Dense Subgraph Maintenance under Streaming Edge Weight Updates for Real-time Story Identification

Albert Angel
University of Toronto
albert@cs.toronto.edu

Nick Koudas
University of Toronto
koudas@cs.toronto.edu

Nikos Sarkas
University of Toronto
nsarkas@cs.toronto.edu

Divesh Srivastava
AT&T Labs-Research
divesh@research.att.com

Abstract

Recent years have witnessed an unprecedented proliferation of social media.

The sheer scale, and rapid evolution of the data involved necessitate highly efficient techniques for identifying important stories at every point of time.

Abstract

The main challenge in real-time story identification is the maintenance of dense subgraphs under streaming edge weight updates.

This is the first work to study the efficient maintenance of dense subgraphs under such streaming edge weight updates.

Introduction

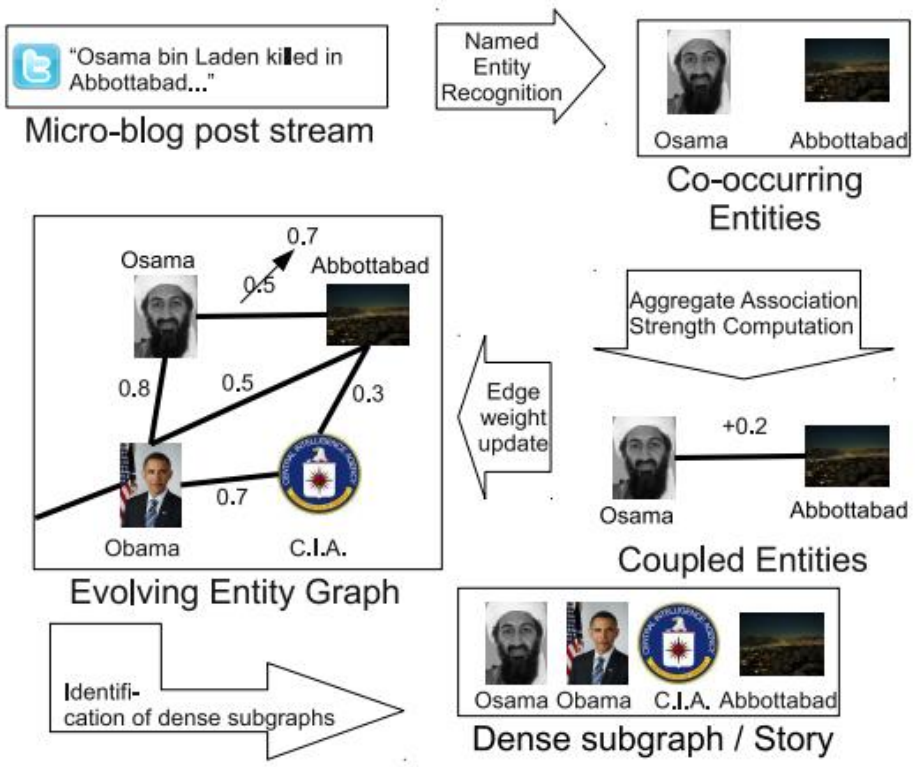


Figure 1: Real-time identification of "bin Laden raid" story, and connection to ENGAGEMENT

- In all cases, stories have a strong temporal component, making timeliness a prime concern in their identification.
- Interestingly, such stories can be identified by leveraging the real-world entities involved in them.

Introduction

- This work thus addresses the problem of dENse subGrAph maintenance for edGE-weight update streaMs under sizE constraiNTs, or ENGAGEMENT for brevity.
- Challenges:
- 1. A change in the weight of a single edge, can impact the density of many subgraphs, necessitating a potentially unbounded exploration of the entity graph.
- 2. Moreover, there does not exist a single definition of graph density suitable for all scenarios.

Introduction

- Contributions:
- 1. Motivated by the need to identify emerging stories in realtime, for a wide range of measures of entity association, we formalize the problem of dENse subGrAph maintenance for edGEweight update streaMs under sizE constraiNTs.
- 2. We propose an efficient algorithm DYNDENS, based on a novel quantification of the maximum possible change caused by a single edge weight update.
- 3. We design an efficient dense subgraph index.
- 4. We validate our techniques via a thorough experimental evaluation on both real and synthetic datasets.

