AFRL-RI-RS-TR-2017-107

XDATA

RAYTHEON BBN TECHNOLOGIES

MAY 2017

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the 88th ABW, Wright-Patterson AFB Public Affairs Office and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2017-107 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
NANCY ROBERTS
Work Unit Manager

/ S /
MICHAEL J. WESSING
Deputy Chief, Information Intelligence
Systems and Analysis Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
MAY 2017	FINAL TECHNICAL REPORT	SEP 2012 – DEC 2013

4. TITLE AND SUBTITLE

XDATA

5a. CONTRACT NUMBER
FA8750-12-C-0239

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62702E

6. AUTHOR(S)

Walt Andrews

5d. PROJECT NUMBER
XDAT

5e. TASK NUMBER
A0

5f. WORK UNIT NUMBER
11

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Raytheon BBN Technologies
9861 Broken Land Parkway, Suite 400
Columbia, MD 21046

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RIEA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2017-107

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. PA# 88ABW-2014-4715
Date Cleared: Oct 8, 2014

13. SUPPLEMENTARY NOTES

14. ABSTRACT

For the XDATA project we researched and implemented software concerning graph analytics & natural language processing (NLP). Throughout these efforts we attempted to focus on the problems that arose when implementing the algorithms in parallel. We also partnered with subcontractors SentiMetrix & Johns Hopkins University for further graph analytics. On the graph side, we researched & provided an implementation of a graph matching algorithm, as well as a "process finder" which identifies patterns in graphs with a temporal element. On the NLP side, we did work on word embeddings, & provided parallel implementations of a wordcloud algorithm as well as a package that combined topic clustering with a Chow-Liu tree to estimate joint probability distributions between the topics revealed in the clustering. We worked with the Twitter, BitCoin, Akamai traceroute, and Akamai CIDR datasets.

15. SUBJECT TERMS

Cloud computing, scalable analytics, deep learning, semantic analysis

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**NANCY ROBERTS**
U	U	U	UU	41	19b. TELEPHONE NUMBER (*Include area code*) **(315) 330-3566**

TABLE OF CONTENTS

LIST OF FIGURES

EXECUTIVE SUMMARY

For the XDATA project we researched and implemented software concerning graph analytics and natural language processing(NLP). Throughout these efforts we attempted to focus on the problems that arose when implementing the algorithms in parallel. We also partnered with subcontractors SentiMetrix and Johns Hopkins University for further graph analytics.

On the graph side, we researched and provided an implementation of a graph matching algorithm, as well as a "process finder" which identifies patterns in graphs with a temporal element.

On the NLP side, we did work on word embeddings, and provided parallel implementations of a wordcloud algorithm as well as a package that combined topic clustering with a Chow-Liu tree to estimate joint probability distributions between the topics revealed in the clustering.

We worked with the Twitter, BitCoin, Akamai traceroute, and Akamai CIDR datasets.

1. Summary

1.1 Patterns in Near-real Time (PINT) Algorithm

For the XDATA project we identified that a proprietary implementation of our Patterns in Near-real Time (PINT) algorithm would be useful for working with the XDATA datasets. We modified the algorithm to be used with these datasets and ran analyses on the Twitter and BitCoin data, searching the Twitter dataset for instances of political protests and the BitCoin data for fraudulent or suspect transactions. At the time of the stop order, we were working to parallelize the algorithm and to use it to analyze the GISR data.

1.2 Two-Step Canonical-Correlation Analysis Algorithm

For the XDATA project we implemented the two-step canonical-correlation analysis algorithm for finding word embeddings. We then used this algorithm on Twitter hashtags in the XDATA Twitter dataset and used it in conjunction with clustering in an attempt to identify protest tweets.

1.3 Graph Matching/Parallel Assignment

For the XDATA project we implemented a parallel version of the graph matching algorithms developed by Cary Priebe and Joshua Vogelstien at Johns Hopkins

University[1]. We were interested in using these algorithms for inference involving the Akamai traceroutes dataset, but found that the parallel implementation proved to be insufficiently efficient for problems of moderate size. As such, we focused on devising and adopting algorithms for the parallel solution of the linear sum assignment problem, which is a component of this and many other graph matching algorithms. As part of this effort we developed a potential improvement to a general auction algorithm and parallelization schemes to improve auction algorithms in a distributed memory setting. At the time of the stop order we were still evaluating these approaches, with the eventual goal of creating a toolkit providing the user with several possible algorithms depending on the nature of her data.

1.4 Network Inference of Link Strength (NILS)

SentiMetrix proposed to develop methods to automatically learn the intensity of network links between entities on the basis of context (for example: location, population, network connections, etc...) and/or content (for example: natural language text data). In order to achieve this, SentiMetrix would extract a variety of factors related to network strength and aggregate them in order to get a concrete score of the relationship between two entities. The result will be a labeled, weighted graph where the weight denotes the strength of the link between two entities and the label denotes the evidence supporting the existence of a link. This would allow one to take advantage of multiple types of information to generate weighted graphs. SentiMetrix further proposed to develop a dashboard where a DoD analyst can interact directly with the proposed NILS system so that the analyst can bring his domain knowledge also to bear on the problem.

1.5 Flickr Anomaly Capture Engine

We developed an algorithm detecting anomalous behavior in the Flickr data set. The goal was to develop the Flickr Anomaly Capture Engine (FACE) to automatically calculate a FAST (Flickr Anomaly Test Score).

1.6 Anomaly Detection in Time Series Graphs

We extended our methodology for statistical inference on time series of graphs to specifically address issues related to Raytheon BBN & XDATA -- in particular, directed graphs, weighted graphs, attributed graphs, and large graphs. Our paper "Locality statistics for anomaly detection in time series of graphs," recently accepted for publication at IEEE Transactions on Signal Processing, provides the foundations for these

[1] *see* JT Vogelstein, et al. *Fast Approximate Quadratic Programming for Large (Brain) Graph Matching*. Computational Statistics & Data Analysis, available at <http://arxiv.org/abs/1112.5507> and Donniell E. Fishkind, Sancar Adali, C.E. Priebe, "Seeded Graph Matching", submitted, 2012 available at <http://arxiv.org/abs/1209.0367>

extensions. Our intention was, and remains, to incorporate these extensions into the "scan statistics on time series of graphs" addition to the iGraph package.

2 Introduction

2.1 Patterns in Near-real Time (PINT) Algorithm

The Patterns In Near-real Time (PINT) algorithm is an implementation of a process finder, originally conceptualized at Johns Hopkins University Applied Physics Lab, for a proprietary US DoD customer. The initial purpose of PINT was to perform automated analysis of intelligence event observations to identify occurrences of known red force processes, where observations correspond to process steps through a many-to-many relationship. Generally, the algorithm allows intelligence analysts or Subject Matter Experts (SMEs) to search for malicious processes and/or events in large scale data.

The processes are modeled as directed acyclic graphs, specified by the user, which induce a partial order on the observable events. PINT identifies these processes using high-performance spectral clustering over the nodes of a pairwise-affinity graph. Each node in that graph represents a possible binding between an intelligence observation and a process step. The process model induces an affinity function *Aff(b1,b2)* over pairs of observation-to-process-step bindings. Each pair of bindings is thus associated with an affinity, which is treated as the weight of an edge in the graph of all possible bindings. The pairwise affinity between two nodes in this graph is used to derive clusters of strongly connected components, which correspond to process instances.

PINT, in its initial, un-optimized, single-node version, handles the matching of hundreds of thousands of observations to process models with hundreds of nodes in seconds.

2.2 Two-Step Canonical-Correlation Analysis Algorithm

Distributional Word Representations

Distributional word representations are a way of using unlabeled data to reduce data sparsity in labeled training data. In natural language processing applications, word features are typically in a very sparse, high dimensional space (the size of the vocabulary). However, one can learn a dense, low-rank representation which captures the correlation between a word and the words in its immediate context. These representations are often called “word embeddings”. They are easier to generalize, and are used as training features for tasks such as named entity recognition[2], semantic role labeling,[3] and

[2] Scott Miller, Jethran Guinness, Alex Zamanian, "Name Tagging with Word Clusters and Discriminative Training".. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.105.9395&rep=rep1&type=pdf

[3] Ronan Collobert, Jason Weston, "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning".. http://ronan.collobert.com/pub/matos/2008_nlp_icml.pdf

chunking[4]. Some different ways of getting word embeddings are Brown clustering[5], deep learning, and spectral learning[6].

Spectral Algorithms

Spectral algorithms rely on singular value decomposition (SVD) as a basic operation[7], and can be used to cluster data that is connected, but not compact or clustered within convex boundaries[8]:

- Compactness, e.g., k-means, mixture models
- Connectivity, e.g., spectral clustering

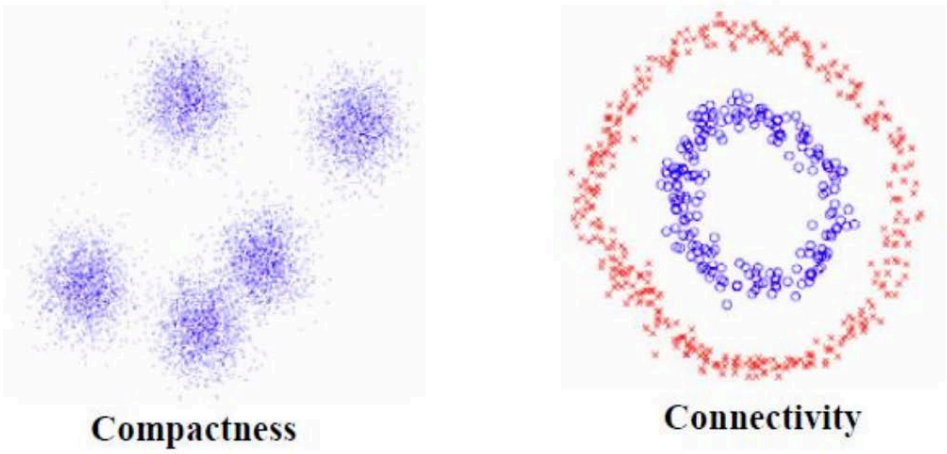


Figure 1: From http://charlesmartin14.files.wordpress.com/2012/10/spec.png

These techniques make use of the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction[9].

Canonical-Correlation Analysis

Canonical correlation analysis tries to find basis vectors for two sets of multi-dimensional variables such that the linear correlations between the projections onto these basis vectors are mutually maximized.[10] The input are data matrices X (size n by d_1) and Y (size n by d_2). The rows, n, are the observations, and the columns, d_1 and d_2, are the variables. We can think of X and Y as providing two views of the same data.

