Set-Differencing Data Structures for Social Network Analysis

David Eppstein

January 10, 2012

Based on work by...

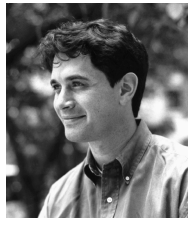
Joe Simons
UCI/MURI

George Varghese
UCSD

Frank Uyeda
UCSD

David Eppstein
UCI/MURI

Mike Goodrich
UCI/MURI



Michael Mitzenmacher
Harvard

Pierre Baldi
UCI

From papers in:

WADS'07/TKDE'10
SIGCOMM 2011
Allerton 2011

and two works in progress

The basic problem:

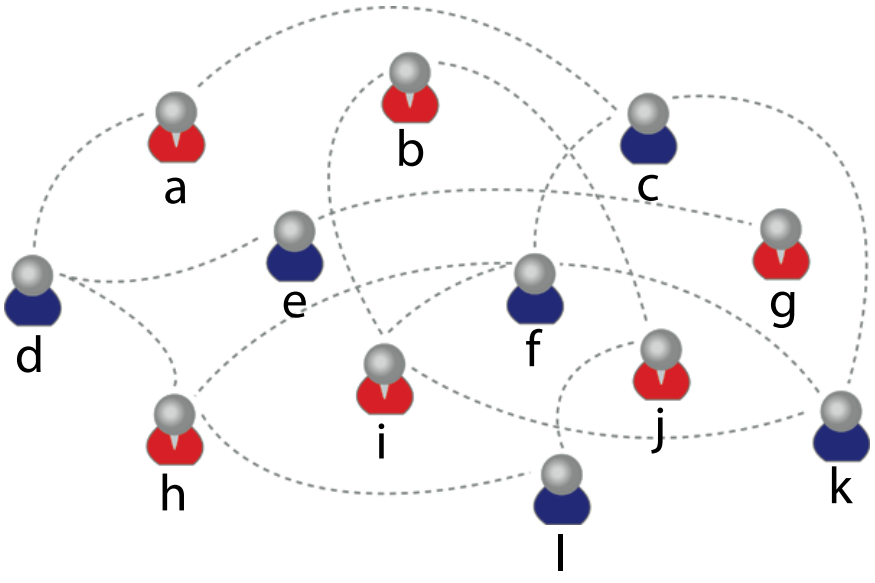
Given multiple sets of objects

Find a compact "sketch" of each set that allows us to:

- Find the elements that belong to one set but not both
- Estimate the Hamming distance between sets

Sets of what?

For social network analysis, use sets of edges in a network



{ac, ad, bi, bj, cf, ck, de, dh, eg, fh, fi, fk, hl, ik, jl}

Different networks with the same actors (e.g. as measured at different times) give us different sets to be compared

We may also compare synthetic networks (time steps of an ERGM simulation) or store more complex objects (ERGM features)

A (made up) example

Suppose the networks to be compared are the contacts among people attending/teaching at a school, sampled daily

A (made up) example, continued

By calculating the level of similarity between networks for different days (e.g. the Hamming distance of their sets of edges) we might learn

- Whether some days of the week have different class schedules than other days
- Whether there were any days in which the communication was significantly disrupted from its typical pattern for that day
- What anomalous communication events occurred that did not fit the typical pattern for that day

A naive solution

Why not store sets as lists of elements, and compare by checking which elements belong to each set?

- Linear time – too slow to compare many pairs of sets
- Not space-efficient for storing many similar sets