FORMALIZATION

- At every discrete time interval, the weights of one or more edges are adjusted.
- The goal is to maintain, at each point of time, all subgraphs with “density” greater than a given threshold.
- In this context, vertices correspond to real-world entities, and edge weights to their (current) pairwise association strengths.

FORMALIZATION

Data Model:

- G= (V, E) with N vertices, w_{ij} is the weight of edges between i and j.
- $update_i = (a, b, \delta)$, signifying that at time instant i, the weight of the edge between vertices a and b changed from $w_{ab}\ to\ w_{ab} + \delta$.

FORMALIZATION

Density:

- $dens(C) = \frac{score(C)}{S_{|C|}}, where\ score(C) = \sum_{i,j \in C \wedge i<j}(w_{ij}), C\ is\ a\ subgraph\ and\ C \subseteq V$
- S_n is a function quantifying the relative importance of a subgraph's cardinality, n, to its density.
- $S_n = \frac{n.(n-1)}{2}$ is defined as the average edge weight.
- $S_n = n$ is defined as the average node degree.

$$\frac{n}{n-1} \leq \frac{S_n}{S_{n-1}} \leq \frac{n}{n-2}$$

FORMALIZATION

Cardinality constraint:

- N_{max} be a maximum cardinality for subgraphs of interest.
- ENGAGEMENT: at every point i $\{V_j | V_j \subseteq V \wedge dens(V_j) \geq T \wedge |V_j| \leq N_{max}\}$ T is a given threshold. We term these output-dense subgraphs.

FORMALIZATION

Notation:

- $V = \{1, \dots, N\}$ denotes the set of vertices in G.
- $\widehat{e_i}$ be the i'th basis vector.
- Denote $C \subseteq V$ *by its corresponding vector* $\underset{c}{\rightarrow} = \sum_{i \in C} \widehat{e_i}$
- $\underset{\mathcal{T}_u}{\rightarrow} = (w_{1u}, w_{2u}, \dots w_{\mathrm{N}u})$ be the neighborhood vector of vertex u.
- $g_n = \frac{S_n}{n.(n-1)}$ By the monotonicity properties of Sn, it follows that $g_n \leq g_{n-1}$.
- We will denote a quantity X before the update as X^- and after the update as X^+.

Dense subgraphs and growth property

Dense subgraphs and growth property:

- C is a dense subgraph iff it has density greater than a given threshold $T_{|C|}$, and cardinality of at most N_{max}.
- *Tn* is defined in a manner that ensures that every dense graph with *n* vertices has at least one dense subgraph with *n*−1 vertices.
- $T_n \cdot g_n > T_{n-1} \cdot g_{n-1}$ and T_n is a monotonically increasing function.
- We requires $T_{N_{max}} = T$.

Dense subgraphs and growth property

Table 1: Definitions of density-related properties

Subgraph C is $\cdots$	iff
Static properties	
dense	$dens(C) \geq T_{\lvert C\rvert}$
sparse	$dens(C) < T_{\lvert C\rvert}$
output-dense	$dens(C) \geq T$
too-dense	$dens(C) \geq T_{\lvert C\rvert+1}$
Dynamic properties	
stable-dense	$dens(C)^{-} \geq T_{\lvert C\rvert} \wedge dens(C)^{+} \geq T_{\lvert C\rvert}$
newly-dense	$dens(C)^{-} < T_{\lvert C\rvert} \wedge dens(C)^{+} \geq T_{\lvert C\rvert}$
losing-dense	$dens(C)^{-} \geq T_{\lvert C\rvert} \wedge dens(C)^{+} < T_{\lvert C\rvert}$

Dense subgraphs and growth property

Table 2: Summary of main symbols used

Symbol	Description
V	Set of vertices in graph
N	Number of vertices in graph
w_{ij}	Weight of edge between vertices i and j
$\vec{\Gamma}_u$	Neighborhood vector of vertex u
$dens(C)$	Density of C $dens(C) = \frac{\sum_{i,j \in C \wedge i<j}(w_{ij})}{S_{\lvert C \rvert}}$
S_n	Quantifies relative importance of subgraph cardinality n to density
g_n	Normalized version of S_n: $g_n = \frac{S_n}{n \cdot (n-1)}$
AVGWEIGHT	Case where $S_n = n(n-1)/2$
SQRTDENS	Case where $S_n = \sqrt{n}(n-1)$
AVGDEGREE	Case where $S_n = n$

