Efficient Computation of Change-Graph Scores

David Eppstein

(includes joint work with Emma Spiro, Mike Goodrich, Darren Strash, Lowell Trott, and Maarten Löffler)
Context: analysis of social networks

Represent interactions among people and their environments as graphs

(often: vertices = people, edges = pairwise interactions)

Goals:

Predict human behavior

Detect anomalous behavior

Handle varied types of graph data and scale well to large networks
Mathematical modeling of social networks

Develop mathematical models with a small number of meaningful numerical parameters that generate graphs resembling real social networks

Why?

- Fitting the parameters to real data tells us how real social nets behave
- The parts of the real networks that do not match the model may be anomalous
- We can use the model to generate test data for other analysis algorithms

Not a pipe, but a model of a pipe

René Magritte, *The Treachery of Images*, 1928–9
Exponential random graph model: graphs shaped by their local structures

Define local features that may be present in a graph:

- Presence of an edge
- Degree of a vertex
- Small subgraphs

Assign weights to features: positive = more likely, negative = less likely

Log-likelihood of $G = \text{sum of weights of features} + \text{normalizing constant}$

Different feature sets and weights give different models capable of fitting different types of social network
Probabilistic reasoning in exponential random graphs

Most basic problem: pull the handle, generate a random graph from the model

With a generation subroutine, we can also:

- Find normalizing constant
- Fit weights to data
- Understand typical behavior of graphs in this model (e.g. how many edges?)
- Detect unusual structures in real-world graphs
Standard method for random generation: Markov Chain Monte Carlo (random walk)

Start with any graph

Repeatedly choose a random edge to add or remove

Calculate change to log-likelihood

Choose whether to perform the update (positive change score: always perform, negative change score: sometimes reject)

After enough steps, graph is random with correct probability distribution
The key algorithmic subproblem:

Add and remove edges in a dynamic graph

At each step, update feature counts (how many of each type of small subgraph it has)

A telephone switchboard, an early example of a dynamic graph

Photo by Joseph A. Carr, 1975, available online under a free license at http://commons.wikimedia.org/wiki/File:JT_Switchboard_770x540.jpg

Because this is in the inner loop, it must be very fast
MURI-funded work on this problem:

The $h$-index of a graph and its application to
dynamic subgraph statistics (with E. S. Spiro)
Presented at WADS, Banff, Canada, 2009.

Shortlisted for best paper award.
Undirected graphs, feature = subgraph with $\leq 3$ vertices

Extended dynamic subgraph statistics using $h$-index
parameterized data structures
(with M. T. Goodrich, D. Strash, and L. Trott)
in preparation

Directed graphs, larger numbers of vertices per feature
See poster session

New research still under development (with M. T. Goodrich, M. Löffler)

Geometric graphs and geometric features
MURI-funded work on this problem:

The $h$-index of a graph and its application to dynamic subgraph statistics (with E. S. Spiro)
Presented at WADS, Banff, Canada, 2009.

Shortlisted for best paper award.
Undirected graphs, feature = subgraph with $\leq 3$ vertices

Extended dynamic subgraph statistics using $h$-index parameterized data structures
(with M. T. Goodrich, D. Strash, and L. Trott)
in preparation

Directed graphs, larger numbers of vertices per feature
See poster session

New research still under development (with M. T. Goodrich, M. Löffler)

Geometric graphs and geometric features
Interdependence among 3-vertex feature counts

\[
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 \\
0 & 1 & 2 & 3 \\
0 & 0 & 1 & 3 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
\text{number of triangles}
\end{bmatrix}
= 
\begin{bmatrix}
\frac{n(n - 1)(n - 2)}{6} \\
m(n - 2) \\
\sum \deg(v) (\deg(v) - 1)/2 \\
\text{number of triangles}
\end{bmatrix}
\]

So if we can maintain the number of triangles in a dynamic graph, we can easily compute all other counts.
Degree-based partitioning of a graph

Select a number $D$

Partition vertices into two subsets:
- $L$: many vertices with degree less than $D$
- $H$: few vertices with degree greater than $D$
What we store:
Number of paths through low-degree vertices

Maintain hash table $C$ indexed by pairs $(u, v)$ of vertices

$C[u, v] = \text{number of two-edge paths } u \rightarrow \text{Low} \rightarrow v$
When edge \((u, v)\) is added or removed:

The number of triangles with the third vertex in \(L\) is stored in \(C[u, v]\) (look it up there)

The number of triangles with a third vertex \(w\) in \(H\) can be counted by examining all possibilities for \(w\) (loop over all vertices in \(H\) and test whether each one forms a triangle)

If \(u\) belongs to \(L\), add degree\((v)\) to \(C[u, w]\) for each neighbor \(w\) of \(u\) (perform a symmetric update if \(v\) belongs to \(L\))

(Very infrequently) update the partition into low and high degree
How much time does it take per change?

Finding triangles involving changed edge takes $O(|H|)$

Each edge is involved in $O(D) x\rightarrow L \rightarrow x$ paths, so updating hash table after a change takes $O(D)$

If $L/H$ partition ever changes, update counts for all $x\rightarrow L \rightarrow x$ paths through moved vertex taking time $O(D^2)$

How to choose $D$ so $|H| + D$ is small and partition changes infrequently?
A detour into bibliometrics

How to measure productivity of an academic researcher?

Total publication count: encourages many low-impact papers

Total citation count: unduly influenced by few high-impact pubs

\( h \)-index [J. E. Hirsch, PNAS 2005]:
maximum number such that \( h \) papers each have \( \geq h \) citations
The $h$-index of a graph:

Maximum number such that $h$ vertices each have $\geq h$ neighbors

$H =$ set of $h$ high-degree vertices
$L =$ remaining vertices, degree $\leq h$

Provides optimal tradeoff between $|H|$ and $D$

Never more than $\sqrt{m}$
Else $H$ would have too many edges
Results:

We can maintain the $h$-index of a dynamic graph in constant time per update (details beyond the scope of this talk).

A relaxed degree partition based on the $h$-index changes very rarely. On average, some vertex changes sides once in every $O(h)$ updates.

As a consequence, we can maintain triangle counts and change scores in time $O(h)$ per update.

All algorithms are simple and implementable.

Later work (Trott poster) generalizes this to more complex features.

Still need to do: implement them and test their actual performance.