SAGEL: Smart Address Geocoding Engine for Supply-Chain Logistics

Abhranil Chatterjee abhranil.chatterjee@flipkart.com
Janit Anjaria anjaria.janit@flipkart.com
Sourav Roy sourav.roy@flipkart.com
Arnab Ganguli arnab.ganguli@flipkart.com
Krishanu Seal krishanu.seal@flipkart.com
Flipkart Internet Private Limited
6/b, 7th main, 80 feet road
3rd block Koramangala
Bangalore, India, 560034

ABSTRACT

With the recent explosion of e-commerce industry in India, the problem of *address geocoding*, that is, transforming textual address descriptions to geographic reference, such as latitude, longitude coordinates, has emerged as a core problem for *supply chain management*. Some of the major areas that rely on precise and accurate address geocoding are *supply chain fulfilment*, *supply chain analytics* and *logistics*. In this paper, we present some of the challenges faced in practice while building an address geocoding engine as a core capability at *Flipkart*. We discuss the unique challenges of building a geocoding engine for a rapidly developing country like India, such as, fuzzy region boundaries, dynamic topography and lack of convention in spellings of toponyms, to name a few. We motivate the need for building a reliable and precise address geocoding system from a business perspective and argue why some of the commercially available solutions do not suffice for our requirements. SAGEL has evolved through 3 cycles of solution prototypes and pilot experiments. We describe the learnings from each of these phases and how we incorporated them to get to the first production-ready version. We describe how we store and index map data on a *SolrCloud* cluster of *Apache Solr*, an open-source search platform, and the core algorithm for geocoding which works post-retrieval in order to determine the best matches among a set of candidate results. We give a brief description of the system architecture and provide accuracy results of our geocoding engine by measuring deviations of geocoded customer addresses across India, from verified latitude, longitude coordinates of those addresses, for a sizeable address set. We also measure and report our system's ability to geocode up to different region levels, like city, locality or building. We compare our results with those of the geocoding service provided by *Google* against a set of addresses for which we have verified latitude-longitude coordinates and show that our geocoding engine is almost as accurate as *Google*'s, while having a higher coverage.

SIGSPATIAL'16, October 31-November 03, 2016, Burlingame, CA, USA
 ISBN 978-1-4503-4589-7/16/10…$15.00
DOI: http://dx.doi.org/10.1145/2996913.2996917

CCS Concepts

•**Information systems** → **Document structure; Combination, fusion and federated search; Information extraction; Enterprise search;** *Business intelligence; Search engine indexing;*

Keywords

Geographic Information Retrieval; Spatio-Textual Searching; Storage and Indexing; Spatial Data Mining and Knowledge Discovery

1. INTRODUCTION

Geocoding is the process of transforming textual description of addresses into a geo-spatial representation, such as a latitude and a longitude. This can be as precise as finding the exact latitude longitude pair (lat-lon) for the text address of a building or retrieving the centroid and/or boundaries for a region from its textual description. For example, if the text represents *India*, the geocoding output may be the centroid of India and/or a set of lat-lon pairs representing the national boundary. In this section, we motivate the business requirement for *Flipkart* to build a geocoding engine and review some of the techniques used by the geocoding community to solve this problem across diverse application domains.

1.1 Why Flipkart needs a geocoder?

With the recent growth of e-commerce in countries like India [10, 22], the speed of order fulfilment has become a driving factor for customer satisfaction. To ensure fast fulfilment, logistics needs to be optimised on speed. However, one of the biggest challenges in optimal logistics planning is knowing the exact locations of deliveries to the customers. Particularly in the Indian context, where addressing scheme is highly non-uniform, and individuals have their own unique ways of communicating their addresses in the textual form, knowing the exact location becomes even more challenging

unless, of course an individual is highly familiar with the complete administrative level hierarchy of the concerned address.

But as an e-commerce company, we try to keep the experience of placing an order, as smooth and as fast as possible for our customers. As a result, we accept customer addresses mostly as a free text input as opposed to using a complicated interface. We cannot simply capture the user's location using the standard ways such as GPS coordinates from mobile devices or wi-fi location or cell phone tower triangulation as our customers do not necessarily place the order for themselves or from the same location where they expect the delivery to happen. Moreover, given the present tech savviness across customer demographies in India, it is highly risky to enforce a purely map based interface where the customers zoom in or navigate to the area of his or her interest and drops a marker on the map.

There are other scenarios that lead to us having to deal with incorrect addresses. For instance, there are operationally identified regions where deliveries are not done due to the area being politically disturbed or criminal activity zones. Customers from these areas sometimes end up giving wrong pin codes (zip codes) or partially complete addresses to place their orders successfully. There are other cases where a particular item is available, say in *Bangalore* and not in *Mumbai* (Availability information is displayed to the customer at the time of placing an order), the customer provides a pin code of *Bangalore* and the text refers to an address in *Mumbai*. In yet another set of cases, we have genuine customers who are confused about their pin code or their locality or sub locality, as the boundaries are subjective. These situations give rise to business problems such as high frequency of missed or delayed deliveries, because either the service executive is not able to comprehend the address written by the customer, or the actual location of the address turns out to be in an area which is outside the service area of the delivery facilities. This severely hampers customer experience and increases logistics costs. Besides, this creates auxiliary problems such as inability to scale operations as we become heavily dependent on the local area expertise of service executives.