N_{max}	Max. cardinality of subgraph to be returned
T	Min. density for a subgraph to be returned
T_n	Min. density for subgraph of cardinality n to be dense
δ_{it}	Tunable parameter of DYNDENS, influences T_n
a, b	Vertices that were just updated
x^-	quantity x before the update
x^+	quantity x after the update
w	Weight of edge (a, b) before the update, ie. w_{ab}^-
$w + \delta$	Weight of edge (a, b) after the update, ie. w_{ab}^+

Dense subgraphs and growth property

Edge weight updates:

- Handling updates with $\delta < 0$, all dense subgraph containing both a and b are examined, and their density is decreased by an appropriate amount.
- If they are no longer output-dense, this is reported.
- If, in addition, they are no longer dense (losing-dense), they are evicted from the index.

Dense subgraphs and growth property

Positive updates:

- Handling updates with $\delta > 0$, the edge weight update corresponds to an increase in weight.
- **Cheap explore:** try to augment all dense subgraphs containing either a or b, with b or a, respectively; resulting newly-dense subgraphs will be inserted into the dense subgraph index.
- **Explore**: try to augment dense subgraphs containing both a and b, with one neighboring vertex; resulting newly dense subgraphs will be inserted into the dense subgraph index.

Dense subgraphs and growth property

Positive updates:

- **Exploration iterations:** The above procedure may need to be performed iteratively for newly-dense subgraphs discovered via exploration or cheap exploration. At most $\left\lceil \frac{\delta}{\delta_{it}} \right\rceil$ iteration exploration iterations.
- **Explore all**: In a few cases, the above exploration may need to be performed on non-neighboring nodes as well, resulting in a very costly procedure.

The DYNDENS Algorithm

Algorithm 1 Algorithm DYNDENS

Input: Updated edge (a, b), magnitude of update δ

1: **if** $\delta < 0$ **then**
2: Update the density of all dense subgraphs containing a and b; evict losing-dense subgraphs from the index; report any subgraphs that are no longer output-dense
3: **return**
4: **for all** dense subgraphs C st. $a \in C \vee b \in C$ **do** {// including $C = \{a, b\}$ if it is newly-dense}
5: **if** $a \notin C$ **or** $b \notin C$ **then**
6: **if** C should be cheap-explored and $C \cup \{a, b\}$ is newly-dense **then**
7: Add $C \cup \{a, b\}$ to the index, report it if it is output-dense
8: **explore**$(C \cup \{a, b\}, 2)$
9: **else**
10: Update the density of C, report it if it just became output-dense
11: **explore**$(C, 1)$

- **Exploration iterations:** If the edge weight update was negative, only some index maintenance needs to be done (line 2).
- Otherwise, some stable-dense subgraphs containing a and/or b are further examined (line 4-11).

The DYNDENS Algorithm

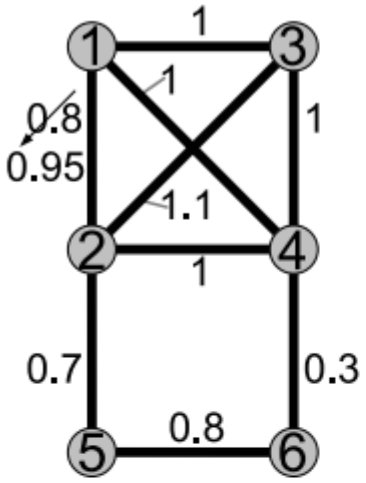
Algorithm 2 Procedure **explore**(C, i)

Input: Subgraph C. Iteration number i

1: **if** C was not too-dense before the update and $i \leq \lceil \frac{\delta}{\delta_{it}} \rceil$ and $|C| < N_{max}$ **then**
2: **if** C is too-dense **then**
3: **for all** $y \notin C$ **do** {// Explore-All}
4: Add $C \cup \{y\}$ to the index; report it if it is output-dense
5: **explore**$(C \cup \{y\}, i+1)$
6: **else**
7: **for all** neighbors y of C **do**
8: **if** $C \cup \{y\}$ is newly-dense **then**
9: Add $C \cup \{y\}$ to the index; report it if it is output-dense
10: **explore**$(C \cup \{y\}, i+1)$

- Algorithm 2 will first ensure that the subgraph should be explored.
- Moreover, as previously mentioned, DYNDENS will not explore around any subgraph more times than necessary.
- Finally, in a few cases, explored subgraphs will need to be augmented with every other vertex, not just neighboring ones.

