UrbanFlow: Large-scale Framework to Integrate Social Media and Authoritative Landuse Maps

Kiumars Soltani^{1,2}
soltani2@illinois.edu

Aiman Soliman^{2,3}
asoliman@illinois.edu

Anand Padmanabhan^{2,3,4}
apadmana@illinois.edu

Shaowen Wang^{1,2,3,4}
shaowen@illinois.edu

¹Illinois Informatics Institute

²CyberGIS Center for Advanced Digital and Spatial Studies

³National Center for Supercomputing Applications (NCSA)

⁴Department of Geography and Geographic Information Science
University of Illinois at Urbana-Champaign, Champaign, IL, USA

ABSTRACT

Everyday massive amounts of geo-tagged information are generated around urban environment using micro-blogging services and content sharing platforms. These new Big Geospatial Data sources provide an opportunity to understand people activities and their interaction with the urban environment. In this regard, it is crucial to integrate geo-tagged micro-data with more authoritative sources such as official landuse maps. This integration would benefit the urban research community by combining real time information about people activities and their spatial interaction with the synoptic view of physical infrastructure as depicted in official landuse maps. However, the scientific effort for integrating heterogeneous data sources is hindered by the lack of scalable Geospatial synthesis capabilities to accommodate the massive volume and fast update of microdata.

We developed UrbanFlow, a platform to integrate geo-located Twitter data with detailed landuse map (parcel level) to detect and analyze individual human mobility patterns. The platform provides scientists with a set of tools to extract key locations of each Twitter user, assess the extraction quality and uncertainty, and analyze city neighbors' connectivity based on detected users' frequent visitation patterns. These capabilities are built on a novel scalable solution for the point in/nearest polygon algorithm, implemented on Hadoop to harness the power of distributed systems to combine massive point data and large number of polygon in scale-out fashion. Our results showed that we are able to effectively process large data stream of Twitter data (2.42 billion tweets) and synthesize that with highly detailed landuse map (468,641 parcels for the city of Chicago).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

XSEDE16, July 17-21, 2016, ,

© 2016 ACM. ISBN 978-1-4503-4755-6/16/07...\$15.00

DOI: <http://dx.doi.org/10.1145/2949550.2949578>

CCS Concepts

•Information systems → Geographic information systems; •Human-centered computing → Social media; •Theory of computation → MapReduce algorithms;

Keywords

Data Synthesis, Point in Polygon, Hadoop, CyberGIS, Interactive Visualization, Social Media, Urban Mobility

1 Introduction

Users of micro-blogging services and content sharing platforms are generating massive amount of geo-tagged information on a daily basis. Although these big data streams are not intended as a source of Geospatial information, researchers have found that ambient geographic information (AGI) complements authoritative sources [8]. In this regard, the digital footprints of users provides a real time information about people activities and their spatial interaction, while more traditional sources such as remote sensing and land use maps provide a synoptic view of the physical infrastructure of the urban environment.

Traditionally trained scientists in social science and geography usually face great challenges when experimenting with new methods to synthesize big data sources because of the data-intensive nature of the problem. For instance, Twitter Streaming API provides close to 965 millions unique geo-tagged tweets for the year of 2014 in the United States boundaries. Similarly, authoritative datasets may include hundreds of thousands of data points for large cities. In order to overcome these challenges we developed UrbanFlow, a platform that allows scientists to synthesize massive geo-tagged Twitter data with detailed land use maps. This platform would allow scientists to gather observations to better understand human mobility patterns in relation to urban land use [14], study cities' spatial networks based on identifying common frequent visitors between different urban neighborhoods and monitoring the patterns of urban land use change. A key aspect of UrbanFlow is utilizing the power of data-intensive computing (using Apache Hadoop [13] and cloud-based services) to process massive number of tweets and integrate them with authoritative datasets, as well as efficiently store them in a database cluster in or-

der to facilitate fast interaction with users. In the rest of this paper, we explain the architecture of UrbanFlow and highlight main components of the system, including a novel distributed point in polygon algorithm. In addition, we explore the visual analytical capability of the framework that enables scientists to acquire quick and meaningful insights into the synthesized dataset. Finally, we evaluate UrbanFlow by conducting an experiment on more than 3 years of Twitter data in the city of Chicago. Our evaluation mechanism focuses for computational performance and interactivity measures two interrelated, yet independent characteristics of UrbanFlow. While computational performance focuses on providing a fast and scalable processing pipeline, interactivity promises a low-latency querying service to ensure the usability of the system for the users.