---

[4] Joseph Turian, Lev Ratinov, Yoshua Bengio, "Word representations: A simple and general method for semi-supervised learning." http://www.iro.umontreal.ca/~lisa/pointeurs/turian-wordrepresentations-acl10.pdf

[5] Peter Brown, Peter deSouza, Robert Mercer, Vincent Della Pietra, Jenifer Lai. "Class-Based n-gram Models of Natural Language". http://acl.ldc.upenn.edu/J/J92/J92-4003.pdf

[6] Paramveer Dhillon, Jordan Rodu, Dean Foster, Lyle Ungar. Two-Step CCA: A new spectral method for estimating vector models of words. http://icml.cc/2012/papers/763.pdf

[7] http://www.cs.columbia.edu/~scohen/naacl13tutorial/

[8] http://charlesmartin14.wordpress.com/2012/10/09/spectral-clustering/

[9] http://en.wikipedia.org/wiki/Spectral_clustering

[10] http://www.fon.hum.uva.nl/Proceedings/Proceedings25/cca_ifaproc2003.pdf

To find the shared basis vectors, find the SVD of $(cov(X,X)^{-1/2} * cov(X,Y) * cov(Y,Y)^{-1/2})$. This gives $U\sum V^T$, where U and V are the matrices that maximize the correlation between X*U and Y*V.

.

2.3 Graph Matching/Parallel Assignment

Graph Matching

Graph Matching algorithms seek to align the nodes in one graph to the nodes in another by finding a mapping of the nodes. More formally this can be defined as finding the permutation matrix that, when applied to the second graph, minimizes the disagreement between the two graphs. The problem is *np*-hard, and thus the current algorithms provide approximations.[11] Despite this, they do not typically run sufficiently fast for many applications using large datasets.

Existing Algorithms

Our starting point was the Fast Quadratic Approximation algorithm proposed by Vogelstein. This algorithm applies the Frank-Wolfe algorithm, using the linear sum assignment problem as the linear approximation for the quadratic graph matching problem. In the authors' testing, this algorithm outperformed competing approaches.

Another approach worth mentioning is the belief propagation algorithm proposed by Bayati et al.[12] An attractive feature of this algorithm is that it is dependent only on the sum of a linear sum assignment, as opposed to the assignment itself. Khan et al. determined that using an approximation algorithm for the linear sum assignment problem causes limited degradation to the quality of the graph matching.[13] While we haven't explored this approach fully, we would briefly note that this is consistent with our observations with approximations for the LSAP - that given a random initial graph - most approximations will find an assignment of reasonably close to optimal weight that will not necessarily contain many matching consistent with the optimal assignment.

Relatedly we also developed an auction-based approximation algorithm foe the linear assignment problem that both parallelizes well and gets somewhat more of the assignments correct than alternate approaches.

Linear Assignment Problem

As mentioned above many graph matching algorithms utilize the linear sum assignment problem. The goal of this problem is to find a one-to-one mapping of one class of nodes to another such that that edge weight is maximized or minimized. The performance issue

[11] Conte, Donatello, et al. "Thirty years of graph matching in pattern recognition." *International journal of pattern recognition and artificial intelligence* 18.03 (2004): 265-298.

[12] Bayati, Gerritsen, Gleich, Saberi and Wang, "Algorithms for Large, Sparse Network Alignment Problems," *2009 Ninth IEEE International Conference on Data Mining.*

[13] Khan, Gleich, Pothen, and Halappanavar, "A multithreaded algorithm for network alignment for approximate matching," 2012.

with our implementation of the algorithm, is that traditional approaches to the LSAP are difficult to parallelize efficiently in a distributed memory environment.

While numerous approaches to the LSAP exist, the two main classes are those based around the auction algorithm, and those based around maintaining a reduced cost matrix and expanding augmenting or alternating paths.[14] While the current best sequential implementations use these paths, these algorithms do not lend themselves to a natural implementation for distributed memory processors, as each expansion of a path may require inter-processor communication.

The auction algorithm, meanwhile, is superficially parallelizable in that it is simple to distribute the nodes onto different processors and calculate bids. The problem, however, is that the algorithm itself involves iteratively updating a global price array. As such, global communication is necessary at each iteration. The naive complexity of the algorithm is *cn^2*. At each iteration, a given processor performs *n* units of the work, leaving the possibility of *cn* iterations requiring communication, where *c* is often substantial. For a more sophisticated implementation, there is the possibility of *nlog(n)* communications. As communication is typically much more expensive than local processor operations, this is less than ideal in practice.

2.4 Network Inference of Link Strength (NILS)

SentiMetrix sought to develop NLP capabilities to:

1. Define an evidence ontology specifying different types of evidence either validating or refuting the hypotheses that two entities are related

2. Use the evidence ontology in order to develop evidence extraction algorithms that capture evidence from text explaining why two entities E and E' are connected from a mix of free text and associated metadata,

3. Develop connection strength scoring algorithms that quantify the strength of connection between E and E' on a 0 to 1 scale (0 meaning completely unconnected, 1 meaning 100% evidence of being connected). In order to do this, we will associate a profile vector PV(u) with each user which will capture both the user's context as well as content-oriented textual information about the user. We will experiment with a variety of metrics to measure distance and/or similarity between multi-dimensional vectors to determine which metrics give us the highest accuracy (of detecting whether two entities: E and E' are the same or not). The result of the algorithms will be a labeled, weighted graph whose vertices are entities found, in the data, whose edges are weighted with the predicted strength of relationship between entities E and E', and whose labels tell us what the

[14] For an in depth overview, see Burkard, Dell'Amio, and Martello, *Assignment Problems*, Society for Industrial and Applied Mathematics, 2012.

evidence for the relationship is. The evidence labels will be drawn from an ontology of evidence types (e.g. location, membership, titles, etc.).

4. Develop a dashboard in which the analyst can examine the different types of evidence labels supporting the existence of a link of strengths between two entities and weight them according to the analyst's domain knowledge.

2.5 Flickr Anomaly Capture Engine

In Flickr, pictures are favorited by individuals. A picture P on average gets a certain number of "favorite" markings during any given time frame T. Suppose P is a population of pictures. We defined the concept of a "favorite distribution" fd(P,T) *w.r.t.* P and T which plots, on the *x*-axis, the number of likes a picture has received during a given time period T, and on the *y*-axis, the percentage of pictures with those "likes." It is well known that when P = ALL (i.e. all Flickr pictures are considered), this results in a long tailed distribution because very few pictures receive a large number of likes. Unusual activity in Flickr can occur in one of at least three ways:

1. Unusual activity occurs when fd(ALL,T) differs substantially from fd(ALL,T-j) for all j less than or equal to some threshold. To measure the "difference" between two distributions fd(ALL,T) and fd(ALL,T-j), we can use one of many measures existing in the literature to identify the difference between distributions.

2. Instead of considering all pictures, we focus on a specific population P of pictures (e.g. pictures from a given country or pictures of a given genre).

3. Unusual activity can be defined by a picture-oriented temporal distribution (POT---distribution for short). Here, we focus on a specific picture I. For an integer j > 0, the (I,j) time series is a time series showing the cumulative number of likes of picture I during the past j time units. Thus, for any given time point t=Now, we have a graph G(I,j,t). The x-axis of the x-axis, time points (Now-j), (Now-j+1),...,(Now-1),Now. On the y---axis, we plot the cumulative number of likes of picture I up to and including time (Now---i) divided by the cumulative number of likes of picture I up to and including time Now. To see if a picture's likes are anomalous, we need to look at the difference in the distributions G(I,j,Now-j),...,G(I,j,Now).

3 Methods, Assumptions, and Procedures

3.1 Patterns in Near-real Time (PINT) Algorithm

Detailed Description of the Algorithm

The PINT process finder algorithm performs three steps:

Step 1	Create a graph of observation-to-process-step bindings
Step 2	Find clusters in said graph
Step 3	Perform quadratic assignment to evaluate each cluster

Step 1 - Create a graph of bindings: A binding is a possible match between an observation (e.g. a tweet) and a process element, called an activity. The graph of bindings is represented by a square adjacency matrix with the edge weights corresponding to the degree of "alignment" or "affinity" between the two bindings. The alignment is a heuristically-computed value indicating whether two observations belong to the same process when mapped to their corresponding process activities. For example, consider two activity-observation *(A-O)* bindings, B_1 and B_2:

B_1: $A_1 – O_1$
B_2: $A_2 – O_2$

The affinity (AFF) or alignment between B_1 and B_2 is computed as follows:

Condition	Action	Explanation
$A_1 == A_2$	AFF = 0	If the activity in the two bindings is the same, then there is no edge between them
$O_1 == O_2$	AFF = 0	If the observation in the two bindings is the same, there is no edge
Distance(O_1,O_2) > max_dist	AFF = 0	If the distance in space between two observations exceeds a threshold, there is no edge
TimeDifference (O_1,O_2) > max_time	AFF = 0	If the difference in time between two observations exceeds a threshold, there is no edge
PartialOrderOfActivitiesViolated(B_1,B_2)	AFF = 0	If the observations are out of order with the activities, there is no edge
ELSE	AFF = $1/d_1 * 1/d_2$	d_1 and d_2 are the number of activities that correspond to O_1 and O_2 respectively. This a measure of combined "relevancy"

Step 2 - Find clusters of bindings: This step finds all strongly-connected component clusters (iteratively and approximately) in the pairwise affinity graph constructed in step 1. Power Iteration Clustering (PIC)[15] is used for this purpose. In this approach, the leading eigenvector of the adjacency matrix is computed approximately. These values are used as "agreement" scores for each activity-observation binding.

Each binding is then evaluated for whether it belongs to one of the existing clusters or to a new cluster. A Jaccard Coefficient between a new binding and the center binding of an existing cluster is calculated and if the coefficient is greater than or equal to a threshold, the new binding is added to the cluster. A binding can be added to multiple clusters. If it is not added to any clusters, it forms the center of a new cluster. The clusters are filtered

[15] https://code.google.com/p/matrix-toolkits-java/
[16] N. Halko, P. Martinsson, and J. Tropp. "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions." http://arxiv.org/pdf/0909.4061v2.pdf

to contain only solutions of some minimum size, in terms of the number of bindings, activities, and observations.

Step 3 - Cluster Scoring: Lastly, we perform a quadratic assignment within each cluster to find the best-fit assignment of observations to activities and to score the cluster. This step ensures that the activity-observation assignments in each cluster identified in step 2 do not violate the constraints of the process model. A constraint checker ensures that the bindings in the solution comply with the process's partial-order of activities.