The DYNDENS Algorithm



(a) Entity graph

Subgraph	Density	output-dense?
Dense, before update		
1, 3	1.0	Y
1, 4	1.0	Y
2, 3	1.1	Y
2, 4	1.0	Y
3, 4	1.0	Y
1, 3, 4	1.0	Y
2, 3, 4	$1.0\overline{3}$	Y
newly-dense, after update		
1, 2	0.95	N
1, 2, 3	$1.01\overline{6}$	Y
1, 2, 4	$0.98\overline{3}$	N
1, 2, 3, 4	$1.008\overline{3}$	Y

(b) Dense subgraph index

Figure 2: Execution example

- A density threshold of T =1, $N_{max} = 4$.
- $\delta_{it} = 0.15, T_2 = 0.9, T_3 = 0.975, T_4 = T = 1$
- Finally, assume the weight of edge(1,2) is updated from 0.8 to 0.95.

Index

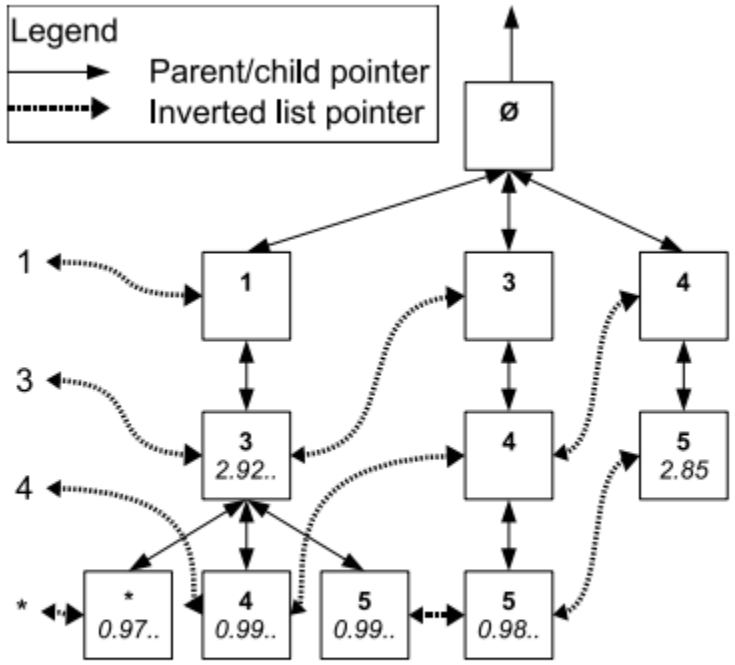


Figure 3: Dense subgraph index

- Each subgraph has a unique id; it is also represented by its (sorted) set of vertices.
- DYNDENS will maintain a prefix tree of dense subgraphs.
- Each no in the prefix tree contains pointers to its children, indexed by vertex id, a pointer to its parent, as well as information
- Inverted lists: a mapping from vertices to all subgraphs containing a vertex, which is implemented as linked lists of prefix tree nodes.