2 Background

Recent research work suggested that geo-tagged tweets are complementary sources of information to characterize the urban landuse types [6]. Information extracted from geo-tagged tweets such as the relative change in number of tweets, number of user and user movements were found to correlate with the urban activity patterns [17]. These promising results support developing integrating platforms for massive geo-located datasets. The demand for these platforms is higher because of the need to monitor and manage rapidly evolving urban landscapes, which are characterized by an increasing trend of urban landuse mixing that unbound to the conventional definitions of landuse types. For example, a traditionally designated residential landuse type could be utilized as a work and education space thanks to growth of cyberspace. Under these conditions, it is impractical to rely solely on on-site surveys with significant latency and integrating social media data would be a promising candidate to fill this gap.

Integrating different Geospatial data streams requires matching objects across datasets at the geometric and semantic levels [11]. The major limitation in this process is the lack of spatiotemporal structure of Twitter data, in contrast to the well-defined topology of a parcel-level official map. Authors have used a range of methods to identify structure in geo-tagged tweets using the statistical properties of the data or other external sources (e.g., administrative boundaries). A more objective way is to interpret the spatiotemporal structure of Twitter data in light of individual human mobility patterns [2, 15]. Human mobility patterns at the individual level were shown to be identical for most of people [7] and composed of few frequently visited locations that explain most of the geo-trace [16]. This approach shows also a promise in the particular case of Twitter as identified key user locations tends to have a dominant landuse/activity type that align with the updated landuse maps [14].

From a GIScience point of view, integrating geo-located Twitter data and authoritative landuse maps could be abstracted as a point in/nearest polygon problem. In this regard, the geo-tagged tweets could be approximated as points while detailed landuse maps are provided usually in the form of polygons (e.g., parcels). Although the classical point in polygon algorithm works fairly well for small datasets, integrating a continuous stream of tweets with detailed parcel polygons landuse maps (usually in the range of 100K polygons) could be very challenging. Several solutions for a parallel point in polygon algorithm were provided in the

Figure 1: UrbanFlow's Architecture

literature [1] [18]. However, their underlying assumption is that the number of polygons is not large-enough to contribute into the intensity of the problem. As we illustrate later, this assumption fails when we deal with highly detailed city maps such as parcel-level landuse data. In our approach, by performing a pre-processing step, we can divide-and-conquer the problem with each parallel worker accessing only a subset of both point and polygon datasets. This approach makes the result point in polygon operation scalable to both point and polygon datasets of any size.

3 Architecture

We collect input data using the Twitter Streaming API [12]. In the first step, two filters are applied to tweets: 1) Spatiotemporal filter: to filter tweets based on spatial extent and timespan of the study and 2) User filter: to filter tweets from users with very low engagement, indicating a very sparse presence over time. The second filter is particularly critical, since we want to capture actual mobility patterns of people and eliminate inactive users with noisy behaviors.

In the next step, the processed tweets dataset is integrated with authoritative dataset using our novel distributed point in polygon algorithm (refer to Section 4.1). Then we identify frequently visited locations of each user (Section 4.2). Finally the clusters are joined with the user-defined partitioning scheme to enable region-based analytics. All the above steps are implemented using Apache Hadoop.

The result of Hadoop-based processing pipeline is written into a MongoDB¹ cluster. Due to the large size of data, we shard the dataset to break data into multiple partitions, each hosted on a separate database instance. The database is connected to the interactive web application via a NodeJS server². Since NodeJS employs a highly event-driven architecture by taking advantage of asynchronous communication model, our web application can support fast simultaneous access from multiple users. Figure 1 illustrates the UrbanFlow's architecture.

4 UrbanFlow Components

In this section, we articulate the main three components of UrbanFlow. First, the novel distributed point in polygon

¹A NoSQL document-oriented database (<https://www.mongodb.com/>)

²An environment to develop server-side web applications, built on Chrome's V8 JavaScript engine (<https://nodejs.org/>)

approach is explained in detail. Next, we define how Urban-Flow recognize frequently visited locations of Twitter users based on spatial clustering and evaluate its uncertainty. Finally we illustrate the visual diagnostics of identified users' frequent visited locations, which are accessible through a web application.