We use a spectral method to perform the quadratic assignment. This algorithm uses the leading eigenvector to make the initial assignment and then makes a series of heuristic "swaps" until a better solution is reached. This solution is approximate.

The cluster is then evaluated in terms of the number of process activities matched by observations vs. the total number of process activities that could be matched (in some processes, sets of activities could be mutually exclusive, e.g. when the process takes two alternative paths).

3.2 Two-Step Canonical-Correlation Analysis Algorithm

Two-Step CCA
The two-step CCA algorithm applies canonical correlation analysis twice in order to find vectors that represent words. We start with a large corpus of text (such as news articles, or Twitter text). We concatenate all of the documents together, to get one large document with *n* word tokens, and a vocabulary of size *v*.

We make three sparse binary data matrices W,L, and R, and compute the embeddings as follows:

- **W** is the words matrix, with *n* rows and *v* columns. Each row is filled with zeros, except for a one in the column to represent the word that is in that position in the text. For example, if the document is "the green apple" then the matrix **W** is:

	the	**green**	**apple**
Word 1:	1	0	0
Word 2:	0	1	0
Word 3:	0	0	1

- **L** is the left-context matrix, where each row contains a one for the word to the left of the word at that position (we add an extra column for words that are at the beginning of a sentence, and do not need to include a column for words that never appear to the left of another word). Here is an example of the **L** matrix for the same sentence:

	****null****	**the**	**green**
Word 1:	1	0	0
Word 2:	0	1	0
Word 3:	0	0	1

- **R** is similar to **L** except that instead of the word to the left, it is the word to the right.

Step 1	Do CCA between **L** and **R**: SVD(cov(**L**,**L**)$^{-1/2}$ *cov(**L**,**R**)*cov(**R**,**R**)$^{-1/2}$) = U$\sum$V^T
Step 2	Using the results from Step 1, create a new context matrix **S** which is a horizontal concatenation of **L***U and **R***V. This will contain some redundant information, but will not lose any differences in information between **L** and **R**. Then, do CCA again between **S** and **W**. The matrix V_2 (from the output of the second CCA) is the basis of the words in **W** with respect to their context **S**. The first 30-50 dimensions should make a sufficient representation of the word.

In practice, L, R, and W will be very large, for example, 600,000 columns and 100 million rows. In order to make the two-step CCA computationally feasible, a few changes are made. First, instead of finding the covariance matrices, we pretend that all the means are zero and find L^TL instead of cov(L,L), L^TR instead of cov(R,R), etc. L^TL and R^TR from step one, and W^TW from step two, will all be diagonal matrices, so their inverse square roots can be found element-wise. Finally, as Dhillon et al. explain in their paper, the FastSVD algorithm from Halko et al.[16] which uses random projections to compute the SVD of large matrices is used in place of traditional exact methods

3.3 Graph Matching/Parallel Assignment

Proposed Algorithms

M-highest Auction

We implemented several approaches to the auction algorithm. The first of these is a general approach (though it can also be implemented in parallel).

In the standard auction algorithm each bidder (or row) selects one object (or column) to bid on. Its bid for this object is the difference in value between this, the highest valued-object, and the second-highest-valued object, plus a slack variable.

[16] N. Halko, P. Martinsson, and J. Tropp. "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions." http://arxiv.org/pdf/0909.4061v2.pdf

A problem that typically arises is a "bidding war." This occurs where a number of bidders have similar high values for a group of objects. Because the values for each bids top objects are similar, the marginal differences and thus the bids are small. As a result it takes numerous iterations for the bids to finally discriminate between the objects. Take the following matrix:

2	16	17
4	20	21
1	7	8

It is fairly easy to discern that the optimal assignment involves assigning the second and third columns to the first and second rows. When running the auction, however, the initial bids for each row all have a value of $1 + e$ (the slack variable) and are placed on column three. In the second round, each bid is placed on the second column, with a value of $2e$. This will continue until the prices of the second and third columns rise enough for row three to instead select column one. Depending on the size of e, this could continue for a number of rounds.

We found that this situation proved to be particularly common in the Fast Approximate Quadratic Algorithm which we were implementing. The first step in this algorithm is to run a linear assignment on the following matrix, where P is uniform doubly stochastic matrix.

$$A * P * B' + A' * P * B$$

Both halves of the sum will produce a scaled matrix, in which each row is equivalent to every other row multiplied by a scalar. While the summed matrix is not strictly scaled, it is in practice very close to being so, and this can lead to bidding wars as explained above, in addition to being a generally time-consuming matrix to solve.

A simple solution would be to initialize P differently, however in doing so one will likely jeopardize the quality of the solution. Another approach that we developed was to have each bidder bid on m-objects at the same time. In theory, where the objects involved in the bidding war are captured in the same m-group, the bidding war can be resolved in a single round. This should be of use where many of the nodes would bid on the same handful of objects, as was the case here. The algorithm proceeds as follows:

While Unassigned Bidders	
Step 1	Get m-highest objects for each bidder. The bid for each of these objects is the value of that object - value of the $m+1^{th}$ object
Step 2	Collect all bids for the same *m* objects
Step 3	Run a greedy / approximate assignment within the object groups, and calculate the bids based on the results of this auction
Step 4	Choose the highest bid for each object. Assign the object to the bidder of this bid
Step 5	Update the price array
End	

Step 3 can be somewhat tricky in that for the auction to work, each bidder, after a successful bid, must be essentially as well off (within *e)* as it would be with any other object. In the above algorithm, however, this is only true where we can guarantee that any objects smaller than the object a bidder bids on will have its price raised to the level that bidder would have bid on it. This requires some adjustment of the bids gathered in Step 1. With these adjustments, however, the standard proof of the auction algorithm holds.

This algorithm certainly works for the toy example. How well it works for real examples depends on the structure of the matrix being assigned and on the choice of *m*. Figure 1 shows the number of modified auction (communication) rounds needed to assign a matrix derived from the following equation, the first step in our graph matching algorithm:

$$A * P * B' + A' * P * B$$

Figure 1a exemplifies the desired effect of the algorithm, as it seems for this matrix the bidding war has been encapsulated. There is a steep drop-off in the number of rounds required when *m* hits three, suggesting that large groups of bidders share the same three highest-valued objects . As shown in Figure 1b, however, this effect is less pronounced when averaged over multiple matrices. Additionally, on a uniform-random matrix, the algorithm does not provide a significant benefit.

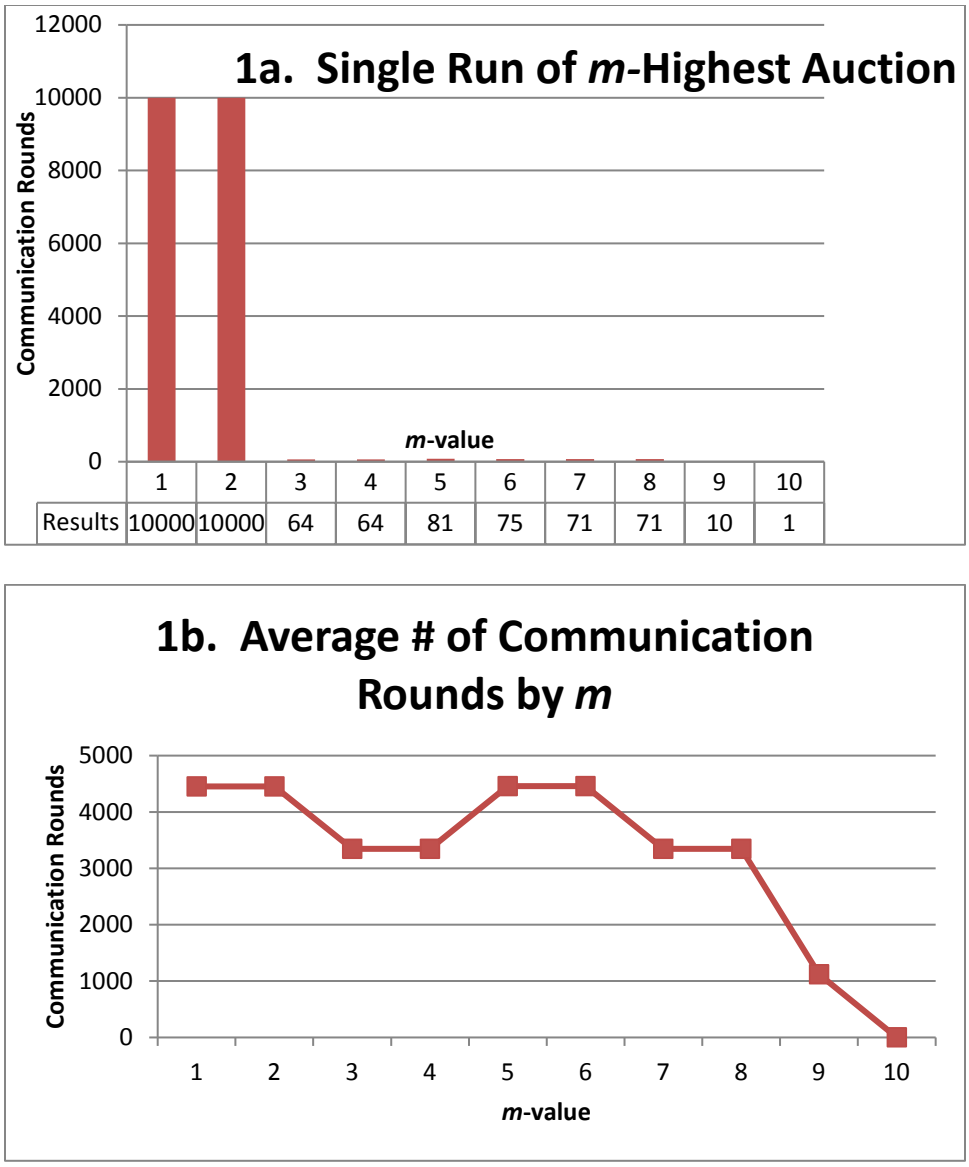


Figure 2: 1a provides an illustration of randomly generated matrix for which the algorithm works as intended. 1b shows that generally the effects are more moderate. Note that the decrease that occurs as the *m*-value reaches its maximum is somewhat illusory – as this indicates a full auction is being performed in a given step.

In short, the algorithm shows both the promise of resolving assignments quickly, but also a need to be applied adaptively. We are interested in exploring adaptive approaches more fully.

Divide and Assign Locally

Ultimately, in designing an algorithm to run in a distributed memory environment, one generally wants to place as much computation as possible on the local processors as local processing is cheap and communication is expensive. The most natural way to do this for the auction (or other forms of the assignment algorithm) is to run sequential auctions on individual processors.