An accurate and reliable Geocoding service can thus help our business by assigning a location context to an address text. Such location context, be it an approximate lat-lon, or a set of location objects such as locality or sub-locality that match the address in question, help the business plan out their fulfilment or delivery in a much more informed manner. Such informed decision making leads to greater customer satisfaction at lesser cost.

1.2 Related Work

The problem of Geocoding has found applications across health-care, crime analysis, logistics to name a few. In this section, we will review some of the prominent works done in the field of address geocoding.

In [5], the authors use the geocoded national address file (*G-NAF*) for geocoding and tagging addresses using a *Hidden Markov Model* (HMM) based tagger. Tagged administrative level toponyms are thus obtained which are further used to lookup in the *G-NAF* and additional data files to obtain locations. The paper describes how the preprocessing of input text and also the addresses in the *G-NAF* file are important. However, it focuses more on the preprocessing part of the geocoding engine rather than the actual algorithm of finding the best possible match. The best possible match happens with a simple look up from various files and retrieving the locations from them. We will mention later in Section 2.3 as to why our initial experiments with HMM had to be shelved in absence of a tagged training data set and a process to build one. A similar approach can be found in [4] where the authors discuss the process of geocoding as a sequence of tagging, cleaning, standardising and verification of the data. In the case of tagging the data, a greedy algorithm has been proposed. In the verification step, a direct look-up or hash encoding is proposed. In the training phase of the *HMM*, the paper mentions the use of postal address guidelines for deciding the states of the model and using the *G-NAF* for training it.

A scoring technique for geocoded result is described in [7] which is based on the level of match compared to the total municipality area. Also, geocoding of street addresses using interpolation in terms of the metric system is described. Like any other geocoding engine it relies a lot on the data availability and also on the quality of the data. Some open source geocode engines such as [13, 19, 20] could not be directly used for Flipkart because of sparsity in Open Street Maps data in India resulting in extremely low coverage (high percentage of empty results for address queries).

A patented geocoding approach [14], is applied to various web documents containing addresses and performs a look up on a table of addresses with their corresponding geographical coordinates. The look up for the location is done in the database based on the intersection in the address components. This geocoding may operate on addresses that are received by the geocoding component or extracted from the documents. Analysis of various available tools for geocoding along with criticism on the aspects of address structuring and preprocessing with respect to the *US Postal zip+4* database, can be found in [27]. In [9], the authors use a *Geocoding Certainty Indicator* (GCI) along with various geocoding techniques so as to include many variations of addresses to build a more robust system of addresses geocoding unlike the traditional GIS products. Any result crossing the GCI threshold is not considered for any statistical analysis.

A case study done in New York for geocoding using parcel or address point level data collected from the local government [3], affirmed the common intuition, that the positional error between the actual and the geocoded location is more in sparsely populated areas compared to densely populated areas. However, the research in this paper deals mostly with addresses having a standard addressing practice, such as presence of street name, street number and zip code in the address. [26] uses neighbourhood data along with a geocode database to obtain the location of a particular queried address. A data structure is designed to store the neighbourhood information along with other unstructured geographic information which can help in building a better geocoding set. A dynamic nearby reference feature scoring approach for scoring or ranking which is primarily based on spatially varying neighbourhoods can be found in [15].

[24] shows how a relational database with spatial search capability can be used to geocode addresses. Based on various rules and matching of address components, a relational database is queried using spatial library routines to obtain a final location. The geographic data is obtained from various sources and a set of rules regarding the tightness of

Table 1: Admin Level Acronyms

Country	CY
State	ST
District	DST
Subdistrict	SDT
City	CT
Locality	L
Sublocality	SL
Sub Sublocality	SSL
Building	BLD
Road	RD
Junction	JN

the match are applied. [17, 16] discusses various current techniques applied in measuring the accuracy and nature of results from a geocoding point of view and proposes a solid technique to better represent and evaluate the outputs from a geocoding engine.

2. ITERATIONS AND EVOLUTIONS

In this section, we will talk about the way we evolved our approach towards address geocoding and the practical learnings we obtained on the ground that forced us to find alternate solutions.

The first problem that we were trying to solve was to be able to automatically group shipments into small geographical regions that were operationally identified, using a text classification approach, as described in [1]. However, the pilot runs of this approach soon presented a new set of challenges such as re-training of classifier models whenever a new delivery facility was on-boarded or an existing one was discontinued. Moreover, the training data had lots of inconsistencies owing to human subjectivity at the absence of nationally recognised standards for region boundaries except state and national boundaries, as mentioned in Section 1.1. As a result it was impossible to get a bound on the classification error. The learnings from these experiments made us re-think the problem from a geo-spatial angle.

In the following subsections we describe the high level structure of the Map data on which we have built our geocoding engine and discuss the initial iterations in our approach.

2.1 The Map Data

The map data (for shapefile data format, see [11]) that we receive from our partner map data provider has some basic structure among its components (also referred to as *administrative levels* or *admin levels*) as shown in Figure 1. For ease of reference, we will follow the acronyms in Table 1 while referring to the various admin levels. It is important to note here that the process of creating, editing and revising the map data is entirely carried out by the data provider and not *Flipkart* and discussions on the same are beyond the scope of this paper. However, it is worth stressing that the structure of the map data follows a generic hierarchical structure of toponyms with textual information such as names, aliases and spatial information such as points, lines or polygons, a trend common across map data available from other providers.

