon coordinates for buildings in the Maricopa County in Arizona. While FZ dataset has its own ground truth Tejada (2003), the ground truth for other datasets is determined using the semi-manual technique described in Section 4.1. I pre-align the schemata of the left and right datasets with the help of a domain expert.

The labeled post-blocking entity pairs is split into 80% train and 20% test. The most important factor that is considered before splitting a spatial dataset is maintaining spatial equivalence (maintaining the distribution of tuple pairs in each region) across the train and test sets. I also maintain the post-blocking pairs' class skew (ratio of matching to non-matching tuple pairs) in the train-test splits. The following subsections provides details on the attributes of the datasets and further pre-processing details.

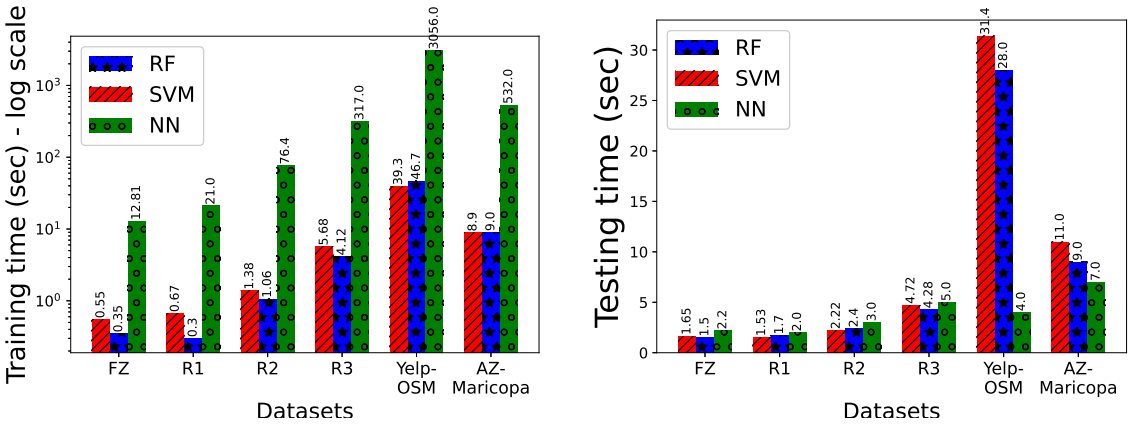
5.2.1 Point × Point

The four restaurant datasets considered for the *Point* × *Point* experiment have both relational and spatial attributes. The left and the right data set came with only

relational attributes when borrowed from Konda *et al.* (2016)'s data repository. I auto-generated the latitude and longitude information using the 'googlemaps' python package. While the Fodors-Zagats data set came with its own ground truth provided by Tejada (2003), I generated the ground truth for all the other datasets using a semi-manual mechanism. The ground truth generation is basically a disjunction of several conjunctive predicates which evaluate string similarity distance and spatial distance between the given data features. All the four datasets have different number of total tuples in the left and right tables and different number of attributes. The Fodors-Zagats (FZ) dataset originally has 6 attributes and I add the coordinates making it 7 features namely: *id, name, addr, city, phone, type, coordinates*. All the $Point \times Point$ sets are perfect oracles with no null values, as Magellan Konda *et al.* (2016) cannot handle null values. Out of the other three datasets Zomato1-Yelp1 (R1) originally has 4 attributes and in total it has *id, name, phonenumber, address, coordinates* as attributes. The Zomato2-Yelp2 (R2) has a total of 11 attributes namely *id, name, votes, rating, phone, address, city, state, zip, cuisine, coordinates*. Lastly the Yelp-Yellow Pages (R3) dataset has a total 10 attributes: *id, name, address, telephone, website, priceRange, category, ratingValue, neighborhood, coordinates* .

