# Qualifying Exam: Doctor of Philosophy in Computer Science

Reet Barik

School of Electrical Engineering and Computer Science
Washington State University

14 August, 2020

# Contents

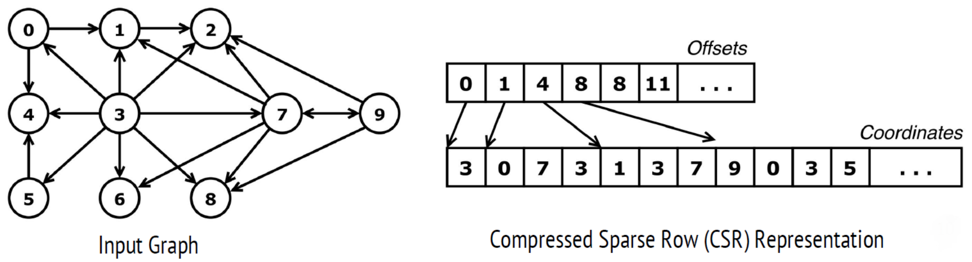
# MinHash for Vertex Reordering

# Overview: Vertex Reordering

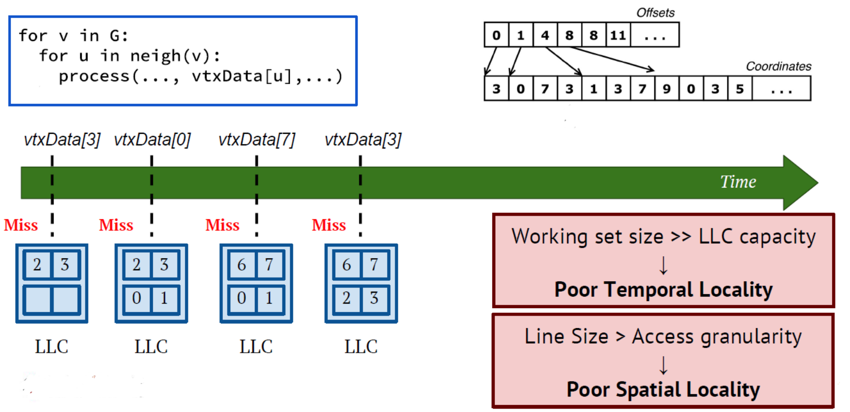
```
for v in G:
  for u in neigh(v):
    process(..., vtxData[u],...)
```