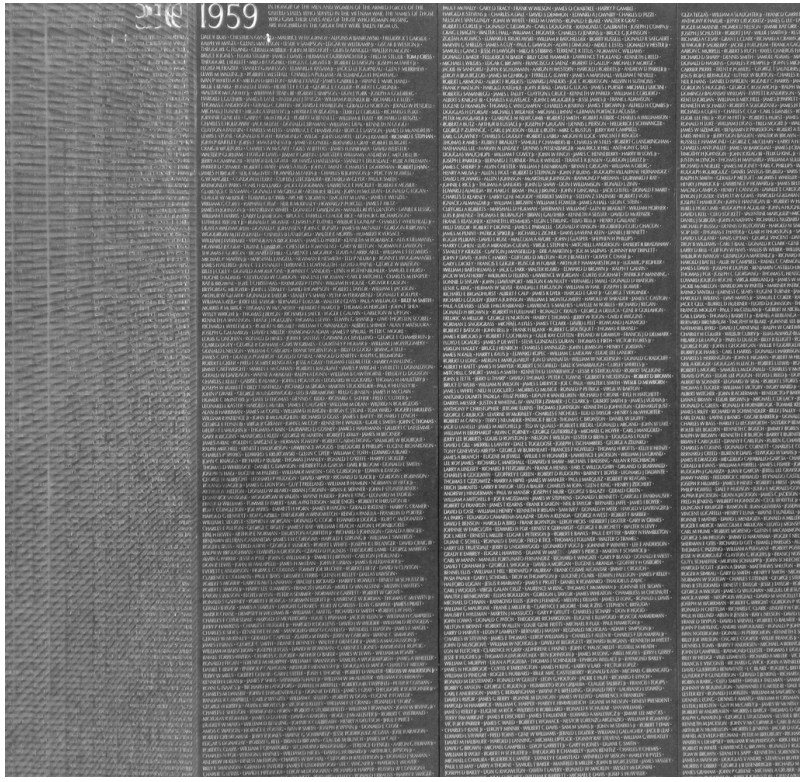


Photo by Laura Padgett from Flickr

Another slightly-less-naive solution

Ok, then, what about storing the sequence of changes from one set to the next?

- Makes additional assumptions that the sets form a time series and that the most similar sets to each other are adjacent in the time series
- Comparisons may be inefficient if many differences cancel each other (short-term changes that are quickly reverted or near-periodicity in the time series)

MinHash (Broder 1997)

Represent each set by the k smallest hash values of its elements

To estimate how similar two sets are, count how many hash values they have in common

Fast, good for estimating large distances (proportional to set size) but inaccurate for smaller distances, and doesn't provide the elements of the difference

Our solutions

For finding the elements in the set difference:

- Store each set in a data structure of size $O^*(k)$, for arbitrary but predetermined value k
- In $O^*(k)$ time, either report all difference elements or determine that there are more than k of them.

For estimating the size of the difference:

- Store each set in a data structure of size $O^*(1)$
- Compute $(1+\epsilon)$-approximation of Hamming distance in time $O^*(1)$, for arbitrary but predetermined value $\epsilon > 0$.

* With high probability, omitting factors polynomial in $\log n$, ϵ, and the logarithm of the failure probability