5.2.2 Point × Polygon and Polygon × Polygon

The $Point \times Polygon$ and $Polygon \times Polygon$ datasets have been carefully generated to encompass various building and establishments of the Arizona state. While Yelp datasets consists of details about restaurants, businesses and establishments in a 'Point' geometry type, OSM provides multi-vertex polygon coordinates for businesses and buildings in Arizona. For the $Point \times Polygon$ experiment I try and find matches between the Yelp datasets and OSM dataset. Both the $Point \times Polygon$ and $Polygon \times Polygon$ datasets have total 5 attributes namely: *id, name, address, cate-*



(a) Training latency

(b) Testing latency

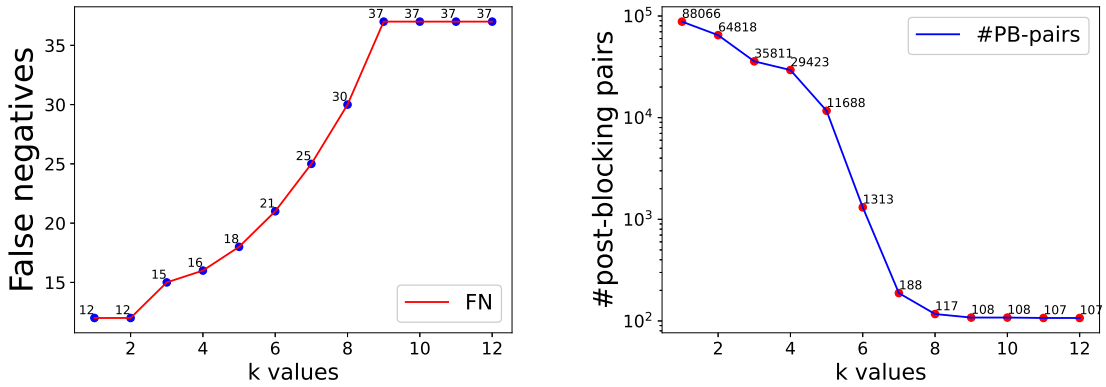
Figure 5.1: Latency Comparisons of the Three Classifiers Across All the Datasets

*gories, coordinates .Whereasforthe*Polygon $\times$*Polygon* experiment I process the two OSM datasets. While the left dataset is information about the business in Arizona, the right dataset is establishments and business in the Maricopa County in Arizona. The fact that both the datasets are from OSM make the manual process of schema alignment easier. Moreover their is much more homogeneity in the representation of the location information as both the geometry data types are same (polygon).

5.3 Comparison of Classifiers

Datasets		Random Forest			SVM			Neural Network		
		Precison	Recall	F1-Measure	Precision	Recall	F1-Measure	Precision	Recall	F1-Measure
Point-Point	FZ	1	1	**1**	1	1	**1**	1	1	**1**
	R1	0.967	0.967	0.967	0.948	1	**0.973**	0.88	1	0.93
	R2	0.99	0.993	**0.992**	0.973	0.993	0.983	0.932	1	0.965
	R3	0.983	0.989	**0.986**	0.969	0.997	0.983	0.908	0.997	0.95
Point-Polygon	Yelp-OSM	0.96	0.972	**0.966**	0.906	0.987	0.944	0.691	1	0.817
Polygon-Polygon	AZ-Marizopa	0.997	0.99	**0.993**	0.971	0.983	0.977	0.901	0.99	0.95

Table 5.2: Test Performance Evaluation of GEM Across Various Classifiers and Datasets



(a) # False Negatives

(b) # Post-blocking Pairs

Figure 5.2: Evaluation of Random Forest Classifier for Various Values of K in Fodors-zagats Dataset