We can see from Figure 1, that the highest admin level for us is the Country, followed by the state, beyond which the structure is slightly different between rural and urban

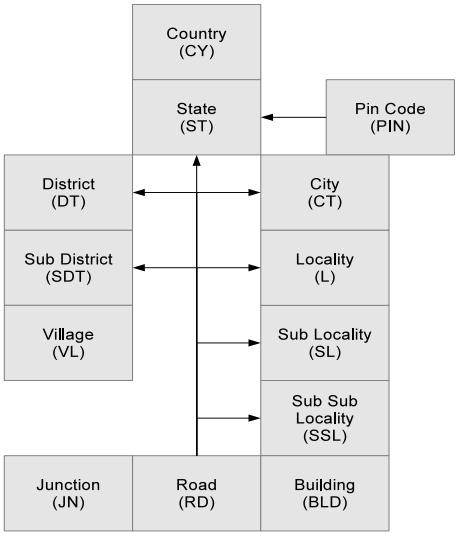

Figure 1: Relationships among map data components

regions. Rural regions are characterised by districts within states, sub districts within districts, and villages within sub districts. Whereas, Urban regions are characterised by cities within states, localities within cities and so on. Roads are common to both rural and urban regions. A road segment run from junction to junction and and have different localities, cities, districts, states etc. on its either side.

Part of the data comes with clearly defined parental relations, for instance, PIN is strictly a child of ST. However, PIN can also indicate a set of CTs and a set of Ls, SLs and so on which we derive using boolean spatial predicates like $CONTAINS$, $INTERSECTS$, $WITHIN$, etc. We describe such derivations in further details in Section 3.

2.2 A Quick Proof of Concept

Having understood the need for a base map data, we had to quickly build a prototype and give a proof of concept on a sample data for a small set of regions. In this phase, we used the inherent hierarchy described in Section 2.1, to build fat address documents. For instance, we created a document for the building *ABC Apartment* with alias *ABC Residency*, located in the locality named *Koramangala* with alias *KMG*, in the city of *Bangalore* with aliases *BLR* and *Bengaluru*, in the state of *Karnataka* with alias *KA*, and added a fat address text field reading,

> *ABC Apartment ABC Residency Koramangala KMG Bangalore BLR Bengaluru Karnataka KA.*

. Similarly, the fat text field in the corresponding locality document would read,

> *Koramangala KMG Bangalore BLR Bengaluru Karnataka KA.*

We indexed all these documents on a setup of Apache Solr [18] setup with JTS Topology suite [8] and searched the

input address text against the fat text field described above. The idea was to leverage the standard relevancy ranking of *Solr* to retrieve the intended document. Moreover, we had the pin code (PIN) as a mandatory signal to be provided by the customer. This allowed us to search for the document corresponding to the input address within the bounds of the PIN using *Spatial Search Filters* (see [18]).To see if this approach worked, we ran a pilot experiment in *Automated Route Planning* for *Last Mile Delivery* [21], where customer addresses were geocoded and the geocoded lat-lon pairs were used as input for further steps in the delivery scheduling algorithm.

The key problem that emerged from this initial pilot was the push back from our operations team due to positional errors in the geocoding output despite the fact the 90% of the geocoded latitude - longitude pairs were found to be within operationally manageable positional error, i.e. Less than 2 to 3 Kms from the actual delivery location. The primary issue was that as per our usual business process, we communicate an expected date and time of delivery (ETD) to the customer, based on his or her geocoded location and any breach on such ETD has high risks of escalations from the customer, negative sentiments spreading across social networks and the like.

Investigating the addresses that were flagged off as wrongly geocoded by ground operations, we found that a major chunk of these addresses had erroneous signals like wrong pin codes, combinations of wrong sub locality and locality, building or sub locality information missing in the base map data, etc. Manually inspecting these addresses even deeper, we could see 2 major trends. It appeared that there were a class of users who were genuinely not sure about the admin level hierarchy in their addresses leading to omission of certain admin levels or even providing a neighbouring admin level. For example, there are 2 adjacent localities named *Indiranagar* and *New Thippasandra*. There is a sub locality in Indiranagar named *LBS Nagar*, which people generally refer to directly by the sub locality name instead of using a composite naming with sub locality name and locality name. But, when a customer address reads *LBS Nagar, New Thippasandra*, the uncertainty faced by the geocoder can be characterized in terms of the following possibilities:

1. Did the customer mean *LBS Nagar* in *Indiranagar* and got the locality wrong ?

2. Is there a *LBS Nagar* in *New Thippasandra* that is missing from the map data ?

The other class of users, whom we typically call the *gaming users*, would deliberately give wrong pin codes to get their orders placed successfully, as operational serviceability of ordered items till date, is a function of the item ordered and the pin code of the delivery location.

2.3 Address Parsing

The address parsing problem is a string segmentation problem, which can be addressed through techniques commonly used by the *Natural Language Processing* (NLP) community for tasks such as named entity recognition or POS tagging. We used an HMM model, which is essentially a Finite State Machine, where each state or hidden state (in our case, these are the admin levels) has a probability distribution over the address tokens it has seen during training and a transition probability distribution over all the states in the model along with a start probability for each state. At prediction time, when presented with a sequence of address tokens, we can run the *Viterbi Algorithm* [12] to find the maximum probability assignment of tags (labels corresponding to the hidden states) to each address token. This problem is mostly considered as solved by NLP researchers and practitioners with a proven achievable accuracy of around 97% and rightly so.