Riedy proposed a simple block auction algorithm that does just this.[17] The algorithm proceeds as follows:

Step 1	Partition bidders (rows) to processors
While Unassigned Bidders	
Step 2	Run local auction where all objects are available on each processor
Step 3	Collect bids and assign objects to winners
Step 4	Update price array and send to all bidders
End	

Our aim was to develop Step 1 so that the partition is done smartly, that is to assign bidders to processors where they will be placed with "similar" bidders. We experimented with difference pre-partitioning steps and partitioning methods to achieve this goal.

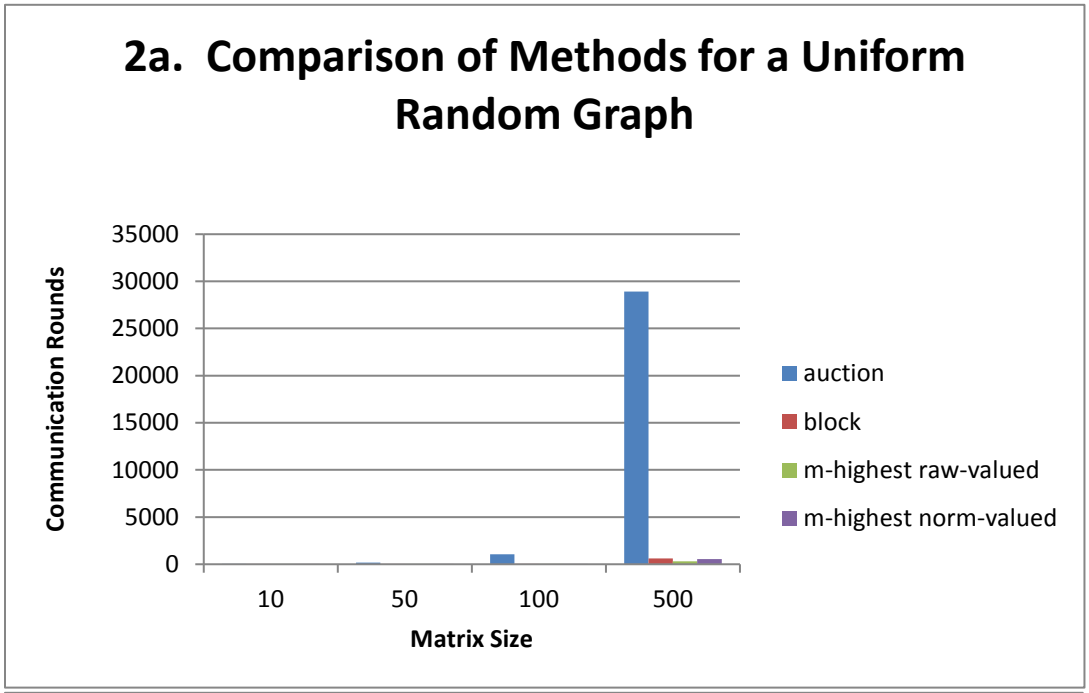
In terms of the pre-partitioning steps, we were interested in reducing the number of edges. Many graph based partitioning algorithms have time complexity that scales with the number of edges. Our initial approach involves assigning a weightless edge to the *m*-highest entries for each row and *m*-highest entries for each column.

We also experimented with "normalizing" the graph before selecting the edges. With many real graphs, certain rows or columns will have higher weights across all objects, and thus thresholding the edges will produce substantially similar rows and columns. This is counterproductive to the goal of creating meaningful partitions. Thus, it might be desirable to modify the graph such that the strength of a node's edges are considered in light of how they compare to the other edges incident to that node. This provides more information as to whether an edge will be included in the final assignment. Newman's modularity score seems to be an appropriate manner to achieve this.[18]

Figure 2 shows some basic results of the described approaches on a random matrix. We believe that the normalization step is more useful where the matrix is non-uniform, but it nonetheless shows improvement over the basic block approach.

[17] Edward Jason Riedy, *Making Static Pivoting Salable and Dependable*, Thesis for University of California at Berkeley, 2010.

[18] Newman, Mark EJ. "Modularity and community structure in networks." *Proceedings of the National Academy of Sciences* 103.23 (2006): 8577-8582.



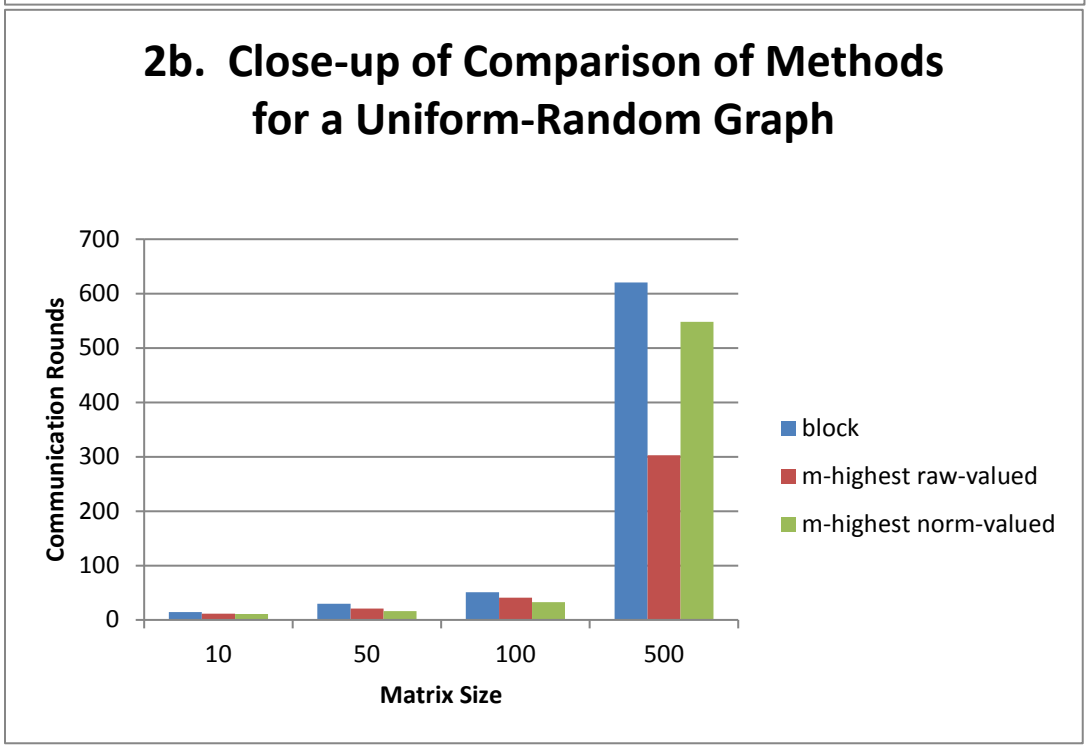


Figure 3: Shows the number of communication rounds necessary with various pre-partitioning approaches. 1a shows that all such approaches require significantly less communication than the standard auction algorithm.

For the partitioning, we looked at spectral and multi-level graph-partitioning algorithms as well as the label propagation algorithm designed for "community detection." Ideally we wished to avoid spectral clustering as it is expensive for large graphs, though this

problem might be somewhat mitigated by recent approximation algorithms. We used the METIS/ParMETIS[19] implementation of graph partitioning as something of a baseline.

We were also intrigued by the label propagation algorithm[20] which is very fast in practice and finds divisions without artificial constraints, which is to say that on a given run a graph could be partitioned into any number of groups of any size. The natural drawback with this is that the groups must then be assigned to a fixed number of processors with a given capacity. It also may find groups too large for any processor which would then require subsequent partitioning – either by the same method or another. That said, it finds high quality groups quickly, so we were interested in developing an approach to creating a partition from the labelled groups.

Co-Cluster and Assign Locally

The above approach involves partitioning the rows and sending the full price array to each processor. This is desirable in that the bids from each processor can then be combined using the standard auction algorithm, and the proof of the auction algorithm holds. A more aggressive approach, however, would be to co-cluster the graph, and limit bidders to the objects on the same processor. Where the graph is well-partitioned, this should enable the optimal assignment to be reached rather quickly, and for "bidding wars" to be encapsulated on a single processor without communication. Generally, the algorithm would follow the form:

While Unassigned Bidders	
Step 1	Normalize cost matrix for co-clustering (optionally)
Step 2	Implement a binary threshold for both rows and columns for highest value elements (optionally)
Step 3	Partition bidders and objects to processors
Step 4	Run local assignment algorithms (not limited to the auction algorithm)
Step 5	Create a price array based on the results of the local assignments
Step 6	Improve upon the result of step 5 using a distributed auction algorithm
End	

Step 5 is the crux of this algorithm. Given an assignment, one can set a price array for the auction algorithm such that the auction will immediately terminate if that assignment is correct, or otherwise continue. It is reasonably efficient to do this given the actual correct assignment. Given an incorrect assignment, however, it becomes more challenging, and was a question we were concerned with at the time of the stop order.

[19] Parmetis: Parallel Graph partitioning and sparse matrix ordering library, G Karypis, K Shloegel, Kumar, 2003.
[20] Raghavan, Usha Nandini, Reka Albert, and Sounder Kumara "Near linear time algorithm to detect community structures in large scale networks" *Physical Review E 76 3, 2007.*

With further work, we believe that such approaches can greatly limit the amount of communication necessary to implement the assignment algorithm, which will be useful for a number of problems as an increasing amount of "scientific" programming is done on commodity clusters.

3.4 Flickr Anomaly Capture Engine

The intent was to evaluate the above approaches and implement one of them using both Flickr data provided by the XDATA project and the new data collected via a SentiMetrix-developed tool. The effort was divided into three tasks:

Task 1: Develop a Flickr time series data set consisting of graphs F_1,F_2,…,F_k describing the state of the (snapshot of the) Flickr social network provided to us over k months. The plan was to use Favorites data from Flickr for data set augmentation.

Task 2: Develop the methodology to find anomalous phenomena occurring in social networks. The question is to determine what constitutes an "anomalous" phenomenon. In order to achieve this, a Flickr Anomaly Score Tracker (FAST) needed to be defined that assigns to each user U and each month M, an anomaly score specifying how anomalous the user was during that month – the more anomalous the user during a given month, the greater the importance is of his pictures. The anomaly score will be between 0 (not at all anomalous) and 1 (maximally anomalous).

Task 3: Develop a preliminary prototype implementation of FAST on the dataset listed above, and a demo GUI.

4 Results and Discussions

4.1 Patterns in Near-real Time (PINT) Algorithm

4.1.1 Application of PINT Algorithms to XDATA Datasets

We successfully applied the PINT algorithm to detect events in the Twitter and BitCoin data, and worked on applying it to the GISR data.