Typical graph processing kernel

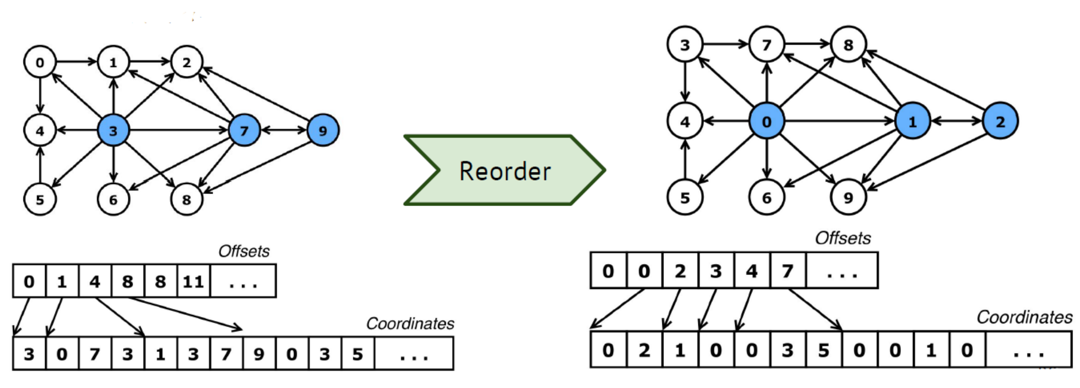
# Overview: Vertex Reordering



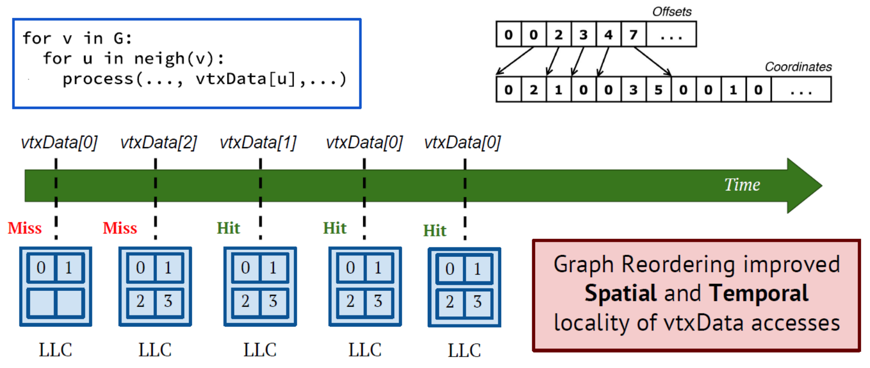
# Overview: Vertex Reordering



# Overview: Vertex Reordering

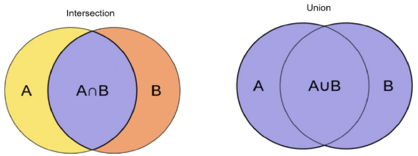


# Overview: Vertex Reordering



# Overview: MinHash

## Jaccard Coefficient




$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

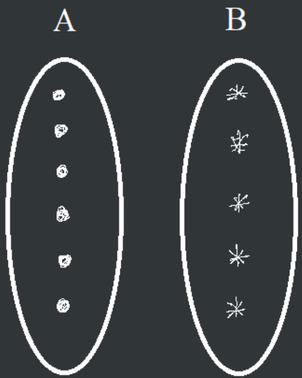
The main objective of the MinHash technique is to estimate the Jaccard coefficient (similarity) of two large sets in an inexpensive fashion.

