Efficient Densest Subgraph Computation in Evolving Graphs

Alessandro Epasto

Joint work with Silvio Lattanzi (Google Research, NY) and Mauro Sozio (Télécom ParisTech)
Social Networks are Constantly Evolving

RECENT ACTIVITY

Julius Caesar is now friends with Brutus. Like · Comment

Brutus

Julius
Social Networks are Constantly Evolving

Julius Caesar is in a relationship with Cleopatra.

2000 Years Ago Like · Comment

Mark Anthony likes this.

Add a comment....
Social Networks are Constantly Evolving

**RECENT ACTIVITY**

Brutus attended Caesar's Betrayal in Senate.

- Brutus
- Julius
- Cleopatra
Social Networks are Constantly Evolving

Julius Caesar
You too @Brutus???

2000 Years Ago Like · Comment

Mark Anthony likes this.

Brutus  Julius  Cleopatra
Social Networks are Constantly Evolving

Brutus

Cleopatra
Social Networks are Constantly Evolving

Cleopatra is in a relationship with Mark Anthony.

2000 Years Ago Like · Comment

Mark Anthony likes this.

Add a comment....
Events in Social Media Streams

- **WWW2015** conference will be held in Florence.
- **Hofmann** confirmed keynote at **WWW2015** in Florence.
- **WWW2015** opens **May 20** in Florence.

Dense subgraphs represent events!
Event Detection

Try it now! Search any hashtag
Try these: #5sos #winterjam #gamergate #brooklynmarathon

Reveal what you missed
We summarize the roar of the crowd and surface the best photos and videos from any event or topic. We automatically find trending people and highlights using deep intelligence to tell the story.

View More Seens
Dynamic Community Detection Algorithms

Most algorithms assume a single static graph in input.

Naive solution: run the algorithm once for each update.

GOAL: efficiently keep track of the communities as the graph evolve.
Densest Subgraph

Definition (Densest Subgraph Problem)

Let $G = (V_G, E_G)$ be an undirected graph. Find a subgraph $H = (V_H, E_H)$ of $G$ with maximum average degree density:

$$\rho(H) = \frac{|E_H|}{|V_H|}.$$
Densest Subgraph

Definition (Densest Subgraph Problem)

Let $G = (V_G, E_G)$ be an undirected graph. Find a subgraph $H = (V_H, E_H)$ of $G$ with maximum \textbf{average degree density}:

$$\rho(H) = \frac{|E_H|}{|V_H|}$$
Densest Subgraph in **Static** Graphs

- **Community** used in Social Networks, Web and Biology.
- **Polynomial** exact algorithm (Goldberg, 1984)
- (2+\(\epsilon\))-approximation MapReduce algorithm (Bahmani et al., 2012).
Densest Subgraph in Dynamic Graphs

No results known* in dynamic graphs with sublinear update time *(before our publication)*.

Naive Approach: $O(m + n)$ time per update!

Our Problem

**Goal:** Preserve a $2+\epsilon$ approximation with average time $O(\text{poly-log}(n+m))$ per update.

**Notice:** Much better than $O(n+m)$ per update and includes output time!
Our Dynamic Graph Model

Start from an **empty graph**.

Arbitrary long **sequence of edge** updates arrives…

This models also **node addition/removals** implicitly.
Incremental and Fully-Dynamic

**INCREMENTAL:** arbitrary stream of edges additions only.

- (A, B)
- (B, C)
Incremental and Fully-Dynamic

**FULLY-DYNAMIC:** stream of edges arbitrary additions and random deletion.

(A, B)  
(B, C)  
(A, B)
Our Goal

Design a Data Structure:

1) AddEdge(u,v)
2) RemoveEdge(u,v)

Both operations can output a new densest subgraph S or nothing.

Invariant: the last subgraph in output is a $2+\varepsilon$ approx. for the current graph
Result for edge additions (incremental)

**Theorem:** We maintain a $2+\varepsilon$ approx. in $O(\log^2(n) / \varepsilon^2)$ average time and linear space.

Significant improvement over naive approach: $O(m+n)$ average time.
Result for edge additions and deletion  
(fully dynamic)

Theorem: We maintain a $2+\epsilon$ approx. in $O(\log^4(n) / \epsilon^4)$ average time and linear space.