For the Twitter data, we extracted tweets in several languages of interest that included linguistic features (unigrams, n-grams, patterns) associated with various social and political protest activities. Firstly, we developed a protest process model, using the generalized versions of the linguistic features as process activities. We then used PINT to detect instances of protests, as indicated by the observed tweets, with constraints, such as a geospatial radius of 35km (roughly a mid-sized city), and a timespan of 24-48 hours.

We also applied PINT to BitCoin transaction data, where we detected sequences of transactions across several days that might indicate illicit activity. One such transaction pattern that we looked for was a high-value transactions followed by circular transactions involving the same source and/or destination accounts as the high-value transactions Our test process model (admittedly simple) contained transactions of the following suspicious types:

1. High Value Transactions: Transactions above some threshold level, initially set to 2,000
2. Circular Transactions: Transactions in which at least one non-trivial sub-transaction is a self-transaction
3. Obfuscated Transactions: Transactions with more than 100 sub-transactions

Our test setup included a first pass, which filtered the transactions of interest from a large BitCoin transaction data set (up to T transactions, where T ranged from 1M to 16M) and constructs a set of PINT observations matching the "suspicious" criteria. We then invoked PINT on the set of observations, which detected instances of suspicious sequences, such as:

(High Value Transaction) – followed by – (Circular Transaction)

(High Value Transaction) – followed by – (Obfuscated Transaction)

We used 4-day and 8-day time periods.

4.2 Two-Step Canonical-Correlation Analysis Algorithm

Application to Twitter Data

Understanding hashtags is important for understanding Twitter data. This section describes some ways that we used the CCA to generate hashtag embeddings. For all approaches we used a subset of 1.14GB of Tweets from the city of Dubai (where most of the tweets were in English). We also looked to use word embeddings to identify protest tweets in Spanish, though this effort ultimately shifted to identifying word clusters to aid in feature selection

Distributional hashtag embeddings

We trained the embeddings for the hashtags by looking at the words to the left and to the right of the hashtag in the body of the tweet.

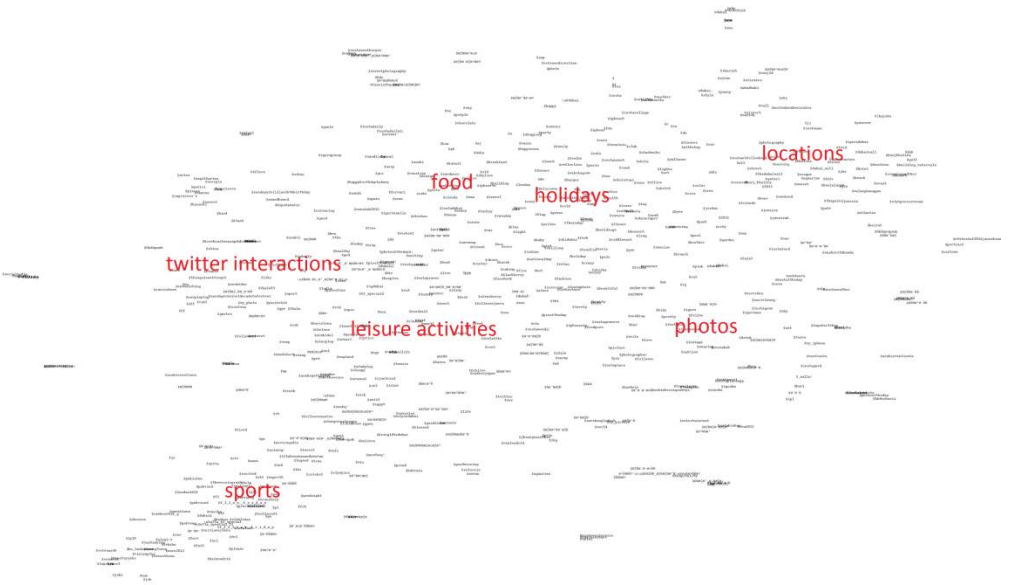


Figure 4: Distributional hashtag embeddings, with some text that I added in red to show a basic idea of what tags are in each area.

Figure 5 shows a visualization of the hashtag embeddings that were generated like this, with the text in red added to provide a basic idea of how it is laid out. However, looking at Figure 6, it is apparent that the embeddings are very noisy. When looking at the tags and their contexts, one thing that stands out is that for most of the hashtag instances, there is a null word either to the left or the right. It seems that most instances of hashtags on Twitter are used at the beginning or the end of a Tweet, so distributional embeddings are not very meaningful.

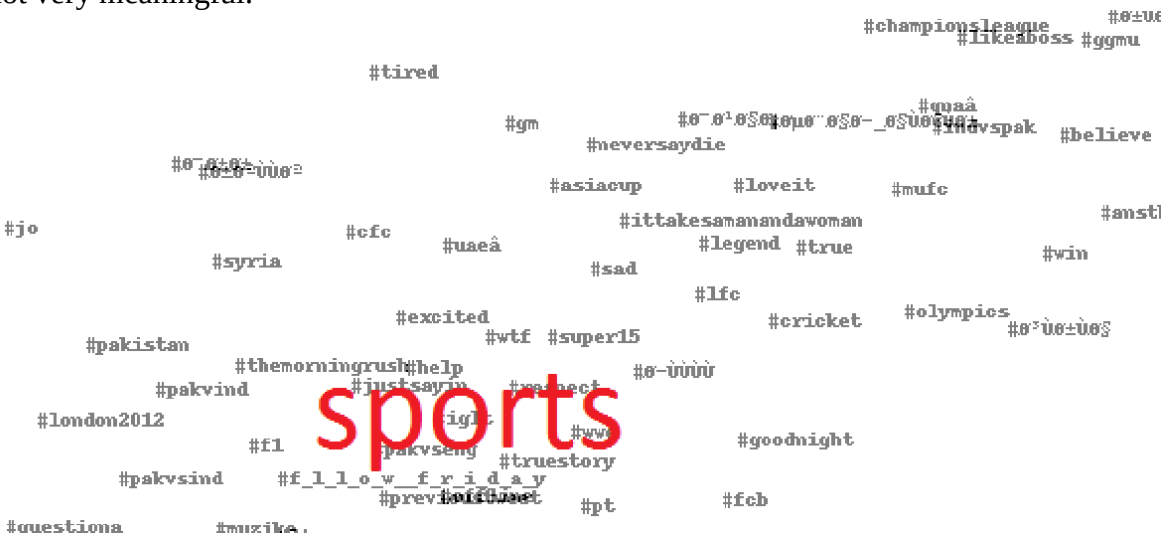


Figure 5: Close up from Figure 5. You can see that the distributional embeddings for hashtags are very noisy.

Bag-of-words hashtag embeddings

Since the distributional representations didn't seem to work very well for hashtags, the next step was to try learning vector representations for hashtags using all of the words in the tweet body. Here we used a bag of words representation for the text of the tweets. The representations are then found using a one-step CCA. The first of the two views is the W matrix of hashtags, with n rows for each instance a hashtag is used, and v_h columns for the size of the hashtag vocabulary. The second is the C matrix (for context), which also has n rows, and v_c columns which is the size of the vocabulary of words that are used in the tweets. The C matrix is a binary matrix, where the values in each row are 1 if the word appears in the tweet and 0 if it does not.

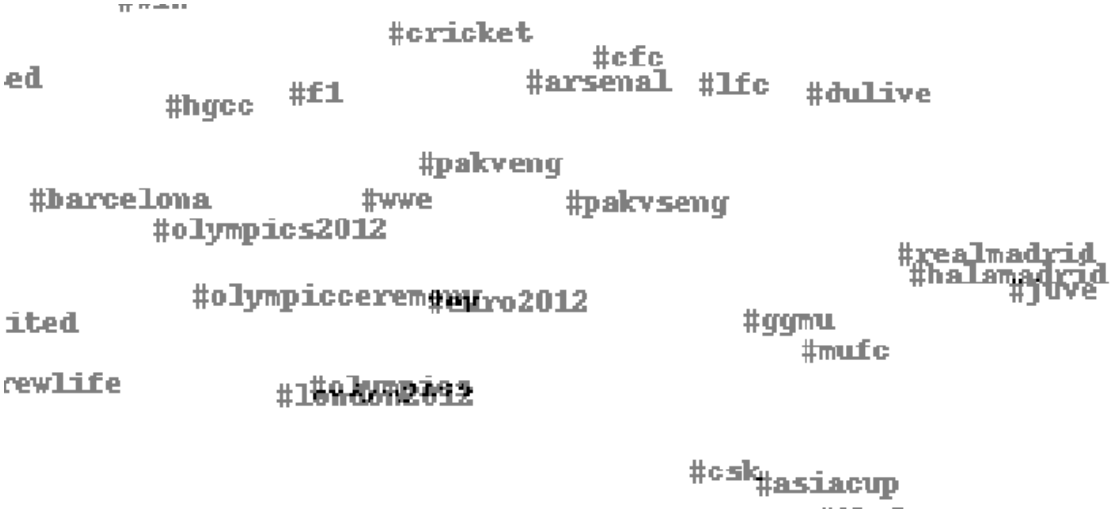


Figure 6: Sports-related hashtags, with bag-of-words embeddings.

Figure 7: Food related hashtags, with bag-of-word embeddings.

It is a lot easier to visualize the topics of the hashtags using the bag-of-word hashtag embeddings than it is using the distributional hashtag embeddings. Looking at Figure7, it is a lot easier to see groups of related hashtags that are close together than it is in Figure

6. It seems like hashtags are a better indicator for topic of the tweet but the position of a hashtag in a tweet is not very important. Figure 8 shows this this holds for foods as well, while Figure 9 shows the embeddings as a whole.