# MinHash Algorithm: Variant 1

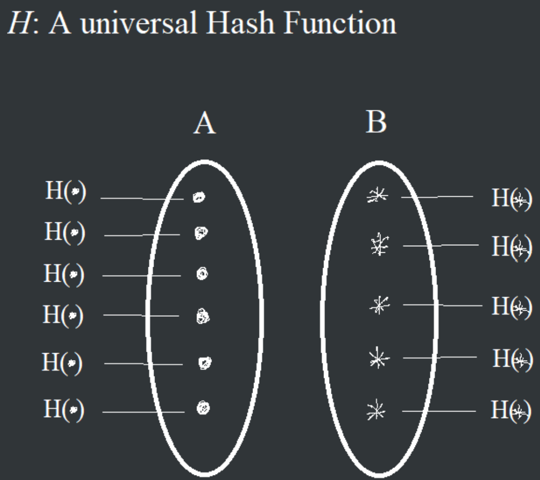
$H$: A universal Hash Function

# MinHash Algorithm: Variant 1

*H*: A universal Hash Function

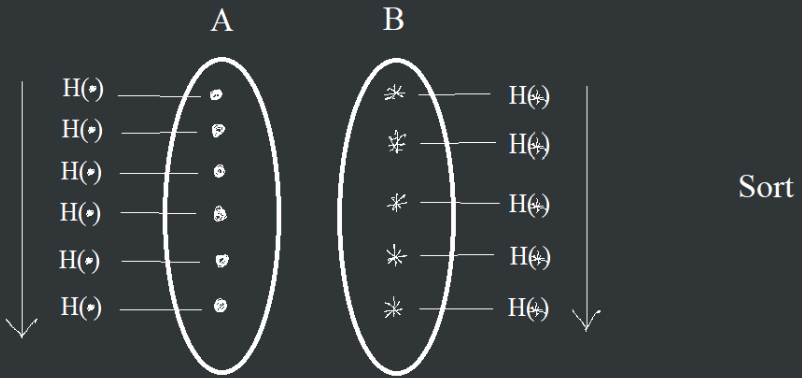


# MinHash Algorithm: Variant 1

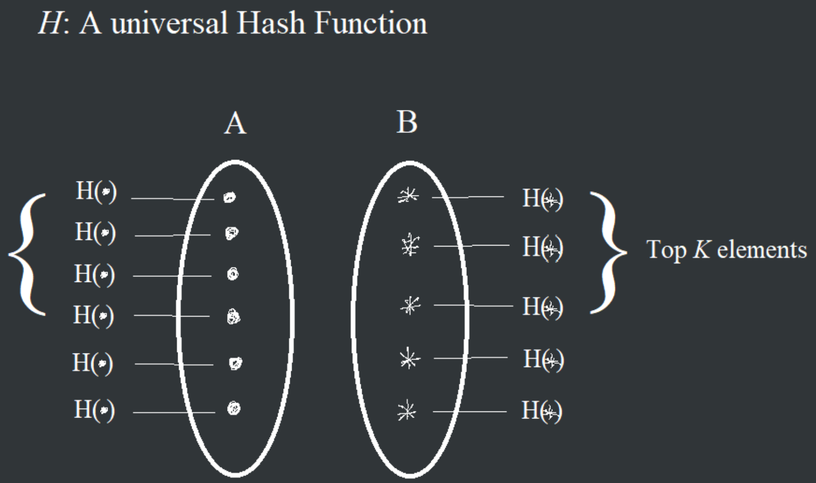


# MinHash Algorithm: Variant 1

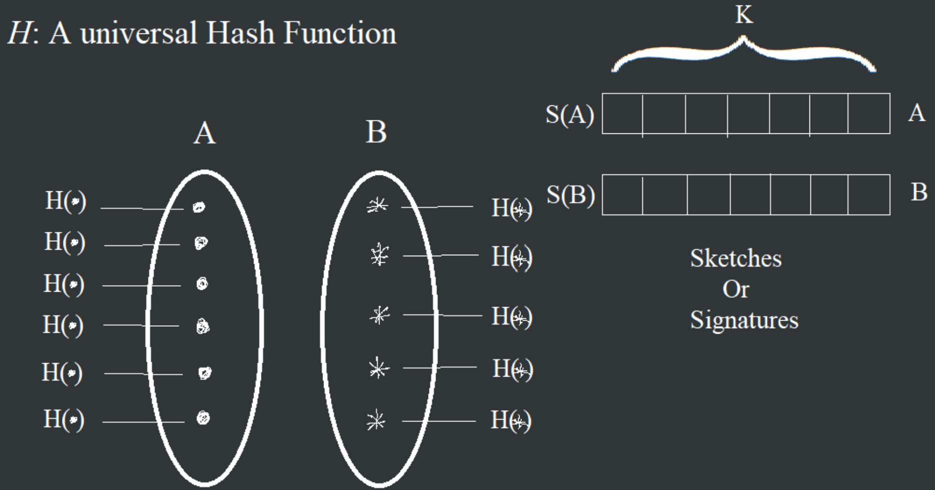
*H*: A universal Hash Function



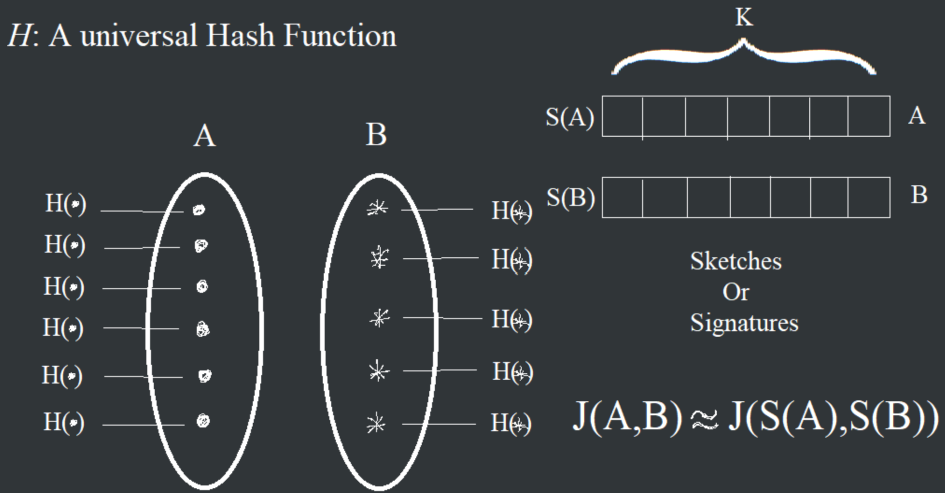
# MinHash Algorithm: Variant 1



# MinHash Algorithm: Variant 1



# MinHash Algorithm: Variant 1

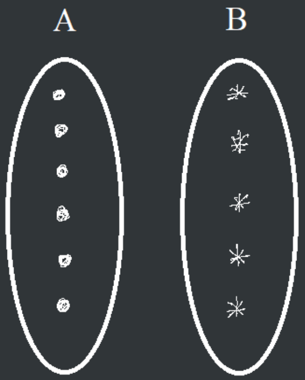


# MinHash Algorithm: Variant 2

$H$: A family of Universal Hash Functions

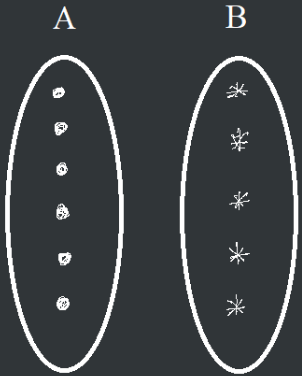
# MinHash Algorithm: Variant 2

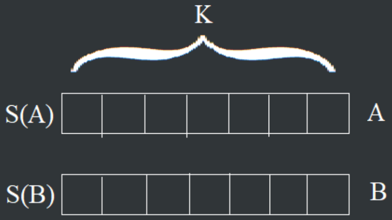
$H$: A family of Universal Hash Functions