4.1 Point in Polygon

The core algorithmic contribution of this work is the distributed point in polygon (PIP) algorithm, implemented using Apache Hadoop. Previous work on this problem [18, 1] suffered from the inability to scale when the number of polygons are large. This limitation is due to the fact that these methods attempt to send all the polygons to every worker, which is clearly inefficient for large number of polygons (e.g. the highly detailed Chicago landuse data used in Section 5). In other word, these approaches only apply the parallelism to the points dataset, ignoring the potential intensity of polygons dataset.

To address the above issue, our approach pre-processes the polygons, and breaks them into smaller spatially contiguous chunks. We store the bounding box information of each chunk into a small file that can be easily distributed among mappers. Therefore, in the map stage, we can quickly identify which chunk each point is associated with. Based on that information, the points are sent to the location-aware reducers. Now each reducer only requires loading a small chunk of the original polygons set, which significantly reduce both reading and indexing time.

Our solution expands the original point in polygon operation to take account for possible inaccuracy in the recorded location of users and the measured boundaries of polygons, by finding the *closest* point in polygon. The goal here is to find the polygon which is closest to a given point, considering a distance threshold ($minDist$). Let's consider point p which lies into partition i with bounding box BB_i . If p matches a polygon which is presented in partition i , our work is done (no approximation required). However, in case p does not match any polygon in partition i , we should look for the polygon that has the minimum distance to p . The main issue is that in partitions' boundaries, this polygon may not intersect with BB_i . We resolve this issue by expanding the polygons in partition i to includes any polygon that is located in distance $minDist$ of bounding box BB_i . While we expand the polygons in partition p beyond BB_i , we will not change the representative bound of partition i . This guarantees that a point will match with one and only one partition. Otherwise an expensive post-processing step is required to handle points that are associated with multiple partitions. Figure 2 illustrates this process.

The Algorithm 1 explains the process of splitting the original polygons into smaller sets. The input for the algorithms consists of polygons, number of partitions and the approximation threshold. The output is the set of partitions and the bounds file which are stored in HDFS. Number of partitions can be determined based on Hadoop cluster's available resources and size of the data. The MapReduce process of associate point dataset with polygons is illustrated through Algorithms 2 and 3. The mappers read and index the bounds file (once per each mapper) to determine which reducer each point should be sent to. The reducers take care of the main point in polygon process. Each reducer reads and indexes the its associated partition only once and then

Figure 2: Example of splitting space into 4 partitions, illustrating how each partition includes more polygons than it's representative bounding box (colored rectangles) to enable *closest* point in polygon operation. The non-filled rectangles show the bounding box of all the polygons in each partition.

use that to find the closest polygon associated with each point. The spatial indexing (for bounds and partitions) are done using R-tree [9].

4.2 Identifying Frequently Visited Locations

Previous studies have indicated that human mobility patterns can be explained by preferential return to few frequently visited locations (FLV) such as home, work, school, etc [14]. There are multiple methods to identify the FLVs, ranging from simple solutions such as imposing a fixed grid on the users' locations [4] to more advanced methods such as spatial clustering techniques [3]. We identify FLVs by using DBSCAN clustering algorithm [5] on each user's recorded locations. DBSCAN method does not assume a predefined number of clusters or any particular spatial shape. This is particularly beneficial for our application, due to heterogeneous nature of human mobility. The result of this step is a series of clusters per user, marked by their spatiotemporal characteristics as well as their rank in the user's everyday activities. The rank of a cluster demonstrates the tendency of the user to return to that FLV and determined based on the number of tweets in the cluster. For instance, for most of the users the first rank cluster is considered as home, while work/school may appear in the lower ranks [14].