Index

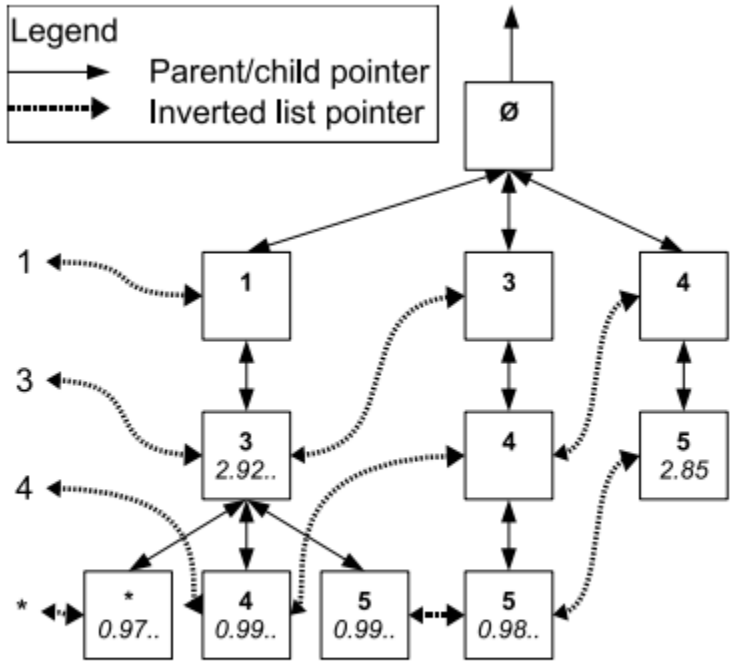


Figure 3: Dense subgraph index

- Looking up $C \cup \{u\}$ is O(|C|+1). O(1) if vertex u is lexicographically greater than any other vertex in C.
- Update of insertion into the index is O(1).
- Enumerating the vertices in a subgraph C is O(|C|).
- Deleting a subgraph C from the index is O(number of the leaf nodes deleted) between O(1) and O(|C|).

Avoiding redundant computation

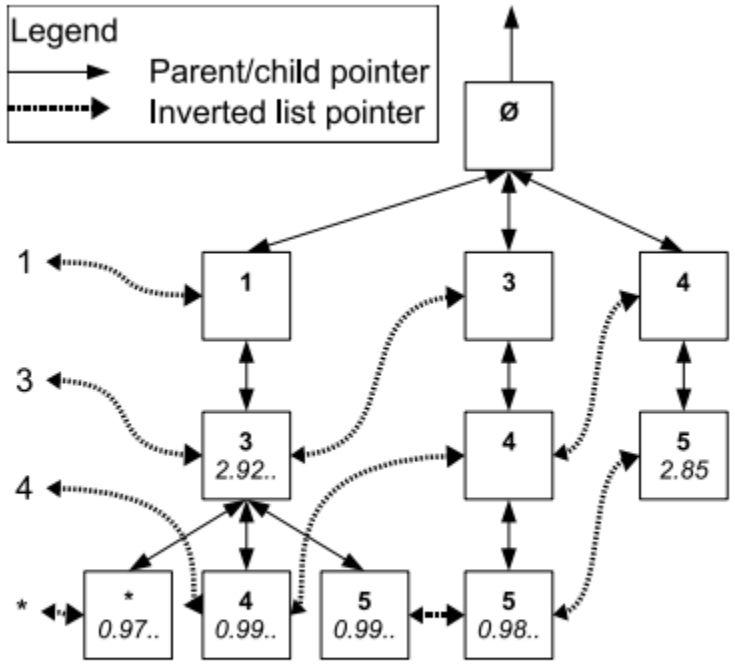


Figure 3: Dense subgraph index

- To ensure that subgraphs that were dense before the update are examined exactly once.
- To greatly reduce the number of newly-dense subgraphs examined more than once.

Implicit representation of too-dense subgraphs

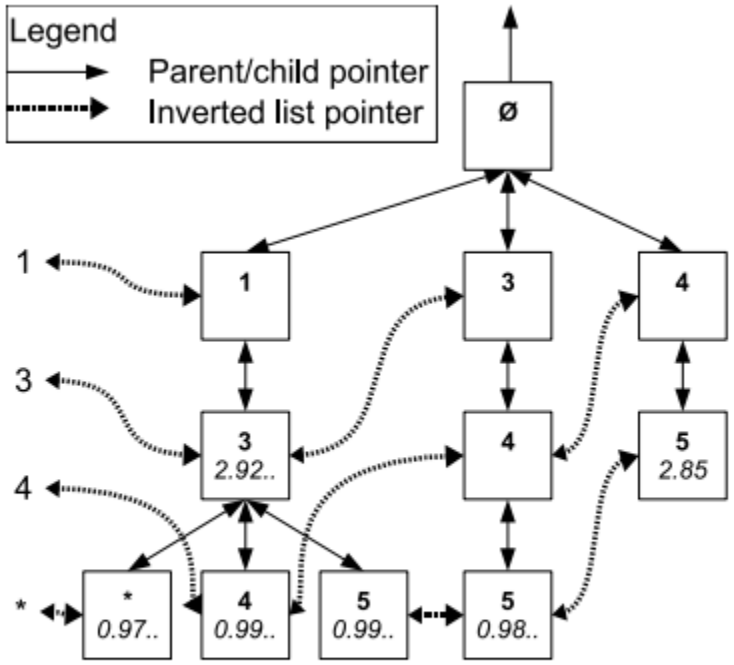


Figure 3: Dense subgraph index

- Too-dense subgraph need to consider its cartesian product with the entire set of vertices V, resulting in |V| dense subgraph insertions into the index.
- IMPLICTTOODENSE:
- (1) introduce a fictitious vertex named *, which is lexicographically larger than all other vertices.
- (2) For every too-dense subgraph C, the index will store a subgraph $C \cup \{y\}$ where y is a vertex disconnected from C.