Very fast also in practice!
Roadmap

• Review Bahmani et al. for static graphs.
• A new static graph algorithm.
• Incremental algorithm.
• Randomized fully-dynamic algorithm.
Static Case - Bahmani et al. Algorithm

Let \( \epsilon > 0 \):

Iteration: 1

1) Compute Avg. Deg = \( K \)
Let $\epsilon > 0$:

**Iteration: 1**

1) Compute $\text{Avg. Deg} = K$

2) Let $T = K (1 + \epsilon)$
Let $\epsilon > 0$:

**Iteration: 1**

1) Compute Avg. Deg = $K$

2) Let $T = K(1+\epsilon)$

3) Remove nodes with degree $< T$
Let $\epsilon > 0$:

Iteration: 2

1) Compute Avg. Deg $= K$

$T = 2.3$
Static Case - Bahmani et al. Algorithm

Let $\epsilon > 0$:

**Iteration: 2**

1) Compute $\text{Avg. Deg} = K$

2) Let $T = K(1 + \epsilon)$
Static Case - Bahmani et al. Algorithm

Let \( \varepsilon > 0 \):

**Iteration:** 2

1) Compute Avg. Deg = \( K \)

2) Let \( T = K (1+\varepsilon) \)

3) Remove nodes with degree < \( T \)
Static Case - Bahmani et al. Algorithm

Iterate until all nodes are removed.

Output the densest subgraph $G_i$. 
Static Case - Bahmani et al. Algorithm

Iterate until all nodes are removed.

Output the densest subgraph $G_i$.

Theorem: (Bahmani et al.)
$2 + \epsilon$ approx. in $\log(n)$ steps.
Towards a Dynamic Algorithm

- **Idea:** Store graphs $G_i$’s.
- When an edge is **added** update the $G_i$’s

This ensures a $2+\text{eps}$ approximation!
Towards a Dynamic Algorithm

- **Idea:** Store graphs $G_i$’s.
- When an edge is **added** update the $G_i$’s
Towards a Dynamic Algorithm

- **Idea**: Store graphs $G_i$’s.
- When an edge is **added** update the $G_i$’s.
Idea: fix Threshold $T$ for all iterations

- Use same threshold $T$ at each iteration.
- **Easier to analyze** and **maintain**.

For correct threshold $T$: same approximation of Bahamani et al.’s algorithm.

You’d better use $T = 3.1$
Moving Threshold (Only Additions)

1) Set $T = 1$ to compute densest subgraph $H$ and output it.

This provides a $2+\text{eps}$ approx. in $O(\text{poly-log}(n))$ average time.
Moving Threshold (Only Additions)

1) Set $T = 1$ to compute densest subgraph $H$ and output it.

2) Maintain the $G_i'$ using threshold $T$ as long as all nodes are removed in $O(\log(n))$ steps.

This provides a $2+\text{eps}$ approx. in $O(\text{poly-log}(n))$ average time.
Moving Threshold (Only Additions)

1) Set $T = 1$ to compute densest subgraph $H$ and output it.

2) Maintain the $G_i'$ using threshold $T$ as long as all nodes are removed in $O(\log(n))$ steps.

3) Repeat from 1) with higher threshold $T = T \times 2$

This provides a $2+\epsilon$ approx. in $O(\text{poly-log}(n))$ average time.
Fully-Dynamic Case

The analysis is significantly harder:

- The density can *increase/decrease* in complex patterns...
- ...densest subgraph is *stable* under *random removals*.
- We tackle the *stability* to recompute the subgraph few times.
Experimental Evaluation - Datasets

- **DBLP & Patent**: co-authorship graph.
- **LastFM**: songs co-listened.
- **Yahoo! Answers**: >1 Billions edges. Edge if two users answer the same question.
Evolution Densest Subgraph

- Density
- Size

DBLP - Sliding Window 5 years
Evolution Densest Subgraph

Patent Citations - Sliding Window 5 years
Evolution Densest Subgraph

Efficient in Highly Dynamic Datasets with Billions of Updates.