We also introduce the purity metric of clusters, to evaluate the choice of clustering parameters. The purity of a cluster is defined as the frequency of the most common lan-

Algorithm 1: Split original polygons into partitions

```
function split(partition, threshold)
  divideDirection ←
    alternate(partition.divideDirection) ;
  newBounds ←
    divideBound(partition, divideDirection) ;
  newPartitions ← new Partition[2] ;
  foreach polygon in partition.polygons do
    if intersect(newBounds[0], polygon, threshold)
    then
      newPartitions[0].addPolygon(polygon) ;
    if intersect(newBounds[1], polygon, threshold)
    then
      newPartitions[1].addPolygon(polygon) ;

  newPartitions[0].divideDirection ←
    divideDirection ;
  newPartitions[1].divideDirection ←
    divideDirection ;
  return newPartitions ;

function generate(polygons, maxPartitions,
  maxReducers, distanceThreshold)
  partition ← new Partition(polygons) ;
  partitions ← new Array() ;
  partitions.add(partition) ;
  while partitions.length ≤ maxPartitions do
    splitPartition ←
      getMostDensePartition(partitions) ;
    partitions.add(
      split(splitPartition, distanceThreshold)) ;
    partitions.remove(splitPartition) ;

  reducersOrder ←
    generateRandomPermutation(1, maxReducers) ;
  bounds ← new Array() ;
  for i = 1...maxPartitions do
    partitions[i].reducerId ←
      reducersOrder[i%maxReducers] ;
    writeToHDFS(partitions[i]) ;
    bounds.add(getBounds(partitions[i])) ;
  writeToHDFS(bounds) ;
```

duse parcel to total number of points in that cluster. For instance, the purity of 1.0 indicates that the cluster is completely located in/near one land use parcel. For a reasonable choice of clustering parameters, we expect the most clusters to have high purity number.

4.3 Visual Analytics

UrbanFlow provides three categories of visual analytics for the users:

1. *Clustering analytics* are used to evaluate the clustering algorithm and obtain general insights on users mobility. They include a) time series plots of tweets volume for hour of the day and day of the week grouped by landuse or cluster ranks (Figure 3(a)), which would allow the user to evaluate how the temporal changes related to the landuse function; b) stacked plots of the semantic composition (landuse labels) of dominant landuse type for all unique users grouped by rank (Figure 3(b)) that summarize the user bias to tweet at different lan-

Algorithm 2: Mapper for point in polygon

```
tree ← null ;
function setup(jobConfig)
  bounds ←
    ReadFromHDFS(jobConfig.boundsFile) ;
  tree ← BuildQuadTree(bounds) ;

function mapper(point)
  partition ← tree.find(point.coordinates) ;
  emit(partition.reducerId, point) ;
```

Algorithm 3: Reducer for point in polygon

```
tree ← null ;
distanceThreshold ← 0 ;
function setup(jobConfig)
  id ← jobConfig.getReducerId() ;
  polygons ←
    ReadFromHDFS(jobConfig.partitionFiles[id]) ;
  tree ← BuildQuadTree(polygons) ;
  distanceThreshold ←
    jobConfig.distanceThreshold ;

function reducer(key, Point[] points)
  foreach point in points do
    matchedPolygons ←
      tree.findByRadius(point, distanceThreshold) ;
    f(p) ← {distance(p, point) | p ∈
      matchedPolygons} ;
    closestPolygon ← argminp f(p) ;
    write(closestPolygon.id, point) ;
```

duse type and c) box plots of clusters semantic purity grouped by rank or landuse (Figure 3(c)), to evaluate the choice of parameters in the clustering algorithm.

2. *Single region-based analytics* are presented when user select a region based on the defined aggregation scheme. In addition to the similar plots as clustering analytics, a pie chart of each cluster dominant language is rendered based on the language field of Twitter API.
3. *Joint region-based analytics* enable users to obtain insights on how regions are related to each other as a part of an urban spatial network. The nodes in this network are individual regions and edges represent the number of users who have frequently visited locations in both nodes. By selecting any region (e.g. census tract), a heatmap will be generated to illustrates the location of the other regions connected with the selected region located (Figure 4(a)). The user can subset the urban network by restricting the landuse type at the source and destination nodes to answer questions such as where the people who work in this area live (Figure 4(b)) or go to university (Figure 4(c)). This ability allows the users to get a more focused insights on the type of human mobility they intend to study. In addition, by selecting two regions, a chord plot will be generated to highlight the type of spatial activity interaction between these two polygons.