Three pluggable ML classifiers are provided for the classification step in *GEM*. Here I compare the performance of Random decision Forests with an ensemble of 20 trees, SVM with a cost-sensitive matrix and a feed-forward neural network with a single hidden layer (see Section 4.4 for details on the classifiers). In the cost matrix, there are no penalties for True Positives and True Negatives but for False Positive:False Negative I exercise a penalty ratio of 1:4. I fix the spatial blocking threshold to an appropriate value, k, which is optimal for that dataset. Results to illustrate the importance of blocking threshold k are shown in Section 5.4. Figure 5.1 shows the time taken by each classifier for training and testing on different datasets. The training time for RF and SVM are almost the same, while NN takes more time. Table 5.2 contains the values of test precision, recall and F1-measure for all datasets over the three classifiers. I observed that F1-measure for the *point* $\times$ *point* datasets over all the classifiers ≥ 0.95. Additionally, RF and SVM are highly accurate with F1-scores of 0.97 and above. The reported results are the best numbers from 5 consecutive executions. All the executions varied with an error rate of 10^{-5}. For *point* $\times$ *polygon*

and *polygon* × *polygon* cases as well, both RF and SVM have similar F1-scores. RF works with a committee of 20 relatively uncorrelated decision trees that captures the nonlinear dependencies among attributes. Unlike SVM, RF can handle non-separable cases effectively.

It is surprising that SVM being a linear classifier performs at par with RF which indicates that the data has very less noise possibly due to effective blocking procedure. Moreover, the cost matrix helps SVM to learn and adapt to class skew. Neural networks in general take longer to train as compared to RF and SVM. Even though the NN I adapt has a simple architecture, it produces competitive scores for most of the datasets and outperforms the more involved NN like Mudgal *et al.* (2018) in both F1-score and test latency (see Figure 5.1b for NN test latency). I will delve into the comparative study of our system against baselines in the later subsections.

5.4 Evaluation of Blocking

As discussed in section 4.2, the spatial blocking mechanism, *GeoPrune*, is used to prune away the obvious non-matches to reduce the search space for mapping same locations. I compute the geohash codes of the locations to assist in the blocking step. While computing the geohash code for a point is fairly simple, it is non-trivial for a polygon. For a polygon, the centroid of the polygon is used for the spatial blocking algorithm (see Section 4.2). Geohash is a 12 character code which is unique for every location on the Earth. Given a blocking threshold, k, I prune away all those pairs whose first k out of 12 characters do not match. This section studies the influence of k on performance of the system. Note that this retrospective evaluation of k is only for empirical purposes. This is because, spatial blocking happens before feature vector creation and choosing an optimal k value is typically done by using validation sets upon which several values of k are tried and evaluated. The process of deciding

an appropriate threshold value for each datasets goes through various iterations and analysis of factors such as latency and class skew. I use the same method to determine k for all datasets. In this subsection, I chose FZ dataset to discuss the effect of k (see Figure 5.2) because it provides its own ground truth.

As per the discussion and results in Section 5.3 Random Forest classifier proves to be accurate and efficient for majority of the datasets. Hence to study the best threshold k for the FZ dataset I will fix RF as the classifier. I use the 112 matching pairs out of 176,423 total Cartesian product pairs provided by Tejada (2003) while labelling the ground truth. While choosing an appropriate threshold k, the most crucial step is to pick a value which is neither too low nor too high. A low value will allow a lot of pairs into the post-blocking set thereby increasing training and test latency whereas, a high value will qualify very less pairs thereby leading to a lot of False Negatives (FN) over the Cartesian product i.e matches deemed to be mismatches and matches that are missed out. We study only the #FN here because the other paramters of the Confusion matrix (i.e. FP, TP and TN) do not get affected by the quality of post-blocking set. As mentioned earlier, the reason to show this valuation is to help understand the effect of blocking threshold k on final test F1-score. In actual experiments the selection of k is done manual using trail and error method on holdout sets from training data before the feature vector creation step. Figures 5.2a and 5.2b respectively shows the number of FNs and number of post-blocking pairs for all values of k (1-12). As it can be observed, with increase in the value of k the #FNs also increases which leads to decrease in Recall. While the #post-blocking pairs decrease with the increase in k which implies that low values of blocking threshold k will lead to high training and test latencies. Hence it is empirically deduced that the most optimal blocking threshold for the FZ dataset is k=*6*, which produces test F1-measure of 1.0 (see table 5.2).

