Partial View Selection for Evolving Social Graphs

Georgia Koloniari and Evaggelia Pitoura

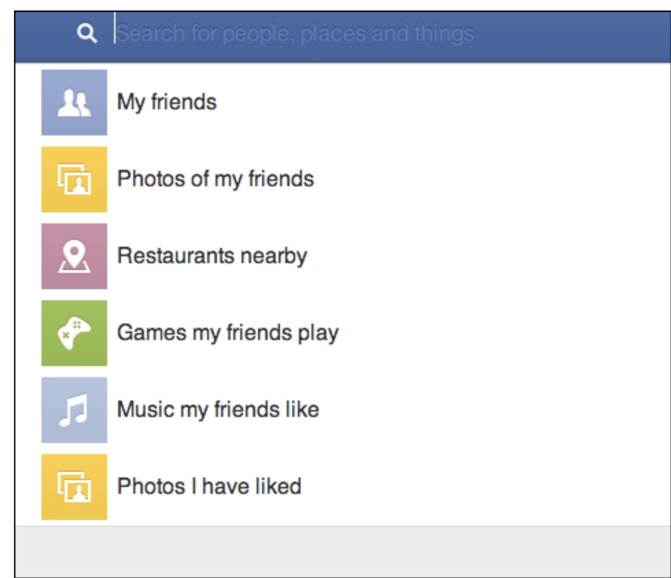
University of Macedonia University of Ioannina

Greece

Introduction

- Social networks represented as graphs $G(V,E)$: V set of users and E set of edges representing the social relationships between users
  - Large scale
  - Very dynamic: evolving through time

- Users query the social network graph, eg. Facebook Graph Search
  - Friends of my friends who visited NYC, New York
  - My friends who live in Thessaloniki and visited NYC, New York



Can we add time to graph search?

Historical Queries:

Queries about the state of the graph in the past

Examples:

- Friends of my friends who visited NYC, New York last year?
- My friends in May 2010 who have visited NYC, New York
- My friends in May 2013?
- Who are the new friends I acquired from March 2013 to June 2013?

But also…

- What was the diameter of the social graph in March 2013?

How do we capture graph evolution?

Graph Snapshot + Graph Log

- Graph snapshot SG_t: snapshot frozen at time t
- Graph Log: update operation + timestamp
  - Add / remove node - Add / remove edge

We require for the graph log to be:

- Complete: maintains all the necessary information to construct a snapshot
- Invertible: can be used for both forward and backward snapshot construction in time

We prove that by storing one snapshot and the graph log for a time interval we can construct any other snapshot in this time interval

Thus, we only store:

- Graph log for time interval $[t_0, t_{cur}]$
- Current Graph Snapshot $SG_{t_{cur}}$

How do we evaluate queries on evolving graphs?

- Usually, two steps:
  1. Construct the graph snapshots required for query evaluation
  2. Evaluate the query on the snapshots

- Snapshot construction is expensive
  - Apply the related parts of the graph log on the current snapshot to retrieve the past snapshots

Query Types

- Global queries
  - compute global properties of G -- traverse the entire graph
  - Examples:
    - What is the diameter of G?
    - What is the degree distribution in G?, etc..

- Targeted queries
  - User-centric queries – traverse only a specific subgraph of G
  - Examples: Queries similar to Facebook graph search
    - Find my friends that live in NY
    - Find the friends of my friends that are interested in graph management, etc...

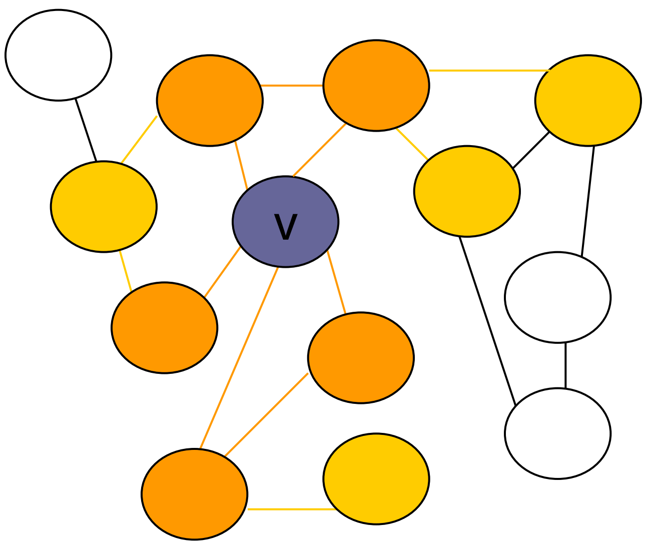
Basic Idea

- For targeted queries, full snapshot construction is redundant
- Instead, construct only the specific subgraph targeted by the query

$\Rightarrow$ Construct the appropriate partial view!