All these visual analytics are presented through a web application in CyberGIS Gateway [10] - an online environ-

(a) Time series plot for number of tweets per hour of the day and day of the week for each landuse type

(b) Stack plot of dominant land use for top visited locations grouped by rank, confirming the high preferential bias to tweet from residential specially for top locations (rank1)

(c) box plot of clustering purity with most of the clusters correspond to a single land use parcel (purity = 1.0)

Figure 3: Visual analytics in UrbanFlow's web application

(a) No filter on the type of the users' clusters.

(b) Residential clusters of the users

(c) University related clusters of the users (less diversity due to the focused nature of this landuse type)

Figure 4: Heatmap of users spread who have work-related cluster in the selected region (marked in blue) with multiple filtering criteria.

ment for a large number of users to perform computing and data-intensive, and collaborative geospatial problem solving. This allows for seamless integration of the application with multiple computing environments (refer to the experiment section) and other CyberGIS applications.

5 Experiments

We conduct an experiment using geo-tagged tweets from January 2013 to January 2016 (2.42 billions tweets for US boundaries), and filter them to only keep tweets that match Chicago’s bounding box (nearly 47 million tweets). To identify the landuse type of each tweet’s location, we use a highly detailed landuse map of Chicago³ that includes 468,641 parcels. The approximation threshold to consider closest polygon is set to 100m. Furthermore, using $\epsilon = 0.00225$ degree and $minPts = 3$ for DBSCAN, 925,819 frequently visited locations are extracted from 718,652 users. Finally, the census tract dataset (2025 tracts) is used to provide users with region-based analytics.

We deploy UrbanFlow on NSF-funded ROGER supercomputer⁴ (equipped with Intel Xeon E5-2660 processors). The hybrid nature of ROGER allows us to access Hadoop cluster, MongoDB cluster and NodeJS servers in the same environment. ROGER’s Hadoop cluster includes 11 nodes, each having 20 cores and 256GB of memory. Based on the system’s setting, it can run the maximum of 121 mappers or 55 reducers at the same time (the memory requirement for mappers and reducers are different in ROGER). In addition, MongoDB database is hosted on 4 OpenStack⁵ managed virtual machines, each configured with 4 cores of the Intel Xeon E5-2660 processors.

To demonstrate the capability of UrbanFlow for fast and effective integration of large-scale data, three sets of experiments are presented: a) point in polygon performance, b) overall computation time for the framework and c) server query latency. All the metrics in these experiments are calculated using average of 3 runs. The block size for the HDFS, which determines the number of mappers is set to 64MB, with replication factor of 2 and the number of reducers is set to 64, unless explicitly noted.

The first experiment compares our point in polygon approach with the existing approach of sharing all the polygons with every worker (from now on we refer to this approach as *all-to-all*). One important note is that based on the Hadoop cluster configurations, running the all-to-all approach may not be possible. This is due to the fact that in a standard Hadoop cluster the available memory for one mapper/reducer is quite limited (usually less than 1GB). The limitation is closely related to the nature of Hadoop clusters which are intended as commodity clusters. Therefore, in most of the Hadoop clusters running all-to-all approach for a slightly large number of polygons is impossible. However, since ROGER is designed to provide real-time analytical capabilities for geospatial problems, it is equipped with high memory nodes. This allows us to implement the all-to-all approach and compare the performance with our distributed algorithm.

We compare the two point in polygon approaches using two metrics: a) *spatial indexing time* which measures the

average time it takes each reducer to build the spatial index based on its associated set of polygons and b) the average query time for the reducers to find the closest polygon. The time for pre-processing step is excluded from this experiment, since it has to only get called once per polygon datasets and as we explain later accounts for a very small part of the computation process. We set the number of partitions equal to the number of reducers to get one partition per reducer, hence eliminating the effect of scheduling and shuffling stages in MapReduce and focus on the core algorithm. As Figure 5 illustrates, using our point in polygon approach we can significantly reduce the spatial indexing time, since less data is required to be read and indexed. However this reduction is not going to be proportional to the number of partitions. The reason lies into the overlapping nature of our partitions which requires storing one polygon in multiple partitions. This time for the all-to-all approach remains nearly constant since it has to index all the polygons regardless of the number of reducers. In addition, we observe that the query time decreases using our point in polygon approach (Figure 6). This decrease is not going to be as substantial as the spatial indexing part since the querying complexity is from $O(\log n)$ where n is the number of polygons. Therefore for a large n the speed up ratio will be very small.