When is a Single Exploration Sufficient?

- ***Sufficient condition*** : |C|=n>2 to contain a stable-dense subgraph of cardinality n-1.

- $\delta \leq (n-2)(n-1)(g_n.T_n - g_{n-1}.T_{n-1})$

Proof sketch: (pigeonhole argument) If all $n-1$ subgrap
of C were sparse before the update, then the contributions of ea
vertex in C to $dens^-(C)$ should be large. Hence, C must be ve
dense. However, C was sparse before the update. Thus, the upda
must have been very large. If the update is not very large, then the
will exist an $n-1$ subgraph that was dense before the update.

Corollary: The $n-1$ subgraph of C that was dense befo
the update will either not contain one of a or b (so augmenti
it with that vertex will yield C), or it will contain both a and
Consequently, for values of n where Equation 1 holds, all newl
dense subgraphs of cardinality n will be contained in $C_A \cup C_B$
$C_{AB} = C_1$.

Instantiating

$$T_n = \frac{1}{g_n}\left(g_{N_{max}} \cdot T + \delta_{it} \cdot \left(\frac{n-2}{n-1} - \frac{N_{max}-2}{N_{max}-1}\right)\right) \quad (2)$$

where δ_{it} is a tunable parameter. Note that this is a reasonable value for T_n from a maintenance perspective ; for instance, if $S_n = n$, then $T_n = (n-1)T_2 + (n-2)\delta_{it} = \frac{n-1}{N_{max}-1}(T + \delta_{it}) - \delta_{it} = O(n)$, while if $S_n = n(n-1)$, then $T_n = T_2 + (1 - \frac{1}{n-1})\delta_{it} = T - \delta_{it}(\frac{1}{n-1} - \frac{1}{N_{max}-1}) = O(1)$.

Bounding the Number of Iterations

- All newly-dense subgraphs of cardinality $\boldsymbol{C_0} \cup \boldsymbol{C_1} \cup \ldots \cup \boldsymbol{C}_{\left\lceil \frac{\delta}{\delta_{it}} \right\rceil}$

Proof sketch: An update of magnitude δ is equivalent to $\lceil \frac{\delta}{\delta_{it}} \rceil$ updates of magnitude up to δ_{it}; furthermore, re-exploring stable-dense subgraphs will not yield any new dense subgraphs, thus only newly-dense subgraphs will need to be explored subsequently.

Discussion: As witnessed from the above result, the magnitude of δ is directly correlated with the impact on dense subgraphs. A useful analogy is that of an edge weight update as a perturbation: the greater its magnitude δ, the further away in the graph its effects can be potentially felt (i.e. the further away dense subgraphs will need to be explored).