5.5 Comparison with Baselines

I will be comparing the performance of *GEM* against various state-of-the-art baseline systems. Discussion on how each of these systems have been adapted has already been provided in section 3.2. While Magellan Konda *et al.* (2016) and QuadSky Isaj *et al.* (2019) are implemented as end-to-end EM systems with their own blocking mechanism, Deepmatcher Mudgal *et al.* (2018) is only a classifier. Hence for it I use post-blocking pairs generated by Magellan Konda *et al.* (2016). Figure 5.3 provides the results of all the systems for various spatial datasets. Alongside comparing F1-scores, we also discuss the scalability of the systems in this section.

The 'percentage of data' column in Tables 5.3 5.4 5.5 indicates the maximum fraction of the left and right datasets that a system is able to handle. 100% implies it was able to process the entire set of pairs in the Cartesian product, while 20% implies that it was only able to handle a maximum of 20% of the left and 20% of the right dataset before the system gave timeout or out-of-memory errors.

All the methods perform well for the FZ dataset in *point* $\times$ *point* scenario. While *GEM* performs the best on *point* $\times$ *point* dataset, Konda *et al.* (2016), Mudgal *et al.* (2018) and Isaj *et al.* (2019) have an F1-score of 0.983, 0.84 and 0.70 respectively for the Fodors-Zagats dataset (refer Table 5.1). Although, the baseline systems are not able to sustain their performance for the datasets of the other two spatial EM scenarios. While Magellan Konda *et al.* (2016) is able to scale for the entire set of pairs in the Cartesian product, due to its coarse post-blocking set and treating spatial attributes as strings, it scores a final F1-measure of only 0.88 and 0.905 for the Yelp-OSM and AZ-Maricopa datasets respectively. On the other hand, DeepMatcher Mudgal *et al.* (2018) being a complex neural network classifier is not able to scale for either *point* $\times$ *polygon* or *polygon* $\times$ *polygon* datasets, which is illustrated by 'TimeOut'

in Figure 5.3. It is only able to process 0.04 of the Yelp-OSM data and performs poorly with an F1-score of 0.0221 (see Table 5.4), mainly because it treats every attribute as long-text and the geometric diversity in terms of point and polygon coordinates only makes the matching process more sub-optimal. Though it can only scale upto 0.25 of the *polygon* × *polygon* dataset (AZ-Maricopa) it performs relatively better with an F1-score of 0.77 (see Table 5.5) because both left and right datasets have a uniform representation of polygon's spatial coordinates that allows for textual matching to work occasionally.