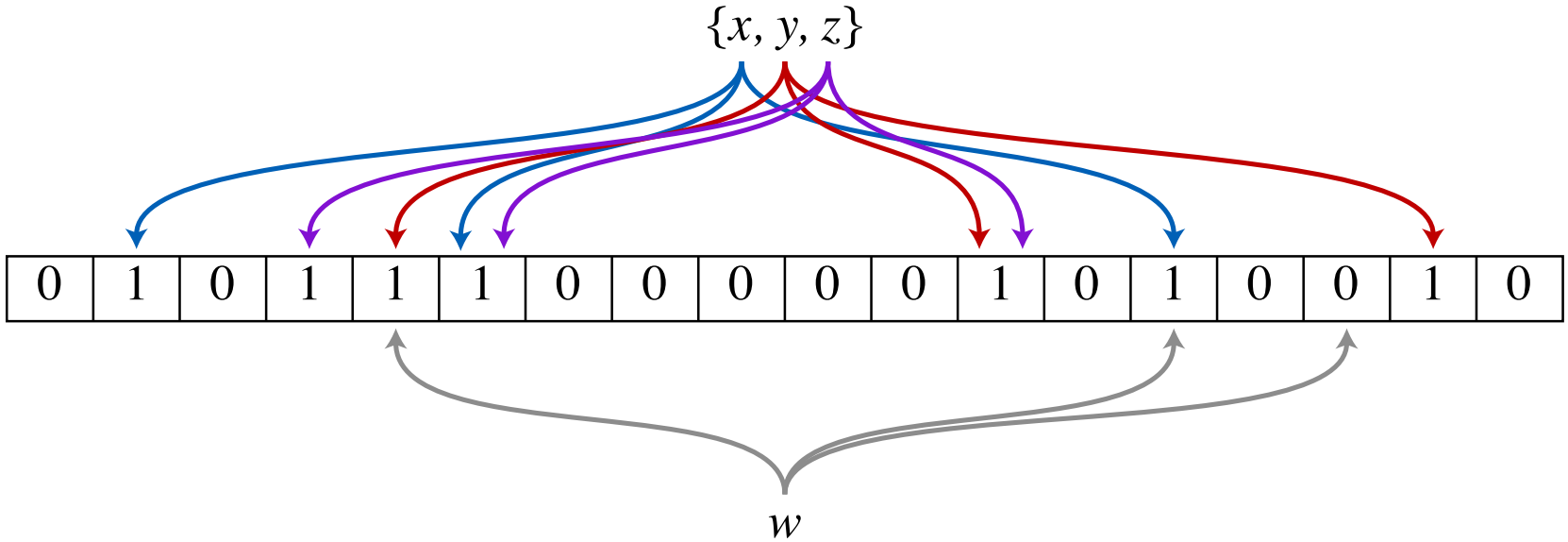
Comparison of solutions

Suppose we have a data set with t networks, each having roughly m edges, and that we want to estimate the distance between every pair of networks

- The naive solution takes time $O(mt^2)$, and computes the distance exactly
- Storing change logs may or may not be faster than the naive solution, hard to analyze
- MinHash takes time $O^*(mt + t^2)$, close to the input size, but computes the distances to within an additive error of $\pm\epsilon m$, accurate only for large distances
- Our solution takes time $O^*(mt + t^2)$ and computes the distances to within a multiplicative factor of $(1 \pm \epsilon)$

How it works (I)

Bloom filter: array of $O(k)$ bits representing a k-element set



Each set element is mapped by a hash function to a constant number of array positions, which are set to one

An element not in the set is unlikely to have all positions nonzero

How it works (II)

Invertible Bloom filter: replace the bits of a Bloom filter by cells holding three values:

- The number of elements mapped to that cell
- The sum of the elements mapped to that cell
- A checksum, the sum of hash values of elements mapped to that cell

Each set element is likely to have a *pure* cell that only it maps to, with element count one and with a valid checksum

An element not in the set is unlikely to appear to be in a pure cell

How it works (III)

Because the cell values in an invertible Bloom filter are additive (sums of element values), we can add and subtract IBF's

The difference of two IBF's is itself an IBF representing the symmetric difference of sets
(with some element counts -1 instead of $+1$)

If the difference has at most k elements, we can decode it by looking for pure cells

If this decoding algorithm fails, then with high probability the difference has size $> k$

How it works (IV)

To estimate the size of the difference of two sets to within a $(1+\epsilon)$ factor, even when the difference is large:

- *Consistently partition* the two sets into samples of approximately $n/2$, $n/4$, $n/8$... elements, according to the number of trailing zeros in a hash value for each element
- Store each sample in an IBF of capacity $O(1/\epsilon^2)$; represent each set by these IBFs of its samples
- Subtract the IBFs representing one set from the IBFs representing the other set
- Use the densest sample for which the subtracted IBF can be decoded to estimate the difference

Variations

We have also looked at the following related problems, less directly relevant to social networks:

- Store sets on different computers across the internet, and communicate their difference in an amount of communication proportional to the difference (distributed version control)
- Combine these data structures with homomorphic encryption to prevent eavesdroppers from being able to learn the sets
- Combine these data structures with geometric range searching techniques to report or estimate the number of differences in a geometrically-restricted subset of two input sets
- Transform DNA sequences into sets in a distance-preserving way, in order to use these data structures for biological sequence comparison

Conclusions

Data sets consisting of many networks on the same set of actors can be stored efficiently using very little storage per set, but still allowing useful information about the comparison between two of the networks to be extracted

Similar techniques have been applied to a wide class of other problems (with sets of objects other than network edges) and have been successfully implemented

Implications for social network analysis look promising but are still speculative