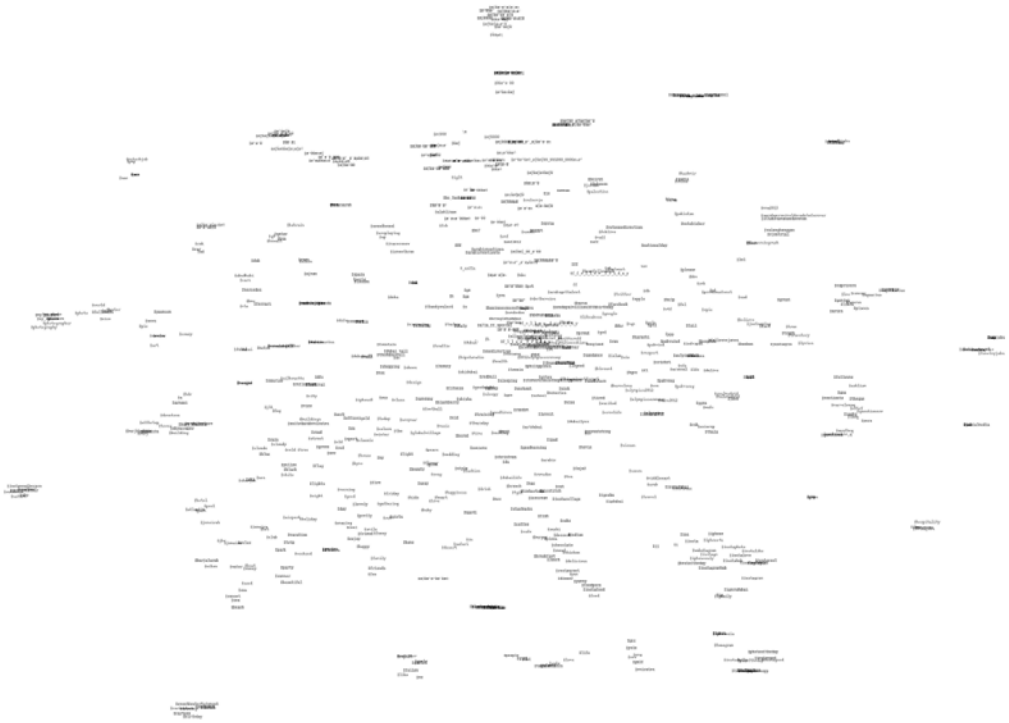


Figure 8: The bag-of-word hashtag embeddings look more separated than the distributional hashtag embeddings (in Figure 2) and it is a lot easier to tell where related topics are.

Word Embeddings for OSI Twitter Text

We also attempted to use word embeddings to identify protest-related tweets. One difficultly in identifying protest-related tweets is that some of the query terms are ambiguous. One Spanish word that we studied in particular is *paro*, which has several meanings. Here are some examples:

- *un paro* – a strike (e.g., *se va a paro*– will go on strike)
- *en paro* – unemployed (e.g., *la oficina del paro* – unemployment office)
- *un paro cardíaco* – a heart attack (literally)
- *Casi me meuro de un paro.* – I almost had a heart attack. (colloquially, not literally)
- *cobrar el paro* – on the dole
- *tasa de paro* – unemployment rate
- *paro la policia* – stop by the police

The goal is to find an unsupervised clustering of the different instances the word "paro" is used, and then identify which clusters are good to use as query terms. Ideally this technique would be something we could adapt to different ambiguous query terms, as well as to Twitter data in different languages. The first approach we attempted was to first do the first step of the two-step CCA, in order to find representations of the context of different words in the tweets and then, do k-means on the contexts surrounding different instances of the word "paro". This makes it possible to split the instances of the word "paro" into several different word types.

The problem with this approach is that there is not enough context. Since Spanish has more articles and other function words than English, just looking at the words to the left and to the right don't give enough context. A better approach was to add a step after doing CCA to combine the left and right context into the state matrix S, doing CCA between the rows in S that correspond to instances of the word "paro" being used, and a hashtag matrix H that corresponds to the hashtag that is used in that tweet (adding a null hashtag for tweets with no hashtag). The CCA between S and H finds matrices A and B, which are the basis that best correlates these two views of the tweet. Then we multiply S*A and do k-means clustering on the rows.

This seemed to help the most to separate the civil unrest related senses into separate clusters from the rest of the senses of the word "paro."

4.3 Graph Matching / Parallel Assignment

Application to XDATA datasets.

Typically graph matching is performed to find actual correspondence between the entities represented by the nodes in two or more graphs. While none of the XDATA problem sets provide this exact problem, a similar application would be to resolve entries in one social network to another, assuming that the networks have overlapping users.

The problem that we were more immediately interested in addressing concerned the Akamai traceroute dataset. In this data, we knew the identity of the various IP addresses. Thus the goal was not to match a known with an unknown, or two unknowns. Instead, we sought to use graph matching to align potentially dissimilar IP addresses to each other based on the structure of the network. Nodes that had similar patterns of neighbors in different graphs (representing different geographical areas or the same geographical area during different periods of time) would then be matched. Given this matching one could infer which IP addresses play structurally similar roles in different networks, or identify and describe shifting or anomalous patterns in a particular network. Generally this matching could serve as a preliminary step for any number of inference problems.

The speed of our initial implementation, however, made this process tedious on graphs of approximately 7,000 nodes, so we shifted our focus to addressing the speed issues. The linear assignment problem (LSAP) is also generally useful and thus achieving better parallel performance would be useful for a number of applications.

4.4 Network Inference of Link Strength(NILS)

During the duration of this effort, the following items were completed:

1. Proposed Kiva Ontology (see below)
2. Node properties extraction code
3. NILS code for building of augmented graph
4. Dashboard code (utilizing Gephi package) In addition, the live dashboard showing the results of NILS processing on two subsets of Kiva data were made available to BBN and the XDATA team

Application to Kiva Dataset

As an example of the connection between named entities, consider the following information about a particular borrower available in the XDATA KIVA data set.

> Text:
> *The Totorilla Community Bank has 24 members and is beginning its fifth cycle of loans. For the most part the members work in agriculture, growing vegetables, raising animals, as well as such activities as selling clothes, food, and groceries. 30 year-old Judith lives with her partner and has one son. She is a director of theBank. She grows alfalfa and vegetables and sells them in the market in Ayacucho. Her plan is to one day start a business buying and selling animals. 30 year-old Yolanda lives with her partner and has two children. She grows vegetables and sells them in the streets of the market in Ayacucho. The members of the Bank need various amounts of money, 300, 600, or 900 soles adding up to a total of 14,050 soles. The loans will be used to buy fertilizer, medicines, and to pay workmen. The members dream of building their own homes, providing their children with a good education, and having a vehicle to transport their products.*
>
> Borrower first name:
> *Judith*
>
> Borrower last name:
> *Ricela*
>
> Gender:
> *female*
>
> City:
> *Ayacucho*
>
> Country:
> *Peru*

This is only a sample of data about one specific borrower, but from this one article, we can already infer many relationships, *viz*. that Judith is associated with Totorilla Community Bank (TCB for short) and that her connection is likely to be quite strong as the bank only has 24 members and she is one of its directors. We also learn that Yolanda is also a member of the bank – so Judith is likely connected to Yolanda as well even though the piece does not explicitly state that. Both are members of TCB (which only has 24 members), both live in the same city, and both sell produce at the Ayacucho market. In addition, the context of this post – which includes metadata about the individual stored in relational tables – gives us information about the person's age, gender, etc.

Kiva Ontology

The proposed ontology will consist of the following three types of data objects:

1. Entities: Entities will be the vertices of the social graph and will include everything from people to organizations.

2. Relationships: Relationships will constitute the edges of the KIVA social graph and will include different types of relationships between the entities in the Kiva social graph (KSG). Edges in KSG will be labeled with the semantic nature of the relationships between KIVA entities.

3. Properties: Vertices in the Kiva network may have different properties. The property labeling in the Kiva ontology will describe properties of these entities.

Entities: Any named entity mentioned in the Kiva data will be considered to be an entity in the KSG. Examples of entities we have seen in the Kiva data include:

- MerardoParedes
- Al Majmoua
- Virgin de la Mercedes (community bank)
- Finca

Though entity extraction is not perfect, once entities are extracted, we will pre-cluster similarly named entities together (please note that this is not the same as clustering entities that are not similarly named but may in fact be the same entity. “Virgin de la Mercedes” and “Virgin de la Mercedes community bank” may be extracted as entities in different parts of the text document. In this case, pre-clustering should cluster these similarly named entities together. Some of these entities can be extracted from the “Name” tag in the Kiva data but others will have to be extracted from the text.

Properties of Vertices: Once pre-clustering has been done, the term “vertex” refers to each “cluster” of similarly-named entities. Each vertex has various properties that we would like to extract over time. These include:

- Type: This will include the following
    - Person: true/false
    - Bank: true/false
    - KPO (Kiva partner organization): true/false
    - Business: a type of business

• Attributes:

- Country: string [can be extracted from “Country” tag]

- Town: string [can be extracted from “Town” tag]

- Latitude: real [can be extracted from “geo” tag]

- Activity: type of activity engaged in by the entity – may be extractable from the “Activity” tag but also need to look into the text for it.

- Company-owned: string denoting the name of the company owned by the vertex (if applicable)

- NumLoans: Number of loans the person received – can be extracted using the “Loan id” and “person” tag

- LoansValue: Total value of the loans the person has received. Can be extracted using the “loan id” and “funded amount” fields

- NumPaidUpLoans: Number of loans received by the person that are marked “paid”. Can be gotten from the “loan” status field. This only refers to fully paid up loans.

- LoanPercentagePaid: This is more complex to calculate but would represent the ratio of how much the entity paid to how much the entity actually owes at the time the Kiva data was made available. All loan amounts – where possible – should probably be computed using local currency to avoid getting messed up by current conversion losses that are also reported in the data.

Relationships: The Kiva data obviously has a huge wealth of data about relationships between people, banks, Kiva partner organizations, and businesses. This leads to 10 possible pairs of (undirected) relationships.