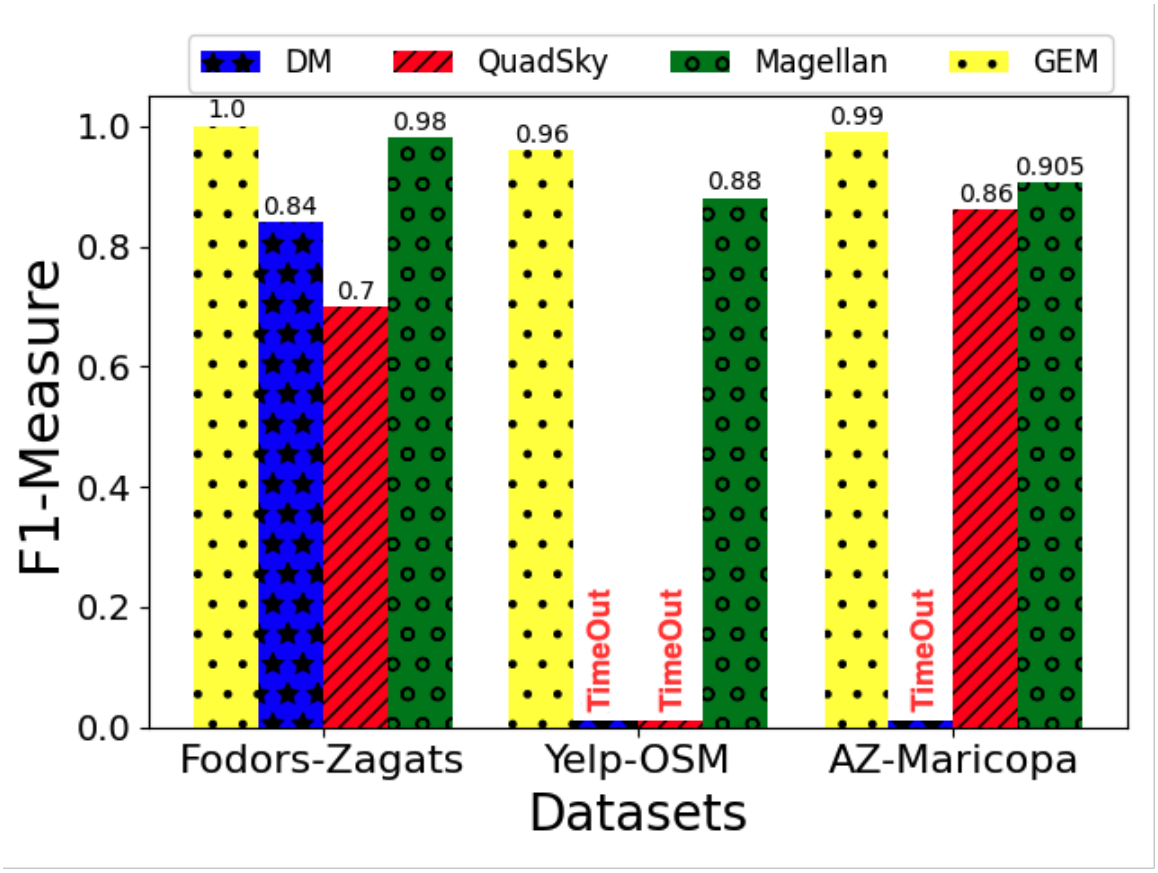


Figure 5.3: Comparison of GEM with Baselines Across 3 Spatial Datasets with Different Geometries

QuadSky Isaj *et al.* (2019) is an end-to-end spatial EM system, but as mentioned

Approaches	percentage of data	#total-pairs	#post-blocking	F1-measure
Deepmatcher	$100\%(left) \times 100\%(right)$	176.4k	450	0.847
QuadSky	$100\% \times 100\%$	372.8k	7746	0.70
Magellan	$100\% \times 100\%$	176.4k	450	0.983
GEM	$100\% \times 100\%$	176.4k	1314	**1**

Table 5.3: Comparison with Baselines for Point-point Case

Approaches	percentage of data	#total-pairs	#post-blocking	F1-measure
Deepmatcher	$20\%(left) \times 20\%(right)$	12.1M	123.7k	0.0201
QuadSky	$15\% \times 15\%$	48M	3878	0.66
Magellan	$100\% \times 100\%$	302.7M	3.11M	0.88
GEM	$100\% \times 100\%$	302.7M	229.3k	**0.966**

Table 5.4: Comparison with Baselines for Point-polygon Case

Approaches	percentage of data	#total-pairs	#post-blocking	F1-measure
Deepmatcher	$50\%(left) \times 50\%(right)$	9.4M	129.06k	0.77
QuadSky	$100\% \times 100\%$	34.7M	3436	0.86
Magellan	$100\% \times 100\%$	16.7M	225.4k	0.905
GEM	$100\% \times 100\%$	16.7M	35864	**0.99**

Table 5.5: Comparison with Baselines for Polygon-polygon Case

in sec 3.2 it takes a single dataset as input and does self-product to create the total pairs as depicted in the '#total-pair' column. Hence it can be noticed in table 5.3 that all the systems support 100% of the data but '#total-pairs' for QuadSky is 372.8k while for others it is 176.4k. For the *point* × *point* case, the QuadSky system scores an F1-measure of 0.70 while GEM performs 42% better with an F1-score of 1.0.