Partial Views

- Partial Views modeled as Egonets
- Egonet(v, R, t)
  - Node v center of the egonet
  - R radius of the induced subgraph
  - t time point at which the egonet is valid (i.e. Egonet a subgraph of SG_t)



Egonet of v with R=1

Egonet of v with R=2

How can we use a partial view?

- Model targeted queries as egonets similar to partial views
- Given a query q, construct the partial view the query requires
  - view construction: apply only the related parts of the log file
- Evaluate the query on the derived partial view

Can we reuse materialized views?

- Determine when a materialized partial view (egonet) can be used to evaluate a query
- We define view subsumption between partial views

Given two partial views, EG_1 and EG_2, EG_1 subsumes EG_2, if the result of the evaluation of any targeted query q on EG_2 is equal to the result of evaluating q on EG_1.

Also:

- Derive new views from materialized views
- Define view extension:
  - In radius
  - In time

Which views should we materialize?

The View Selection Problem

Given the current graph snapshot, the graph log and a set of N targeted queries, select from the set of corresponding query egonets a set C of K egonets, K < N, such that, if the egonets in C are materialized, the total evaluation cost of the query workload is minimized.

Selection Algorithms:

- Exhaustive: considers all possible subsets of K egonets
- Random: randomly select K egonets
- Greedy: at each step, select to materialize the egonet with the maximum construction cost

We propose two-phase greedy selection

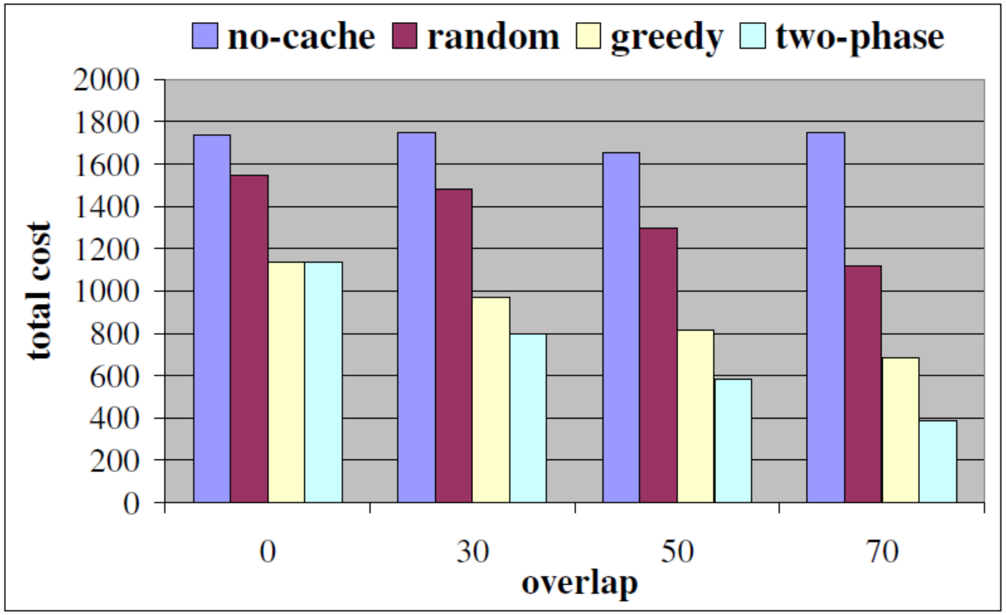
Two-Phase Greedy Selection

- Group egonets according to their center
- At each iteration
  - For each group
    - Select the egonet with the greatest construction cost
    - Re-evaluate the total construction cost of the group
    - Compute the benefit for materializing the egonet
  - Select the group with the greatest benefit
  - Update all costs
  - Proceed to next iteration until K egonets are selected

View Selection Comparison

- Measure total view construction cost for a given query workload
- Data from New Orleans Facebook Network (Viswanath et al, WOSN 2009)
- x-axis: overlap among queries (% queries with the same center)
- y-axis: construction cost

Cache size	10
Query Workload	100
Query Time	random
Nodes	500
R	1



The more overlap, the best performance for the two-phase greedy selection

Conclusions

We deal with the problem of supporting historical queries on evolving graphs

- Avoid full snapshot construction for targeted queries. Instead, use partial views defined as egonets
- Define view subsumption and view extension
- Address the view selection problem
- Introduce a two-phase greedy selection algorithm

Thank you!

Questions?