For a free text address input, it is useful to parse the input using some intelligent parser in order to make sense of the tokens provided. At times, customers take the liberty of giving additional information such as driving directions from popular landmarks or time slots of his or her availability at the address, which adds lots of noise in the input and ultimately does not help the process of geocoding. However, a good address parser for Indian addresses is not available till this day and we have made an attempt at building one. We trained an HMM with a set of tagged addresses. For instance, the address

> ABC Apartment Koramangala Bangalore Karnataka

would appear in the tagged address set as

> ABC_BLD Apartment_BLD Koramangala_L Bangalore_CT Karnataka_ST.

However, in order to get high prediction accuracy from an HMM, what is extremely important to have is a good training data set. For our initial experiments on small regions where we manually generated tagged address sets, the tagger's accuracy was high as expected. But it is not feasible to manually generate tagged address set for every region of India unless other techniques like crowd sourcing are used which from a logistics business investment stand point was difficult to justify. We tried out other techniques of generating a tagged address set such as capturing the lat-lon coordinates of the customer addresses at the time of delivery and later reverse geocoding the same to get the admin level names and match against the address tokens and tag them. However, due to inherent errors of observation in capturing lat-lon coordinates on low cost GPS enabled mobile devices, it was especially difficult to get address tokens intended for building numbers and names tagged properly. This gave rise to the problem of too many untagged tokens and after some iterations trying to fix the same, we had to shelve the experiments on the grounds of low Return on Investment and focussed on alternate methods.

2.4 Successive Containment Search

Not having built a successful parser which we hoped would solve most of our problems, we experimented with an interim solution which was a series of searches and containment within the admin regions to find the narrowest possible region. However, there were 2 major issues in this approach:

1. A series of searches means a series of input-output (I/O) operations which makes the response of the geocoding service very slow - longer the addresses, slower the service.

2. Given that the admin level contours, especially at levels L, SL, SSL, are highly subjective, it is often the

Table 2: Major fields and corresponding field types in Solr schema

Field Name	Field Type
DOCTYPE	string
NAME	string
ALT_NAMES	text_general
id	string
POINT	location
BBOX	location_rpt
POINTS	location_rpt
CY_ID	string
CY_NAME	string
ST_ID	string
ST_NAME	string
L_ID	string
L_NAME	string
SL_ID	string
SL_NAME	string
SSL_ID	string
SSL_NAME	string
TEXT	text_en_split_tight

case that the customer provided signals about a certain BLD does not agree with the SSL, SL or L information in the corresponding BLD document in our index.

In this approach we were able to geocode till BLD level for only around 23% cases across urban areas in India. The experimental results in this approach forced us to consider the I/O bottleneck of sequential search and fuzziness in the usage of admin level names in addresses.

3. SAGEL

In Section 2, we have described the problems of partially missing information or partially incorrect information in text addresses of the customers. In Section 1.1, we have also mentioned the difficulties of removing free text input options from the customer and transition to a purely map based interface. Under such circumstances, we had to design an algorithm that is tolerant to partial information and incorrect toponym signals to some extent.

In the following subsections, we describe the document structuring, storage and indexing, federated searching and fusing the results and finally a dynamic programming algorithm to rank a set of candidate results.

3.1 Document Structure and Indexing

As shown in Table 2, we create a document for each record in each *shapefile* for all the admin levels ordered from larger admin levels like ST to smaller admin levels like SSL or BLD. The order of creating documents and indexing them is important because we expect to query and retrieve documents corresponding to parent admin levels while creating the document for smaller admin levels.

For instance, as in the example used in Section 2.2, while creating the document for *ABC Apartment* of type BLD, we need to run spatial queries on *Solr* to retrieve the documents for the SSL, SL, L, CT and ST that contains it. The spatial search is done using 3 fields mentioned in Table 2, namely *POINT*, *BBOX*, and *POINTS*. *POINT* field is of the *location* type and stores the lat-lon pair for point features like BLD and the centroid or popular point for polygon features like CT. *POINTS* is of location recursive prefix tree (*location_rpt*) type and stores a Well Known Text (WKT) representation [25] of polygons for ST, CT, DT, L , etc. and polylines for RD. *BBOX*, also of *location_rpt* type, stores the bounding box represented by the lat-lon pairs of the north east and south west corners, useful for doing approximate spatially bounded searches.

3.2 Query Preprocessing, Retrieval and Post Filtering

Before invoking search queries on *Solr*, we do some basic refining of the query, namely, word tokenization using the *python nlkt* package and spell correction using the *spell correct* capability of *Solr* with a dictionary of terms present in the base map data and a threshold edit distance of 2. Using the preprocessed address as the query, we do a federated search on *Solr*, to retrieve the matching documents in each admin level.

Each of these federated queries look for specific values of DOCTYPEs and searches the TEXT field using the *Standard Query Parser* of *Solr*, allowing fuzzy matches up to edit distance of 2 . For example, if the input query is *ABC Apartment Koramangala Bangalore*, then we expect to retrieve the document for *ABC Apartment* while looking for DOCTYPE of BLD, *Koramangala* while looking for DOCTYPE of L and *Bangalore* while looking for DOCTYPE of CT. Recall that the TEXT field is populated as mentioned in Section 2.1 and hence is a fat text field consisting of the aliases of the document and its ancestors.