Yahoo Answers - Sliding Window 100M edges
Update Time vs Epsilon

Avg. Time per Update vs Epsilon

Scales much better with Epsilon than worst case.
Comparison with Static Algorithm

Avg. Time per Update vs K

Microseconds

Data sets: dblp, patent-coal, patent-cit, lastfm

Our Algorithm
K=100000
K=10000
K=1000
Comparison With Static Algorithm

Max Relative Error Static Algorithm vs K

- Relative Error

- dblp
- patent-cit
- patent-coal
- lastfm

- 100000
- 10000
- 1000
Conclusions and Future Work

• It is possible to maintain the densest subgraph efficiently in dynamic graphs.

• **Future work:** Recent Techniques (Bhattacharya et al.) to define 2+eps with adversarial removes?

• Top-k Densest Subgraph in Dynamic Graphs.
Thank you for your attention
Recent Results - STOC

Concurrently to our work Bhattacharya et al., STOC 2015 introduced a novel streaming algorithm for densest subgraph with strong guarantees.

- Different model: Update vs Query time.
- Strong space constraints (cannot store entire graph).
- Adversarial additions and deletions.

- $4+\varepsilon$ approx with $O(n \ poly \ log)$ space, $O(poly \ log)$ update time, $O(n)$ query time.
- $2+\varepsilon$ approx with $O(n \ poly \ log)$ space, higher time complexity.
Incremental Case: Only Additions

Lemma

*During each round we can maintain the invariant with total cost $O(m \log(n) \varepsilon^{-1})$.*

Lemma

*For any sequence of $m$ edges additions, there are at most $O(\log(n) \varepsilon^{-1})$ rounds in total.*
Max density is stable with different epsilons.
Analysis of the Algorithm

We divide the edge additions in Rounds.

Round 1
- Add
- Add
- Add

Run of Static Algorithm
- H
- Overflow

T <- T(1+eps)

output

Round 2
- Add
- Add

Run of Static Algorithm
- H
- Overflow

output T <- T(1+eps)

Round i
- Add
- Add

Run of Static Algorithm
- H
- Overflow

output
Densest Subgraph - LP Primal

There is a \([0, 1]\) variable \(y_i\) for each node \(v_i \in V\), while there is a \([0, 1]\) variable \(x_{ij}\) for each edge \(e_{ij} \in E\).

\[
\text{max} \quad \sum_{ij \in E} x_{ij} \\
\text{s.t.} \quad x_{ij} \leq y_i \\
\quad \quad \quad \quad x_{ij} \leq y_j \\
\quad \quad \quad \quad \sum_{i \in V} y_i = 1, \\
\quad \quad \quad \quad x_{ij}, y_i \in [0, 1] \\
\quad \quad \quad \quad \forall e_{ij} \in E, \forall v_i \in V.
\]
Definitions

We say that an algorithm is a \textbf{approximation} of the densest subgraph problem for $a > 1$ if it outputs a graph with density at least:

$$\frac{\text{OPT}}{a}$$

We say that an operation has \textbf{T amortized time} if for any sequence of $k$ update operations the total time is

$$O(kT)$$
Densest Subgraph - LP Primal Dual

• The dual problem is the well-known graph orientation problem.
• Given undirected graph $G$ find directed graph $H$ obtained orienting the edges of $G$ arbitrarily, that minimizes the maximum in-degree.
• If $G$ has orientation of max in-degree $< D$ then density of densest subgraph is $< D$.
• Hence, if it is possible to remove all nodes by recursively removing nodes with degree $< D$ then max density is $< D$. 
We divide the edge additions and deletions in **Rounds**.
Fully Dynamic Algorithm

We divide the edge additions and deletions in Rounds.

Run Static Algorithm

Round i

Add  Rem  Add

Invariant Fails

H

Bad Round < $O(m / \log(n))$ removals

Good Round > $O(m / \log(n))$ removals
Idea: in good rounds removals “pay” for all the operations

We can show that there are never more than poly-log consecutive bad rounds (w.h.p)