Person-Person relationships	• Significant---other [spouse/girlfriend/boyfriend/live--- in] • Other---family [family relationship other than S.O.]
Person-Bank relationships	• Member (person is a client of the bank) • Officer (person is an officer of the bank) • Borrower (person is a borrower from the bank)
Person-Business relationships	• Owner (person is either a full or part owner of the business) • Employee (person works at the business) • Customer (person is a customer)
Person-KPO relationships	• Affiliated (person is affiliated with the KPO) • GotLoan (person previously got or has a loan through the KPO) • AppliedLoan (person applied for a loan from the KPO)
Bank-Bank relationships	• DidBusiness (the two banks did business together – now or in the past)
Bank-KPO relationships	• DidBusiness (bank and KPO did business together – now or in the past)
Bank-Business relationships	• DidBusiness (the bank and business did business together)
KPO-KPO relationships	• DidBusiness (the two KPOs did business together – now or in the past)
KPO-Business	• DidBusiness (same as above – KPO and business did business together in the past or current)
Business-Business Relationships	• DidBusiness (the two businesses did business together)

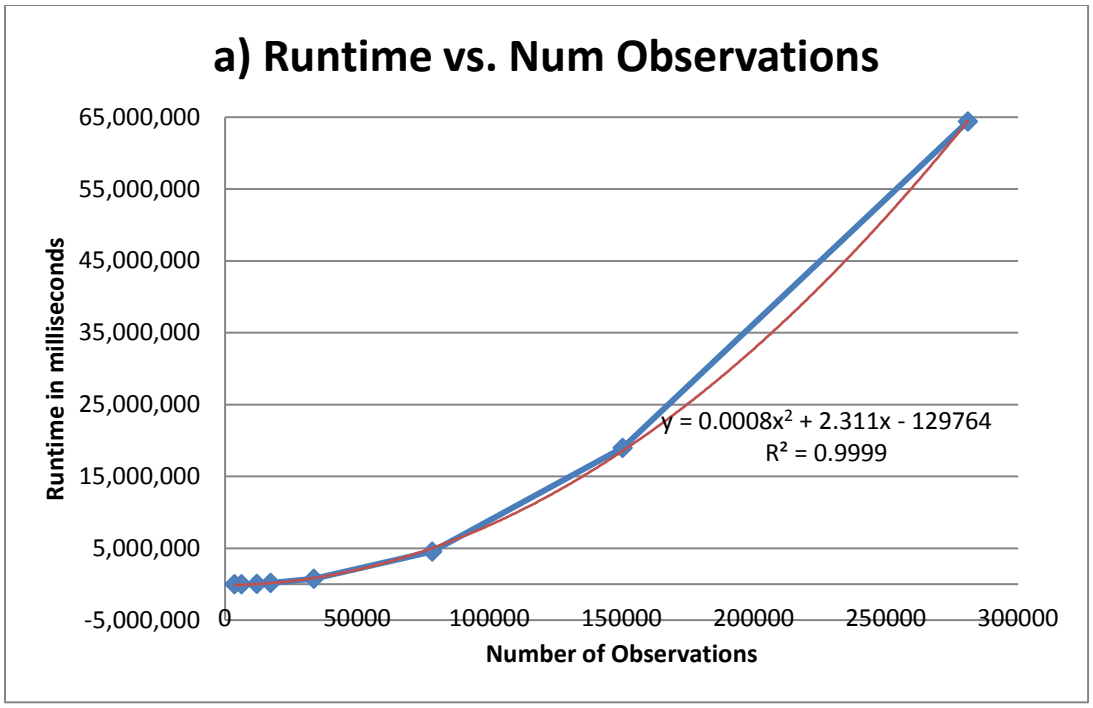
4.5 Flickr Anomaly Capture Engine

As Flickr has a huge number of pictures, specialized index structures needed to be developed to for any of the aforementioned computations. By the time the project pivoted to our work with the Kiva dataset (May 2013), the app to collect Flickr data was up and running Data cleanup of the XDATA Flickr set was done. Sample data set selection was in progress.

5 Conclusions

5.1 Patterns in Near-real Time (PINT) Algorithm

Our experimentation with PINT confirmed that the slowest part of the algorithm is the first step, the creation of the affinity graph. We found that it consumed about 80% of the runtime of PINT. Indeed, this is an N^2 step with respect to the number of bindings N (where #bindings <= #observations * #activities in a process). Steps 2 and 3 are approximately linear, according to the experiments. Step 3 seems to be very fast, consuming only about 0.4% of the total runtime, while Step 2 consumes approximately 19% of the PINT runtime. As seen in Figures 1 and 2, the overall performance is quadratic with a very small leading coefficient. The following figures demonstrate the performance in terms of runtime and the number of processes found.



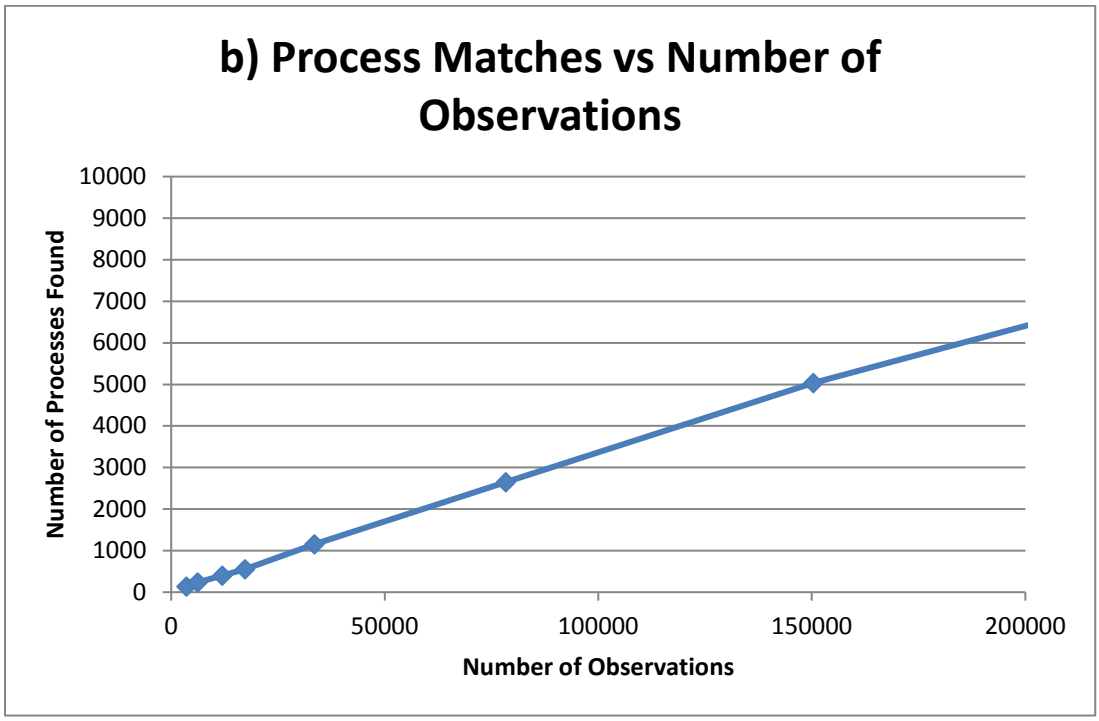
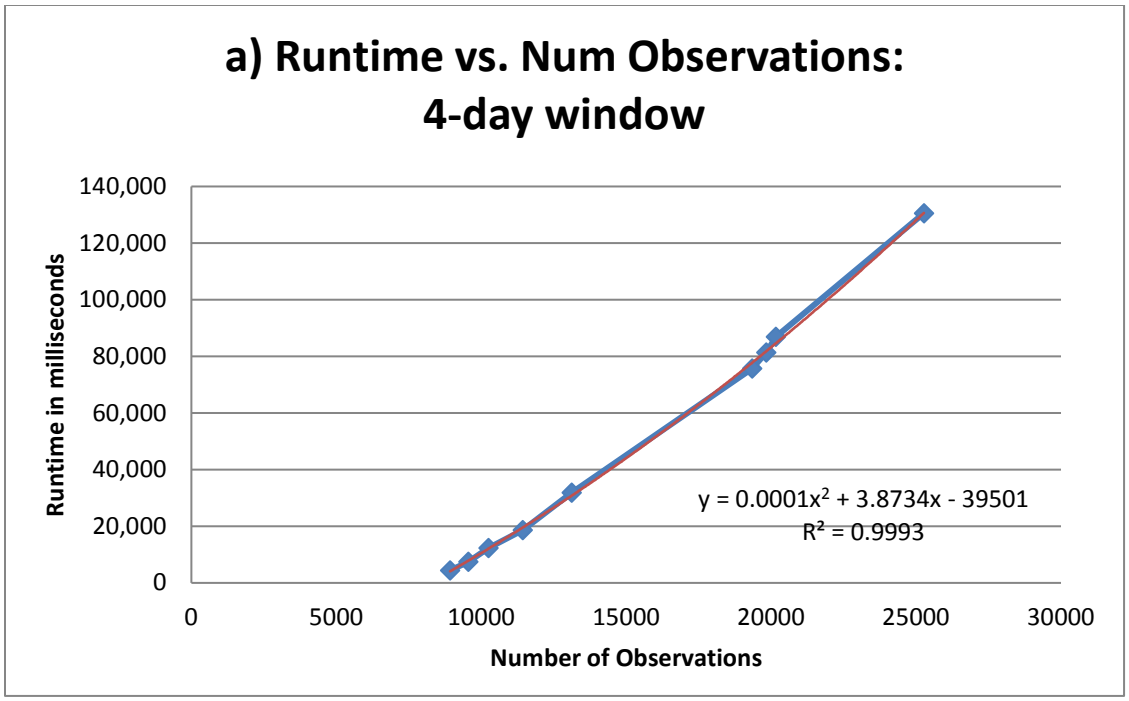


Figure 9: PINT performance over Twitter data. Note that: a) runtime is low-coefficient quadratic; b) processes matched are linear with the number of observations.



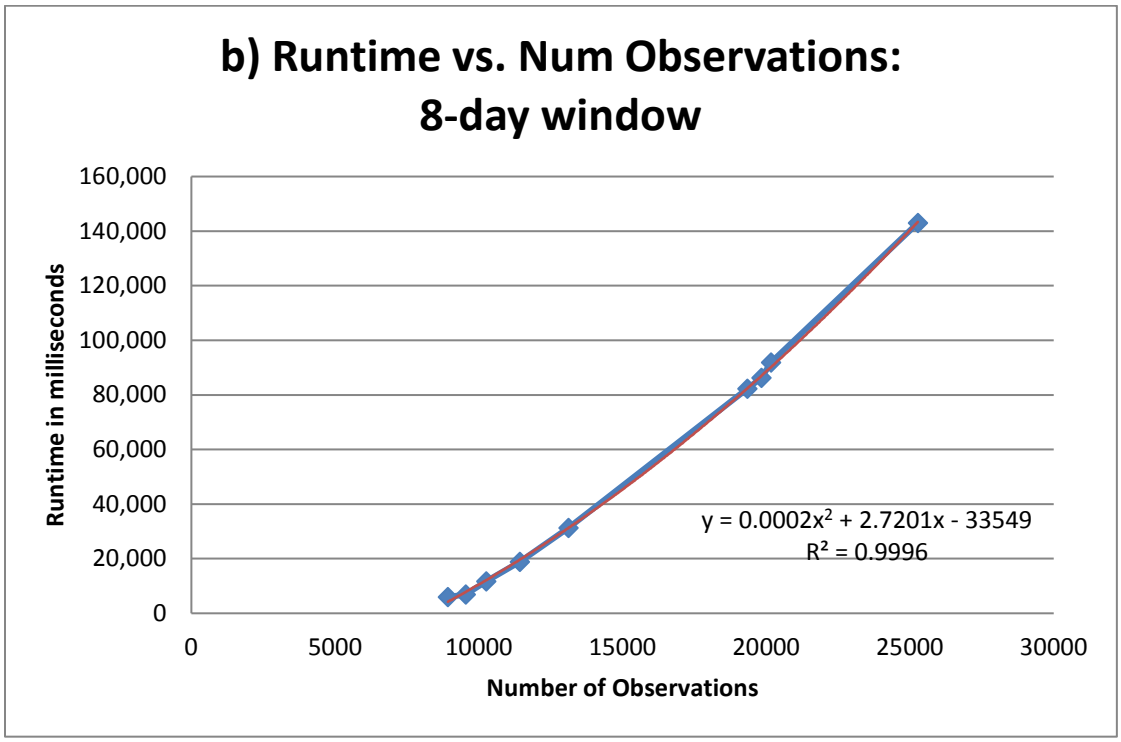


Figure 10: Pint performance over BitCoin data in terms of runtime vs. number of observations (after filtering transactions): a) using the 4-day window condition b) using the 8-day window condition.