It needs to be mentioned here that we have overridden the *Classic Similarity* computation in Solr which computes a normalized TF-IDF score given a query and a document. For a generic search application, such techniques are proven to be useful. However, on a specific document index as in our case, we do not want the score for a document to be scaled up or down when some query terms match toponyms and have high or low TF-IDF scores respectively. Also normalization would have computed lower score for documents with more aliases in their hierarchy as that would have resulted in longer TEXT fields. We implemented a *Custom Similarity* which would return the value of 1 for normalized TF-IDF score on each token match.

3.3 Ranking results with a Candidate Graph

After the candidates are retrieved from Solr via the parallel search queries, we need to find out what the user had intended. For instance, If the original query was

D475 Saket New Delhi Delhi 110017,

the parallel queries will retrieve *Delhi* for ST, *New Delhi* for CT, *110017* for PIN,*Saket* for type L , and *D475* as BLD for residential addresses. Note that the *Solr* query for type VL can also retrieve some village somewhere in the country named *Saket* if there is one. The query for type BLD would almost surely have retrieved multiple BLD documents matching the number *D475*. We only ensure that if there is a *D475* in *Saket, New Delhi, Delhi*, it will be ranked higher than the ones in other parts of *New Delhi* followed by the ones in other parts of the state *Delhi*, followed by the others in other parts of the country.

Then we have to find out the most coherent path from

ST down to the narrowest possible candidate, which in this case is BLD, and return geo information (like lat-lon) for the same. The above example was indeed a *happy case* where not much of toponym ambiguity was present and the re-ranking would be easy. However if the query says,

Vaishnavi Summit 560034

there is not much information in there. We have to use the signals available to implicitly figure out that *560034* means the ST value is *Karnataka*, CT value is *Bangalore*, L is *Koramangala*, *1st block*, *3rd block* as possible values of SL. There might be a *Vaishnavi Summit* elsewhere in the country as well but we have to figure out the path of maximum coherence that leads us to

Vaishnavi Summit building in *Koramangala 3rd block, Bangalore, Karnataka.*

To achieve this, we use a *Dynamic Programming* [2, 6], formulation and characterise the solution in terms of the solutions to the sub-problems as follows:

$$S(d) = s(d) + max(S(c(d))) \quad (1)$$

where $S(d)$ is the net score for the document d and $s(d)$ is the *Solr* score for document d (in case it was retrieved explicitly from *Solr*, else 0 if it was implicitly inferred from ancestral information in other documents). $c(d)$ is a child of the document d in terms of admin level hierarchy.

We implement this approach using a graph, which we named the *Candidate Graph*, with the node for the country *India* as the start or root node. Note that, we interchangeably use the terms, node and document. A node represents an admin level component like *New Delhi* and has a document linked to it. A node is of type $EXPLICIT$ if it was explicitly retrieved from *Solr*. Otherwise, if it was derived from the ancestral information in other retrieved documents, the node is of type $IMPLICIT$. While adding the node to this graph, we propagate the information and keep computing $S(d)$ for every document in the ancestral path from current document up to the *root* node as described in Algorithms 1,2 and 3. This makes our task easier for finally preparing the ranked list of documents. We just need to do a traversal (Algorithm 4) starting from the *root* node, choosing the unvisited child with the maximum net score (sum of the node's self *score* and its maximum child score, MCS) and backtracking once we hit a leaf node. We stop traversing the graph once the maximum number of results required is achieved or all the nodes are visited.

Considering an example query of *Vaishnavi Summit Koramangala*, Figure 2 illustrates the ranking using a candidate graph post retrieval from *Solr*. For each of the results from each parallel query for different admin levels, we create a node of type *EXPLICIT* (marked with *E*), and add their parents as nodes of type *IMPLICIT* (marked with *I*) as described in Algorithm 3. Only the nodes corresponding to *Vaishnavi Summit* in *Bangalore*, *Vaishnavi Layout* and *Vaishnavi Summit* in *New Delhi* are of type *EXPLICIT*. Once the graph is built, we traverse the graph starting from the source node (admin level CY), using Algorithm 4. The top result from the candidate graph traversal results in the darkened path from the source to the leaf in Figure 2, representing the address *Vaishnavi Summit, 3rd Block Koramangala, Bangalore, Karnataka, India.*

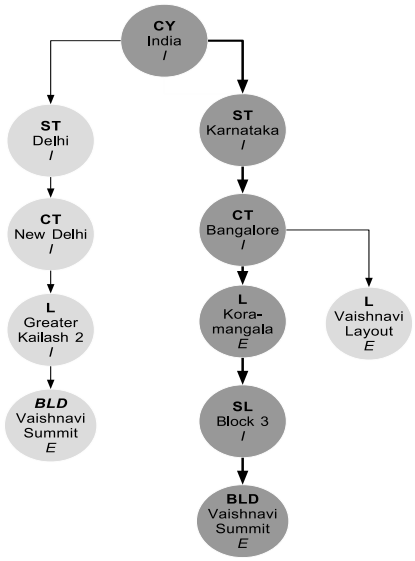

Figure 2: Candidate Graph Example

Algorithm 1 *ADDCHILD*

Require: Node N, Child node C
1: Add C to children of N
2: Update Maximum Child Score (MCS) of N, $N \rightarrow MCS \Leftarrow max(N \rightarrow MCS, C \rightarrow score + C \rightarrow MCS)$