The state-of-the-art end-to-end spatial EM system, QuadSky Isaj *et al.* (2019), provides support only for the *point*×*point* data and it suffers due to the approximation of a *polygon* to its centroid (point). Doing so in the crucial step of feature vector creation results in significant information loss and compromised F1-scores. Hence for the AZ-Maricopa dataset, QuadSky produces and F1-score of 0.86 (see Table 5.5) while GEM performs 15% better with an F1-score of 0.99. The state-of-the-art spatial EM system is not able to scale for the Yelp-OSM dataset and hence is denoted by a 'TimeOut' in Figure 5.3. It is only able to process 0.0225 of this dataset and produces an F1-score of 0.66 (see Table 5.4). As it can be observed from Figure 5.3 GEM performs the best under all the 3 scenarios (see Section 3.1). It provides the best trade-off between precision and recall, and the best F1-scores of 1, 0.966 and 0.993 for the three datasets respectively.

Chapter 6

CONCLUSION AND FUTURE WORK

In this dissertation, I proposed an end-to-end Geospatial EM system *GEM* that can match spatial entities of diverse geometries such as *point* and *polygon*. *GEM* provides EM support for three scenarios: *point* × *point*, *point* × *polygon* and *polygon* × *polygon*. The proposed lightweight and efficient *GeoPrune* blocking mechanism uses geohash codes and a blocking threshold to prune the obviously non-matching pairs. Next, for the feature vector creation we use Apache Sedona's capabilities to create spatial feature dimensions that capture the proximity between a geospatial entity pair. Lastly, for the classification we provide three machine learning classifier: Random Forest, SVM and Neural network. Experiments on varying the threshold, k, for the *GeoPrune* blocking mechanism established the tradeoff between quality and latency. While lower values of k resulted in high latencies due to more post-blocking pairs, higher values of k resulted in several False Negatives. The comparison of classifiers revealed that RF and SVM prove to be the most efficient. The experiments on large-scale datasets showed that *GEM* achieved F1-scores of 1, 0.96 and 0.99 for FZ, Yelp-OSM and AZ-Maricopa datasets respectively, emphasizing that the system providing native support for diverse geometry types can outperform geometry-agnostic spatial EM baselines. Possible directions for future work include extending *GEM* to complex spatial objects such as multi-polygons, handling null spatial attributes, and transfer learning for cross-domain EM where a matcher learned from a label-rich source domain is applied to a label-scarce target domain.

REFERENCES

Balkić, Z., D. Šoštarić and G. Horvat, "Geohash and uuid identifier for multi-agent systems", in "KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications", pp. 290–298 (Springer, 2012).

Balley, S., C. Parent and S. Spaccapietra, "Modelling geographic data with multiple representations", International Journal of Geographical Information Science **18**, 4, 327–352 (2004).

Bansal, N., A. Blum and S. Chawla, "Correlation clustering", Machine learning **56**, 1-3, 89–113 (2004).

Barlaug, N. and J. A. Gulla, "Neural networks for entity matching: A survey", ACM Transactions on Knowledge Discovery from Data (TKDD) **15**, 3, 1–37 (2021).

Barret, N., F. Duchateau, F. Favetta and L. Moncla, "Spatial entity matching with geoalign (demo paper)", in "Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems", pp. 580–583 (2019).

Berjawi, B., E. Chesneau, F. Duchateau, F. Favetta, C. Cunty, M. Miquel and R. Laurini, "Representing uncertainty in visual integration.", in "DMS", pp. 365–371 (2014).