Figure 5: Time to build the spatial index for point in polygon.

To evaluate the overall computing time of the framework, we have to determine the optimal number of partitions and reducers. The decision to choose either numbers comes with trade-offs. First, while having more reducers usually lead to higher parallelism (if the available resources allow), it will also add considerable overhead to the Hadoop framework. In addition, choosing large number of partitions reduces the risk of highly dense regions and load imbalance. However, by adding more partitions we also increase the number of duplicate polygons in the boundaries. Table 1 shows the total computing time of point in polygon operation for different number of partitions and reducers. Since each reducer should have at least one partition to process the upper part of the table is undefined. We observe that for a fix number of partitions, we usually get better performance as we move to larger number of reducers. Based on Table 1 having 256 partitions with 128 reducers provides us with the best computing time. Here by increasing the number of partitions,

³Landuse Inventory for Northeastern Illinois - <http://www.cmap.illinois.gov/data/land-use/inventory>

⁴Resourcing Open Geo-spatial Education and Research

⁵Open-source software for creating and managing clouds

Figure 6: Average query time for reducers in point in polygon

we reduce the time required to build the spatial index, as well as reducing the risk of having skewed and dense regions. However, we observe that by moving to 256 reducers the overhead of reducers overcome the extra parallelism and the overall computing time increases.

Partitions/Reducers	8	16	32	64	128	256
8	317	-	-	-	-	-
16	303	158	-	-	-	-
32	331	236	136	-	-	-
64	290	192	131	117	-	-
128	282	173	139	121	118	-
256	279	183	145	114	106	123

Table 1: The point in polygon computing time for different combinations of number of partitions and reducers (in seconds)

The overall computation time for more than 3 years of Twitter data is 1295 seconds with tweet extraction and filtering step taking over 50% of the computation time (Figure 7). This is due to the fact that loading all the raw tweets (338 GB) and process them to apply the filters is quite time-consuming. However, this step, as well as the pre-processing and finding landuse steps, are usually only called once in the course of an study, while the researchers may try multiple clustering parameters or aggregation schemes. The second time-consuming part of the pipeline, which accounts for almost 20% of the overall time is the clustering process. This step is particularly time-consuming since some the long-tailed nature of social media leads to a group of overly active users with many locations. Therefore the reducers which handle these users usually have a considerably higher running time.

To ensure the interactivity of UrbanFlow, we measured the query latency for the two most intense query patterns in our framework:

- *getCluster*: Query a region to get all of the clusters which are located in that region.
- *getSpread*: Query a region to find the spread of users from that region.

Figure 7: UrbanFlow's running time for each component (time in seconds and percentage)

The pattern *getSpread* is particularly intense, since it requires two levels of queries to get the users of that region and find out other frequently visited locations of those users. For both patterns, we run queries on all the regions (2025) and measured mean, median and 90% percentiles of query latencies. Table 2 shows the result for query latencies in pattern 1 and 2. Using 90% percentile of query latencies we can conclude that the interaction of the users is most likely take less than half a second, even for the most intense queries.

Query Pattern/Latency	Mean	Median	90% Percentile
<i>getCluster</i>	152	103	260
<i>getSpread</i>	247	154	466

Table 2: Query latency in UrbanFlow for defined patterns in milliseconds

Our experiments confirm that UrbanFlow can efficiently handle processing large datasets while providing a fast interactive environment for the users to analyze the result.

6 Conclusion and Future Work

UrbanFlow provides scientists with an interactive environment to synthesize social media fine-resolution data with authoritative dataset, to gain deeper insight on mobility patterns through complex urban area. Using distributed algorithms (e.g. point in polygon) which are implemented on top of Apache Hadoop, the framework is able to scale out to large datasets. This provides a unique opportunity for scientists to move from working with small sample of data and expand their studies to include more spatial and temporal dimensions.

The UrbanFlow platform employs Geospatial algorithms optimized for distributed computing environments, to enable fast processing and low-latency query performance. Coupled with a set of visual analytics, UrbanFlow allow users to evaluate the performance and uncertainty of Twitter data synthesis with other sources and adjust their experimental setting. In addition, the flexibility of defining inputs and aggregation polygon maps would allow users to explore and test new method for Big Data fusion not limited to authoritative landuse maps.