Algorithm 2 *ADDNODE*

Require: Document $D, TypeT$
1: **if** D is already seen **then**
2: Get node N for D
3: **else**
4: Create new node N for D
5: **end if**
6: **if** T is $EXPLICIT$ **then**
7: $N \rightarrow type \Leftarrow EXPLICIT$
8: $N \rightarrow score \Leftarrow D \rightarrow score$
9: **end if**

Algorithm 3 *BUILD*

Require: Document D
1: $C \Leftarrow D$
2: $cnode \Leftarrow ADDNODE(C, EXPLICIT)$
3: **while** *cnode* has parent **do**
4: $P \Leftarrow$ Parent of C
5: $pnode \Leftarrow ADDNODE(P, IMPLICIT)$
6: $ADDCHILD(cnode, pnode)$
7: $cnode \Leftarrow pnode$
8: **end while**

Algorithm 4 $VISIT$

Require: Candidate Graph node, N, Stack S, Results R
1: Push N on top of S
2: **if** N is **not** a leaf node **then**
3: Sort child nodes of N by ($score+MCS$) in descending order
4: **for each** child, C in sorted children of N **do**
5: Return $S, R \Leftarrow VISIT(C, S, R)$
6: **end for**
7: **end if**
8: **if** N is an explicit node **then**
9: Copy stack elements, $SC \Leftarrow S$
10: Append SC to R
11: **end if**
12: Pop N from top of S
13: **return** S, R

3.4 Results

We ran experiments to measure the accuracy and the admin level coverage of *SAGEL* and compared them with that of the geocoding API of Google against a test set of real customer addresses across India for which we had verified lat-lon coordinates.

To generate the test set, we started with all shipment deliveries made by Flipkart in the last 6 months across India for which we had delivery location coordinates recorded with high accuracy. The captured lat-lon has an accuracy of observation in metres associated with it which indicates the 1-sigma radius of the approximately Gaussian lat-lon observation with the captured lat-lon being the mean of the distribution. We put a maximum threshold of 100 metres on this accuracy and any observations with accuracy of more than 100 metres were ignored to generate a set of nearly 4 Lakh (0.4 Million) addresses. We aggregated all measurements from repeated deliveries made against each customer address and further considered only those addresses for which we had at least 3 accurate observations with a maximum allowed standard deviation of 0.0005. After this we were left with only 1.5 Lakh (0.15 Million) addresses which formed our test set.

Also note that we have grouped subjective admin levels in to equivalent classes such as L, SL, and SSL as one equivalence class. This was important for fair comparison between the 2 geocoding systems as one particular toponym might represent SL for SAGEL and L for Google or vice versa. However at clearly defined admin levels such as ST, such problems do not arise and hence we do not force ST in to an equivalence class.

We have obtained a higher coverage at different admin levels from SAGEL as opposed to Google which gave empty responses for around 18% cases. As shown in Figures 3 and 4, SAGEL was able to geocode up to BLD level for 36.63% cases while Google was able to do so for 3.5% cases. For geocoding up to L, SL, SSL levels as well, SAGEL covers 36.63% cases beside Google's 1.6%.

In terms of accuracy, we measure the positional error when the system is able to geocode up to BLD level and in cases of up to higher admin level geocoding, we count the number of cases where the admin level was accurately found, by comparing with the reverse geocoded information of the captured lat-lon. For BLD level geocoding, as shown in Figure 5, SAGEL shows a 95 percentile positional error of close to 2 kilometres. It means that out of all the addresses that SAGEL was able to geocode up to BLD level, 95% were geocoded to within 2 kilometres of their recorded lat-lon coordinates. Meanwhile, Google has a 95 percentile positional error of more than 5 kilometres. Table 3 lists the comparison between Google and SAGEL in terms of precision and recall percentages for geocoding at different region admin levels. Google was able to geocode at least upto L, SL, SSL group for only 17.21% while being correct 86.6% of the time while SAGEL was able to geocode 89.08% addresses at least upto the same group with an accuracy of 77.11%. For the CT, SDT, VL group, Google was able to geocode at least upto the group for 24.65% with accuracy of 92.83% whereas SAGEL was able to geocode 99.97% addresses at least upto this group with an accuracy of 94.07%. The third row shows that Google was able to geocode 81.56% addresses at least upto ST level while being correct 75.15% times while SAGEL was able to geocode all addresses at least upto ST level while being correct 98.05% times.

From a performance perspective, it is important to ensure that the index fits in the main memory of the machines having the *Solr* index so as to minimise the number of disk reads while executing the search queries. Our production machines hosting *Solr* have 50 GB memory while the all-India index is close to 30 GB in size and *Solr* heap space usage is bound by 8 GB. In case the sum of index size and maximum heap space usage becomes larger than the available main memory size, it is recommended to optimise the index and choose an appropriate *shard*-ing strategy [18] so as to ensure that the index fits in the memory. Also, it is important to note that fuzzy search is a costly operation and depending on how critical it is to have very low latency response from *Solr*, a choice regarding use of fuzzy search should be made. There are alternate strategies that can be adopted, such as use of phonetic matching to get approximate matches. Our experiments show that fuzzy search queries that take around 500 ms can be reduced to around 100 ms by using phonetic matching. However, phonetic matching can result in false matches as well and finding the right combination of tokenizers, analyzers and phonetic filter while creating the phonetic field-type, is essential to get the latency benefit while not compromising the search result quality.