6 Recommendations

6.1 Patterns in Near-real Time (PINT) Algorithm

6.1.1 Future Work

Parallelizing PINT

The main focus of our research into the parallelization of the PINT algorithm has been to find appropriately scalable matrix math algorithms. Recall that the PINT clustering algorithm, Power Iteration Clustering (PIC), a variant of spectral clustering, requires a calculation of the leading eigenvector of the adjacency matrix of the pairwise affinity graph.

We reviewed the matrix math implementation currently being used in PINT and started exploring parallel implementations. The current implementation uses a package called Matrix Toolkits Java[21] (MTJ). We use version 0.9.14 of this library and this implementation uses LAPACK and BLAS for computing eigenvalues[22]. Apparently this library also supports a native implementation of BLAS, but our simple test-setup for PINT does not currently use that, so the Java implementation, JLAPACK, is currently used. This leaves a lot of room for optimization.

We found that there is a newer version of the MTJ project, maintained on GitHub[23]. The main difference appears to be that the newer version better supports sparse vector/matrix computations using another package called netlib-java[24] (and Templates)[25]. Both the old and the new versions support a feature called matrix.distributed, which is some level of support for distributed matrix math; however our code is not using that feature right now.

Exploring parallel implementations of matrix math, we found several potential routes. One alternative would be to use Colt and its parallel version Parallel Colt[26], which uses multithreading, not distributed computation. Some Colt users point out that its performance leaves a lot to be desired. Additionally, there are other libraries (e.g. JBLAS[27]) that are also multithreaded and might be preferred. This route was abandoned in favor of exploring massively parallel implementations.

Another route in parallelizing PINT is to pursue massively distributed matrix math implementations, including those over the Map-Reduce framework. Recall that PINT's two core processes are the construction of a large similarity graph, where each node represents an observation-to-process-activity pair, and spectral clustering of the nodes

[21] https://code.google.com/p/matrix-toolkits-java/
[22] https://code.google.com/p/matrix-toolkits-java/wiki/FAQ
[23] https://github.com/fommil/matrix-toolkits-java
[24] https://github.com/fommil/netlib-java/
[25] http://www.netlib.org/templates/
[26] https://sites.google.com/site/piotrwendykier/software/parallelcolt
[27] http://freecode.com/projects/jblas

according to their similarity values. The steps that would benefit the most from parallelization (and are readily parallelizable) are:

1. Construction of the affinity matrix for observation-activity pairs (the adjacency matrix of the graph)
2. Computing the leading eigenvector of the affinity matrix in order to perform spectral clustering

We found two candidates for massively distributed matrix manipulation to be noteworthy, Apache HAMA[28] and Apache Mahout.[29]

We reviewed both Apache Mahout and Apache HAMA. It seems that HAMA could potentially yield a more efficient (faster) implementation; however, the Mahout package is more mature and better-supported. Our initial approach would be to test PINT with the spectral clustering algorithms that are already part of Mahout. If that approach does not result in sufficient performance boost, we will continue with a HAMA-based implementation.

GISR Dataset

We have started looking into applying PINT to other types of large data. In particular, we worked towards developing an interface to the multi-INT Global Intelligence, Surveillance and Reconnaissance (GISR) dataset. The availability of different types of sensor data will be beneficial in terms of demonstrating PINT not only as an "anomaly detection" algorithm, but also as a data fusion tool, whereby it can discover processes in temporally-aligned data from disparate sensor modalities.

We reviewed the GISR dataset, made available on a separate VPN enclave at the XDATA lab. We developed a plan for testing our algorithms with the GISR data, which involves the following steps:

1. Analyze the schemas of the various data sources that are part of GISR
2. Develop the appropriate data conversion scripts to feed the filtered data into PINT observation sets for initial testing
3. Perform filtering in order to extract data samples suitable for the PINT algorithm (PINT requires location, timestamp and type of observation)
4. Perform testing and benchmarking in the lab
5. Apply PINT on the entire GISR dataset (or select data source in the dataset, consuming all available data).
6. Report findings

[28] http://wiki.apache.org/hama/SpMV
[29] http://mahout.apache.org/

7. Review the PINT algorithm for bottlenecks and possible optimizations

We completed step 1 and started work on steps 2 and 3. Due to the necessity to be physically in the lab, we did not have a chance to test our conversion and filtering scripts on the actual GISR data. Having reviewed the schemas and the available data, we feel confident that the PINT algorithm will be applicable to the Hydra-Magic data, the VEGA tracks data and the SRHawk track data. Our desire in applying PINT to this collection of sources is to combine phone calls and text messages with Track data in order to uncover processes of interest, such as congregation/loitering, forming routes and route deviation. As part of this analysis, a PINT pre-processor will perform fusion of the phone call and text messages with track data observations so that the PINT process matcher can operate over the fused data.

User Interface

Finally, we explored some pre-existing user interface code for PINT. Unfortunately, the code is fairly dated, implemented in Java Swing, and represents a fairly limited interface for displaying observations and process matches (solutions). A screenshot is shown on Figure 3 below:

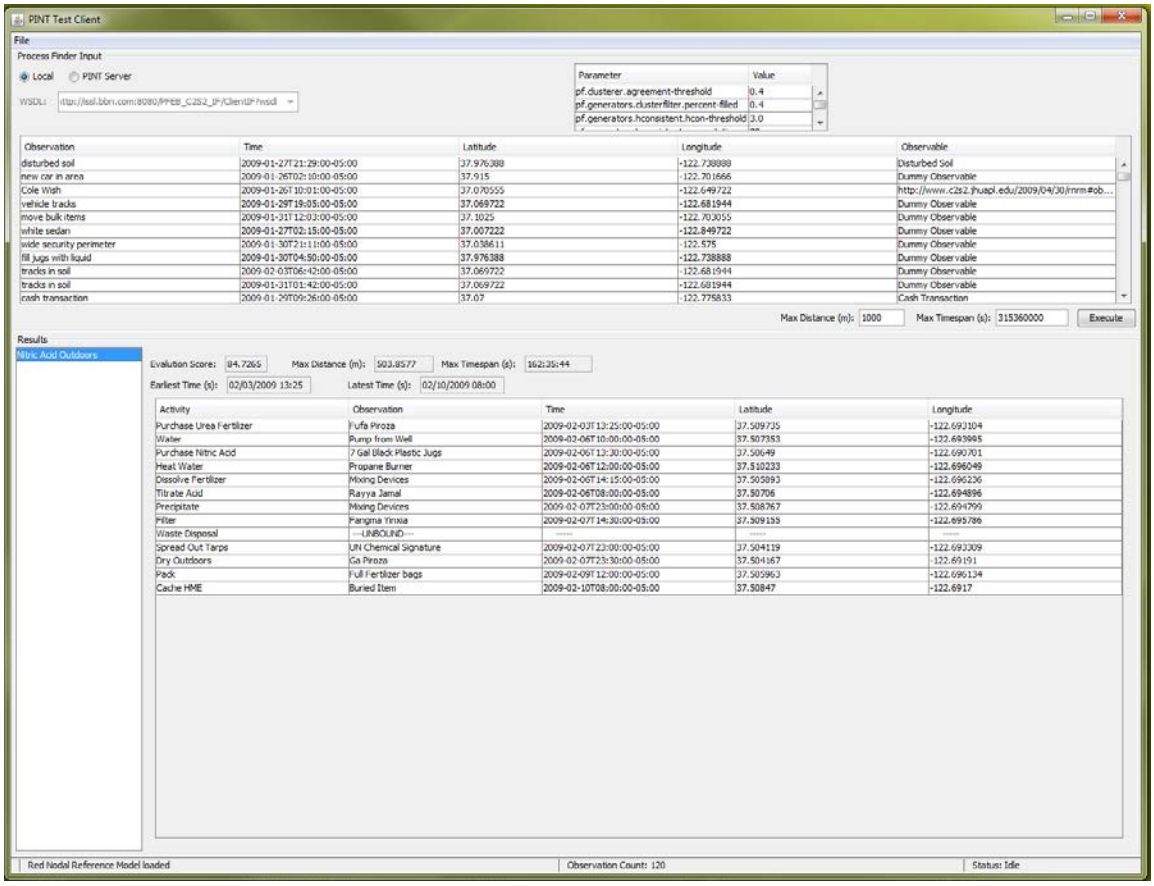


Figure 11: PINT User Interface for Testing Purposes.

7 Appendix

Additional Software Submitted

In addition to the software corresponding to the research efforts described above, the following software was also made available on the BBN Github.

Wordcloud

This package generates an HTML word cloud in which word size is proportional to a score. The score is computed using a modified version of TF-IDF, in which term frequencies are taken from one set of documents (the "documents of interest") and document frequencies from another (the "overall collection").

This scoring method emphasizes words that are prevalent in the documents of interest relative to the overall collection. (If both sets of documents are the same, then the score is a conventional TF-IDF score.)

The package is designed to use Hadoop streaming. It consists of a driver script and several helper scripts.

Text Structure

This package provides various options for performing unsupervised clustering (LDA, PLSA) on text documents and then using the Chow Liu algorithm to create a tree showing joint probabilities between the topics. It uses the Vowpal Rabbit Executable.

8 List of Acronyms

CCA – Canonical Correlation Analysis
FACE – Flickr Anomaly Capture Engine
GISR - Global Intelligence, Surveillance and Reconnaissance
KPO – Kiva Partner Organization
KSG – KIVA Social Graph
LDA – Latent Dirichlet Allocation
MTJ – Matrix Toolkits Java
NILS - Network Inference of Link Strength
NLP – Natural Language Processing
PIC – Power Iteration Clustering
PINT - Patterns in Near Real Time
pLSA – probabilistic Latent Semantic Analysis
SMEs - Subject Matter Experts
SVD – Singular Value Decomposition