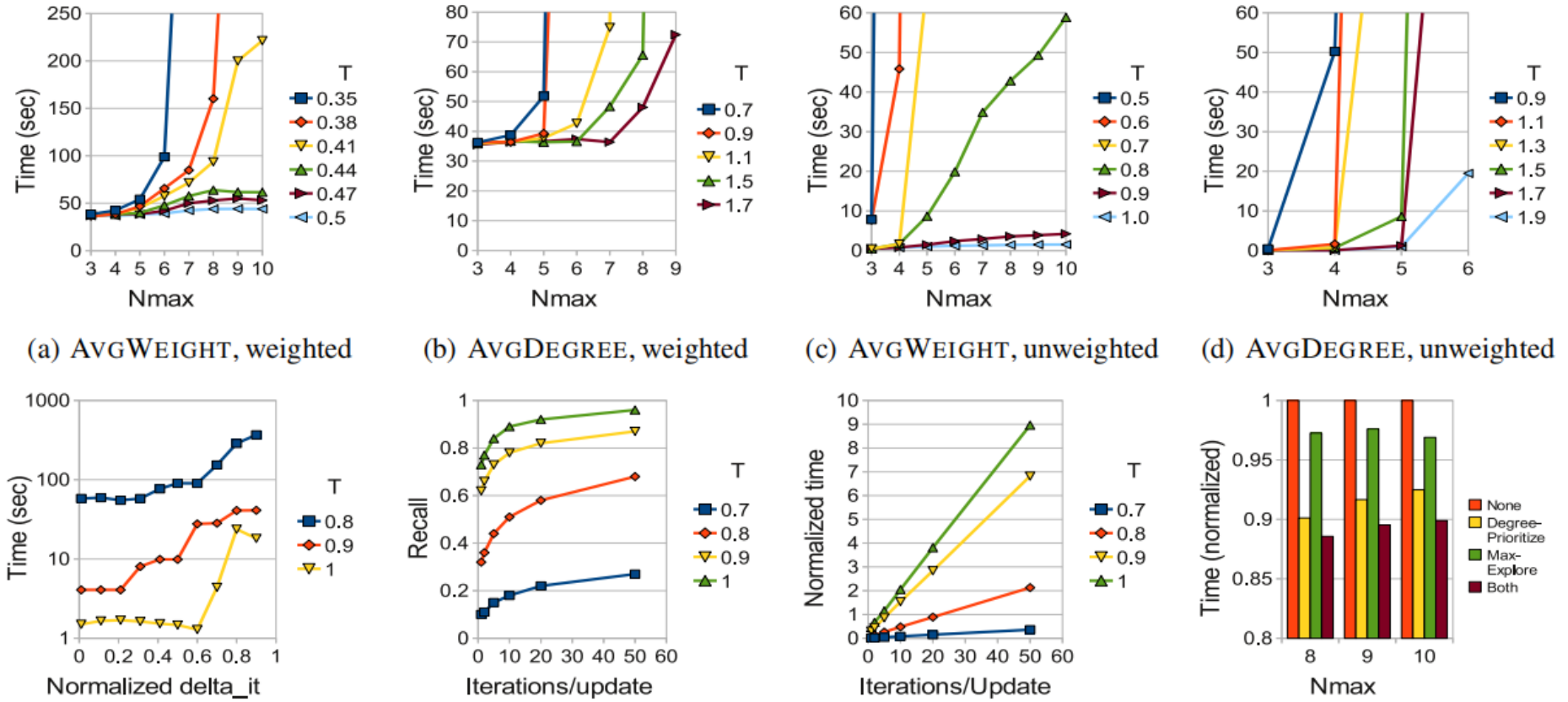
Bounding the Number of Iterations

- All newly-dense subgraphs of cardinality $\boldsymbol{C_0} \cup \boldsymbol{C_1} \cup \ldots \cup \boldsymbol{C}_{\left\lceil \frac{\delta}{\delta_{it}} \right\rceil}$

Proof sketch: An update of magnitude δ is equivalent to $\lceil \frac{\delta}{\delta_{it}} \rceil$ updates of magnitude up to δ_{it}; furthermore, re-exploring stable-dense subgraphs will not yield any new dense subgraphs, thus only newly-dense subgraphs will need to be explored subsequently.

Discussion: As witnessed from the above result, the magnitude of δ is directly correlated with the impact on dense subgraphs. A useful analogy is that of an edge weight update as a perturbation: the greater its magnitude δ, the further away in the graph its effects can be potentially felt (i.e. the further away dense subgraphs will need to be explored).

Evaluation



(a) AVGWEIGHT, weighted (b) AVGDEGREE, weighted (c) AVGWEIGHT, unweighted (d) AVGDEGREE, unweighted

(e) Effects of δ_{it}, unweighted (f) Recall of GRASP, unweighted (g) Performance of GRASP relative to DYNDENS, unweighted (h) Effect of heuristics, synthetic

Figure 4: Experimental evaluation

HEURISTICS

- ***MAXEXPLORE***: an improvement over the previous bound, that takes the graph neighborhood of the updated edge.
- ***DEGREEPRIORITIZE:*** a way to organize the search space, and thus often avoid performing redundant explorations.
- (1) introduce a fictitious vertex named *, which is lexicographically larger than all other vertices.
- (2) For every too-dense subgraph C, the index will store a subgraph $C \cup \{y\}$ where y is a vertex disconnected from C.

谢谢大家！