4. MINING LANDMARKS FROM DELIVERY DATA

All the discussions until this point, has been related to searching and validating on the base map data. However, given that we have delivery executives going to customer door steps and delivering, we have a potential of learning new regions and landmarks from the delivery data. We have started looking at building our own data layer by mining information from addresses where we have delivered shipments and have accurately captured lat-lon coordinates.

As a first step, we have tried out very basic methods of extracting landmark points from the free text address using regular expression based parsing of n-grams preceded by common prepositions such as, *before*, *after*, *in front of*, *behind*, etc. As mentioned in Section 3.4, delivery lat-lon coordinates are captured during delivery to customer addresses. We reverse geocode these delivery points and do a

Table 3: Precision and Recall in Geocoding at higher admin levels

Admin Level Group	Google precision	Google recall	SAGEL precision	SAGEL recall
L, SL, SSL	**86.6**	17.21	77.11	**89.08**
CT,SDT,VL	92.83	24.65	**94.07**	**99.97**
ST	75.15	81.56	**98.05**	**100.0**

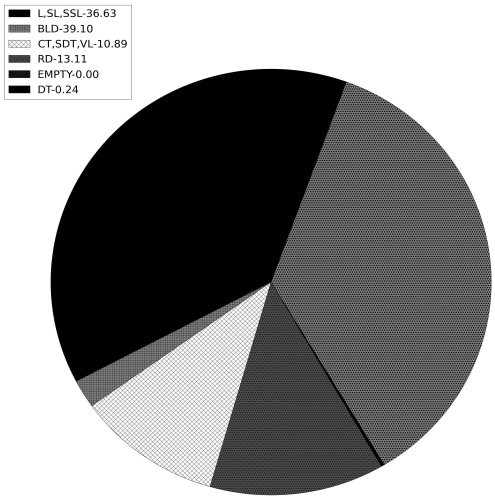

Figure 3: Admin level coverage of SAGEL

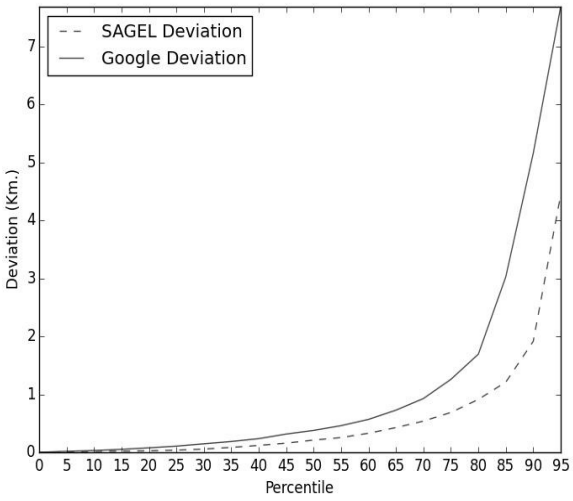

Figure 5: Comparison of Positional Error for point level geocoding

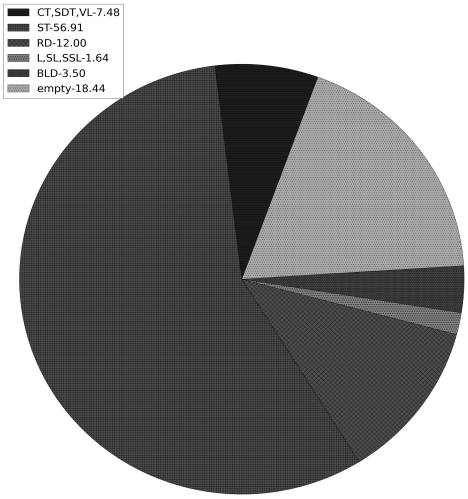

Figure 4: Admin level coverage of Google

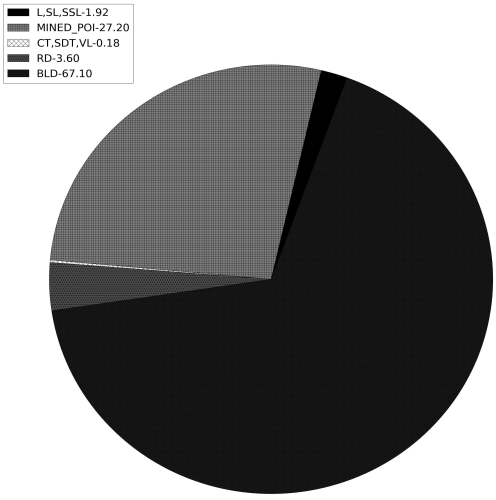

Figure 6: Admin level coverage of SAGEL including mined *PoI*

first phase segmentation by grouping them based on corresponding reverse geocoded admin level hierarchies. In each of these segments, *MinHash Shingles* [23] is used to cluster multiple n-grams with little variance in their text, which essentially means the same landmark point. These clusters become BLD level candidates below the L, SL or SSL admin levels. If enough observations are found inside these clusters then the cluster centroid is used as the location for the landmark or Point of Interest (PoI) mined from the delivery data.

We have run a small experiment on the deliveries in the city of *Bangalore* and the results (Figure 6) show that we can increase our geocoding coverage up to BLD level, by mining PoIs.