Christen, P., "The data matching process", in "Data Matching", pp. 23–35 (Springer, 2012).

Christen, P., T. Churches and M. Hegland, "Febrl–a parallel open source data linkage system", in "Pacific-Asia Conference on Knowledge Discovery and Data Mining", pp. 638–647 (Springer, 2004).

Cochinwala, M., V. Kurien, G. Lalk and D. Shasha, "Efficient data reconciliation", Information Sciences **137**, 1-4, 1–15 (2001).

Domingos, P., "Multi-relational record linkage", in "In Proceedings of the KDD-2004 Workshop on Multi-Relational Data Mining", (Citeseer, 2004).

Elmagarmid, A. K., P. G. Ipeirotis and V. S. Verykios, "Duplicate record detection: A survey", IEEE Transactions on knowledge and data engineering **19**, 1, 1–16 (2006).

FacebookOpenSource, "Fasttext package", URL `https://fasttext.cc/` (2020).

GeohashPubnub, "Geohash", URL `https://www.pubnub.com/learn/glossary/what-is-geohashing` (2020).

Getoor, L. and A. Machanavajjhala, "Entity resolution: theory, practice & open challenges", Proceedings of the VLDB Endowment **5**, 12, 2018–2019 (2012).

Isaj, S., E. Zimányi and T. B. Pedersen, "Multi-source spatial entity linkage", in "Proceedings of the 16th International Symposium on Spatial and Temporal Databases", pp. 1–10 (2019).

Jaro, M. A., "Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida", Journal of the American Statistical Association **84**, 406, 414–420 (1989).

Ji, S. and L. Carin, "Cost-sensitive feature acquisition and classification", Pattern Recognition **40**, 5, 1474–1485 (2007).

Kaggle, "Kaggle yelp dataset", URL `https://www.kaggle.com/yelp-dataset/yelp-dataset` (2021).

Karam, R., F. Favetta, R. Kilany and R. Laurini, "Integration of similar location based services proposed by several providers", in "International Conference on Networked Digital Technologies", pp. 136–144 (Springer, 2010).

Konda, P., S. Das, P. Suganthan GC, A. Doan, A. Ardalan, J. R. Ballard, H. Li, F. Panahi, H. Zhang, J. Naughton *et al.*, "Magellan: Toward building entity matching management systems", Proceedings of the VLDB Endowment **9**, 12, 1197–1208 (2016).

Köpcke, H. and E. Rahm, "Frameworks for entity matching: A comparison", Data & Knowledge Engineering **69**, 2, 197–210 (2010).

Landau, G. M. and U. Vishkin, "Fast parallel and serial approximate string matching", Journal of algorithms **10**, 2, 157–169 (1989).

Liu, J., H. Li, Y. Gao, H. Yu and D. Jiang, "A geohash-based index for spatial data management in distributed memory", in "2014 22Nd international conference on geoinformatics", pp. 1–4 (IEEE, 2014).

Meduri, V. V., L. Popa, P. Sen and M. Sarwat, "A comprehensive benchmark framework for active learning methods in entity matching", in "Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data", pp. 1133–1147 (2020).

Monge, A. and C. Elkan, "An efficient domain-independent algorithm for detecting approximately duplicate database records", (1997).

Monge, A. E., C. Elkan *et al.*, "The field matching problem: Algorithms and applications.", in "Kdd", vol. 2, pp. 267–270 (1996).

Morana, A., T. Morel, B. Berjawi and F. Duchateau, "Geobench: a geospatial integration tool for building a spatial entity matching benchmark", in "Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems", pp. 533–536 (2014).

Moussalli, R., M. Srivatsa and S. Asaad, "Fast and flexible conversion of geohash codes to and from latitude/longitude coordinates", in "2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines", pp. 179–186 (IEEE, 2015).