We are actively working with the potential academic communities, such as geographers, social scientist and urban

planners to expand the range of core Geospatial capabilities of our system. Future improvements include utilities to infer key location landuse types based on tweets cluster attributes (e.g., temporal behavior), which would allow extending our framework to areas with no existing landuse map. In addition, we are exploring network analysis methods that can be incorporated into the application, to identify hotspots and tightly coupled regions.

ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation (NSF) under grant numbers: 1047916, 1354329 and 1443080. The work used the ROGER supercomputer, which is supported by NSF under grant number: 1429699. The authors would also like to thank the members of the CyberInfrastructure and Geospatial Information Laboratory (CIGI, <http://cigi.illinois.edu/>) for their insightful comments and discussions.

7 References

- [1] Esri gis-tools-for-hadoop. <https://github.com/Esri/gis-tools-for-hadoop>, 2013.
- [2] J. I. Blanford, Z. Huang, A. Savelyev, and A. M. MacEachren. Geo-located tweets. enhancing mobility maps and capturing cross-border movement. *PLoS ONE*, 10(6):1–16, 06 2015.
- [3] N. Bora, Y.-H. Chang, and R. Maheswaran. *Social Computing, Behavioral-Cultural Modeling and Prediction: 7th International Conference, SBP 2014, Washington, DC, USA, April 1-4, 2014. Proceedings*, chapter Mobility Patterns and User Dynamics in Racially Segregated Geographies of US Cities, pages 11–18. Springer International Publishing, Cham, 2014.
- [4] G. Cao, S. Wang, M. Hwang, A. Padmanabhan, Z. Zhang, and K. Soltani. A scalable framework for spatiotemporal analysis of location-based social media data. *Computers, Environment and Urban Systems*, 51:70 – 82, 2015.
- [5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.
- [6] V. Frias-Martinez and E. Frias-Martinez. Spectral clustering for sensing urban land use using twitter activity. *Engineering Applications of Artificial Intelligence*, 35:237 – 245, 2014.
- [7] M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, June 2008.
- [8] M. F. Goodchild. The quality of big (geo) data. *Dialogues in Human Geography*, 3(3):280–284, 2013.
- [9] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’84, pages 47–57, New York, NY, USA, 1984. ACM.
- [10] Y. Liu, A. Padmanabhan, and S. Wang. Cybergis gateway for enabling data-rich geospatial research and education. In *Cluster Computing (CLUSTER), 2013 IEEE International Conference on*, pages 1–3, Sept 2013.
- [11] J. Martinez. Semantic Integration of Authoritative and Volunteered Geographic Information using Ontologies. 4 2013.
- [12] A. Padmanabhan, S. Wang, G. Cao, M. Hwang, Z. Zhang, Y. Gao, K. Soltani, and Y. Liu. Flumapper: A cybergis application for interactive analysis of massive location-based social media. *Concurrency and Computation: Practice and Experience*, 26(13):2253–2265, 2014. CPE-13-0348.R2.
- [13] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The Hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST ’10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.
- [14] A. Soliman, J. Yin, K. Soltani, A. Padmanabhan, and S. Wang. Where chicagoans tweet the most: Semantic analysis of preferential return locations of twitter users. In *Proceedings of the First ACM SIGSPATIAL International Workshop on Smart Cities and Urban Analytics*, UrbanGIS ’15, 2015.
- [15] K. Soltani, A. Padmanabhan, and S. Wang. Movepattern: Interactive framework to provide scalable visualization of movement patterns. In *Proceedings of the 8th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS ’15, 2015.
- [16] C. Song, Z. Qu, N. Blumm, and A.-L. Barabási. Limits of predictability in human mobility. *Science*, 327(5968):1018–1021, 2010.
- [17] S. Wakamiya, R. Lee, and K. Sumiya. *GeoSpatial Semantics: 4th International Conference, GeoS 2011, Brest, France, May 12-13, 2011. Proceedings*, chapter Urban Area Characterization Based on Semantics of Crowd Activities in Twitter, pages 108–123. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
- [18] J. Zhang, S. You, and L. Gruenwald. Lightweight distributed execution engine for large-scale spatial join query processing. In *Big Data (BigData Congress), 2015 IEEE International Congress on*, pages 150–157, June 2015.