5. CONCLUSIONS AND FUTURE WORK

We have described our journey of building a Geocoding System starting from base map data for a country like India, where we face unique challenges like dynamic landscape, lack of standardisation in mid and low level administrative region contours and lack of standardised naming, aliasing and address writing practices. Despite these obstacles towards building an accurate geocoding engine with high recall, we have iterated through a series of experiments and prototypes and built a system that as a logistics unit, we can use to geocode customer addresses and take logistical decisions like which delivery facility to send a shipment to, which first mile route to choose to pickup a shipment from the seller, which last mile route to choose to deliver a shipment to the customer or where to set up the next delivery facility.

There are plenty of scopes to enhance SAGEL. The address parser work needs to be revisited with alternative models, such as conditional random fields or recurrent neural networks, which have the potential of achieving higher prediction accuracy. On the other hand, it is hard to accurately model the ground reality with just the surveyed map data with frequently changing region boundaries, new regions and points of interest coming into existence and new aliases being used for existing toponyms. As a first step towards learning new toponyms, we have started mining landmark points of interest from our delivery data as outlined in Section 4. But further work is being done in order to build a concrete learning pipeline that consumes delivery data and identifies toponyms which will help us have a stronger data layer in regions where our deliveries are more dense.

6. ACKNOWLEDGMENTS

We would like to thank Shivram Khandeparker for his advices in design and architecture of the system and Piyush Srivastava for his contributions in the brainstorming sessions, coordinating with the stakeholders, planning the development iterations and ensuring smooth execution of multiple pilot experiments.

7. REFERENCES

[1] T. R. Babu, A. Chatterjee, S. Khandeparker, A. V. Subhash, and S. Gupta. Geographical address classification without using geolocation coordinates. In *Proceedings of the 9th Workshop on Geographic Information Retrieval*, page 8. ACM, 2015.

[2] R. Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences*, 42(10):767–769, 1956.

[3] M. R. Cayo and T. O. Talbot. Positional error in automated geocoding of residential addresses. *International journal of health geographics*, 2(1):1, 2003.

[4] P. Christen and D. Belacic. Automated probabilistic address standardisation and verification. In *Australasian Data Mining Conference (AusDM?05)*, pages 53–67, 2005.

[5] P. Christen, T. Churches, A. Willmore, et al. A probabilistic geocoding system based on a national address file. In *Proceedings of the 3rd Australasian Data Mining Conference, Cairns*. Citeseer, 2004.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*, volume 6. MIT press Cambridge, 2001.

[7] C. A. Davis, F. T. Fonseca, and K. A. Borges. A flexible addressing system for approximate geocoding. In *GeoInfo*, 2003.

[8] M. Davis. Jts topology suite, 2006.

[9] C. A. Davis Jr and F. T. Fonseca. Assessing the certainty of locations produced by an address geocoding system. *Geoinformatica*, 11(1):103–129, 2007.

[10] Flipkart.com. https://www.flipkart.com. Accessed: 2016-06-21.

[11] U. ESRI and W. PaperdJuly. Esri shapefile technical description. *Comput. Stat*, 16:370–371, 1998.

[12] G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[13] Gisgraphy. http://www.gisgraphy.com. Accessed: 2016-09-21.

[14] X. Ge. Address geocoding, Aug. 23 2005. US Patent 6,934,634.

[15] D. W. Goldberg. Improving geocoding match rates with spatially-varying block metrics. *Transactions in GIS*, 15(6):829–850, 2011.

[16] D. W. Goldberg, M. Ballard, J. H. Boyd, N. Mullan, C. Garfield, D. Rosman, A. M. Ferrante, and J. B. Semmens. An evaluation framework for comparing geocoding systems. *International journal of health geographics*, 12(1):1, 2013.

[17] D. W. Goldberg, J. P. Wilson, and M. G. Cockburn. Toward quantitative geocode accuracy metrics. In *Ninth International Symposium on Spatial Accuracy Assessment in Natural Resources and Environmental Sciences*, pages 329–32, 2010.

[18] T. Grainger and T. Potter. *Solr in action*. Manning Publications Co., 2014.

[19] Photon. https://github.com/komoot/photon. Accessed: 2016-09-21.

[20] Nominatim. https://github.com/twain47/Nominatim. Accessed: 2016-09-21.

[21] A. Rajan, S. Roy, A. Chatterjee, V. V. Gargay, V. Sharma, and S. Khandeparker. PARCEL, a planning and adaptive route computation engine for logistics in India. In *ICAPS System Demonstration*, page To Appear. AAAI, June 2016.

[22] A. Sharma. Dot-coms begin to blossom in india. *Wall*

Street Journal, 12, 2011.

[23] N. Spasojevic and G. Poncin. Large scale page-based book similarity clustering. In *2011 International Conference on Document Analysis and Recognition*, pages 119–125. IEEE, 2011.

[24] P. Wang, J. Sharma, and L. Qian. Geocoding using a relational database, May 20 2008. US Patent 7,376,636.

[25] E. Westra. *Python geospatial development*. Packt Publishing Ltd, 2010.

[26] I. H. White and R. Fazal. Geocoding based on neighborhoods and other uniquely defined informal spaces or geographical regions, Nov. 16 2007. US Patent App. 11/941,698.

[27] D.-H. Yang, L. M. Bilaver, O. Hayes, and R. Goerge. Improving geocoding practices: evaluation of geocoding tools. *Journal of medical systems*, 28(4):361–370, 2004.