Mudgal, S., H. Li, T. Rekatsinas, A. Doan, Y. Park, G. Krishnan, R. Deep, E. Arcaute and V. Raghavendra, "Deep learning for entity matching: A design space exploration", in "Proceedings of the 2018 International Conference on Management of Data", pp. 19–34 (2018).

OSM-Data, "Osm data", URL `https://www.openstreetmap.org/` (2021).

Papadakis, G., J. Svirsky, A. Gal and T. Palpanas, "Comparative analysis of approximate blocking techniques for entity resolution", Proceedings of the VLDB Endowment **9**, 9, 684–695 (2016).

Parambath, S. P., N. Usunier and Y. Grandvalet, "Optimizing f-measures by cost-sensitive classification", in "Advances in Neural Information Processing Systems 27", (2014).

PyPi, "Python pygeohash", URL `https://pypi.org/project/pygeohash/` (2016).

PyPi, "Python client for google maps services", URL `https://pypi.org/project/googlemaps/` (2021).

Roy, A. and E. Pebesma, "A machine learning approach to demographic prediction using geohashes", in "Proceedings of the 2nd International Workshop on Social Sensing", pp. 15–20 (2017).

Ruiz-Lendínez, J. J., M. A. Ureña-Cámara and F. J. Ariza-López, "A polygon and point-based approach to matching geospatial features", ISPRS International Journal of Geo-Information **6**, 12, 399 (2017).

Sarwat, M. and TheApacheFoundation, "Geospark: Manage big geospatial data in apache spark", URL `https://www.youtube.com/watch?v=SW7ok75YghE` (2020).

Sehgal, V., L. Getoor and P. D. Viechnicki, "Entity resolution in geospatial data integration", in "Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems", pp. 83–90 (2006).

Tejada, S., "Restaurant dataset", URL `https://www.cs.utexas.edu/users/ml/riddle/data.html` (2003).

Tejada, S., C. A. Knoblock and S. Minton, "Learning object identification rules for information integration", Information Systems **26**, 8, 607–633 (2001).

Tejada, S., C. A. Knoblock and S. Minton, "Learning domain-independent string transformation weights for high accuracy object identification", in "Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining", pp. 350–359 (2002).

TheApacheFoundation, "Simmetrics java library", URL https://github.com/Simmetrics/simmetrics (2004).

TheApacheFoundation, "Apache systemds", URL https://systemds.apache.org/ (2015).

TheApacheFoundation, "Apache sedona", URL https://sedona.apache.org/ (2020a).

TheApacheFoundation, "Apache sedona", URL https://sedona.apache.org/api/sql/GeoSparkSQL-Predicate/ (2020b).

Turney, P. D., "Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm", Journal of artificial intelligence research **2**, 369–409 (1994).

Verykios, V. S., A. K. Elmagarmid and E. N. Houstis, "Automating the approximate record-matching process", Information sciences **126**, 1-4, 83–98 (2000).

WaikatoUniversity, "Weka: The workbench for machine learning", URL https://www.cs.waikato.ac.nz/ml/weka/ (2020).

Walter, V. and D. Fritsch, "Matching spatial data sets: a statistical approach", International Journal of geographical information science **13**, 5, 445–473 (1999).

Wikipedia contributors, "Geohash — Wikipedia, the free encyclopedia", URL https://en.wikipedia.org/w/index.php?title=Geohash&oldid=991104594, [Online; accessed 29-December-2020] (2020).

Yu, J., *System Support for Large-scale Geospatial Data Analytics*, Ph.D. thesis, Arizona State University (2020).

Yu, J., J. Wu and M. Sarwat, "Geospark: A cluster computing framework for processing large-scale spatial data", in "Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems", pp. 1–4 (2015).

Yu, J., Z. Zhang and M. Sarwat, "Spatial data management in apache spark: The geospark perspective and beyond", GeoInformatica **23**, 1, 37–78 (2019).