# MinHash Algorithm: Variant 2

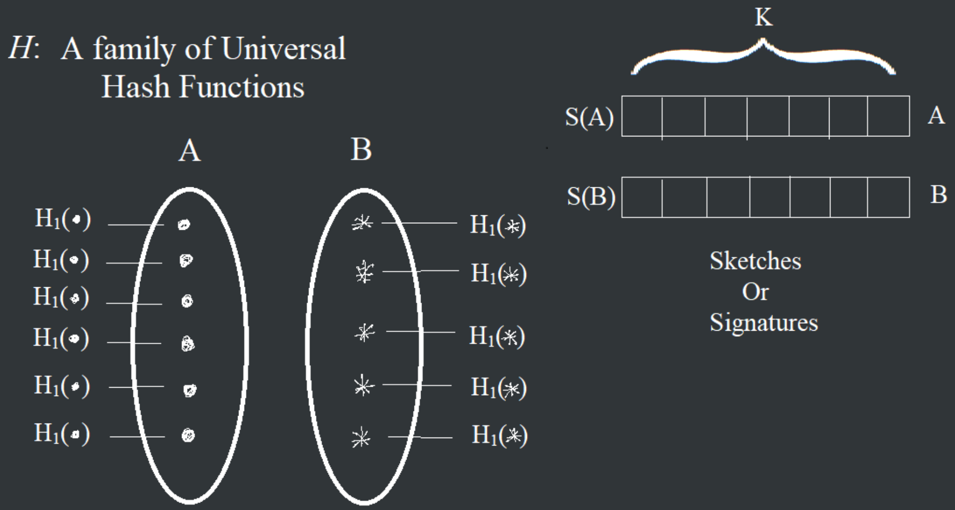
*H*: A family of Universal Hash Functions



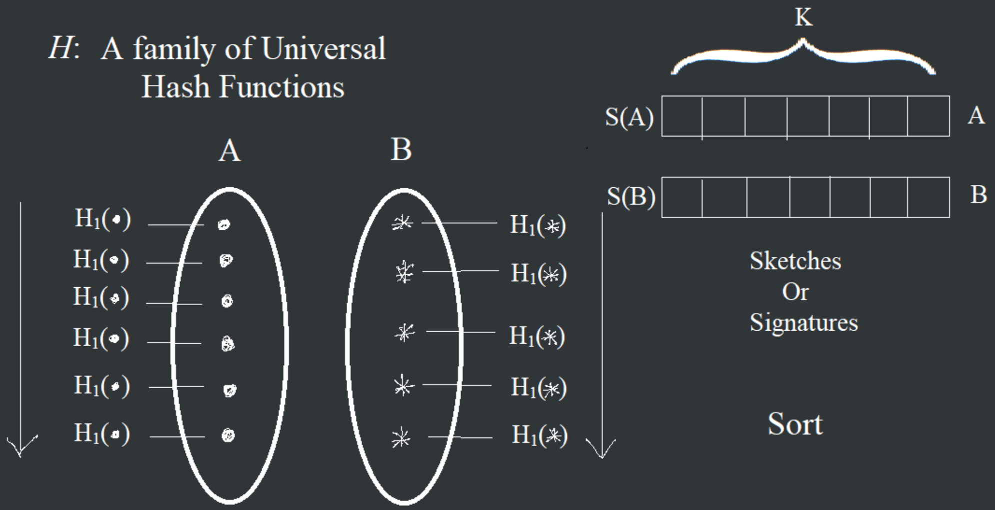




Sketches
Or
Signatures

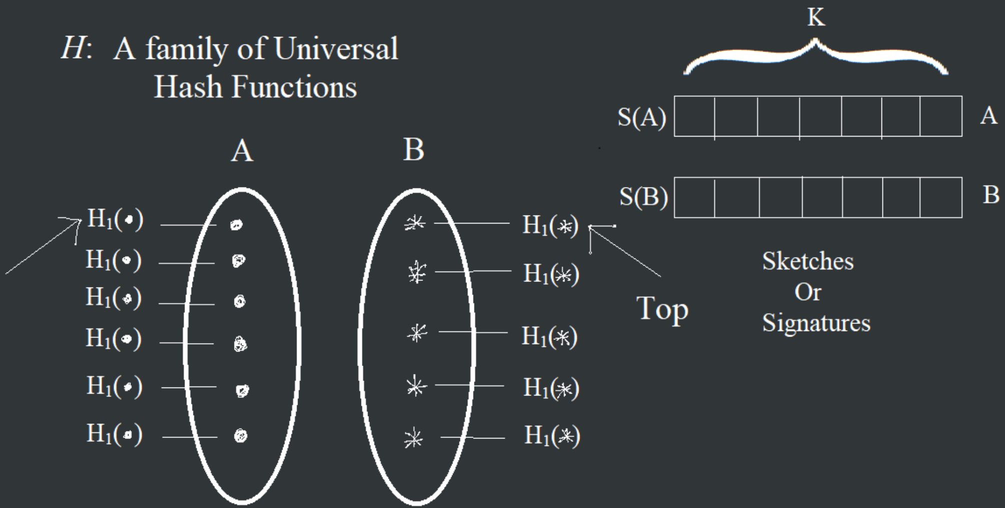
# MinHash Algorithm: Variant 2



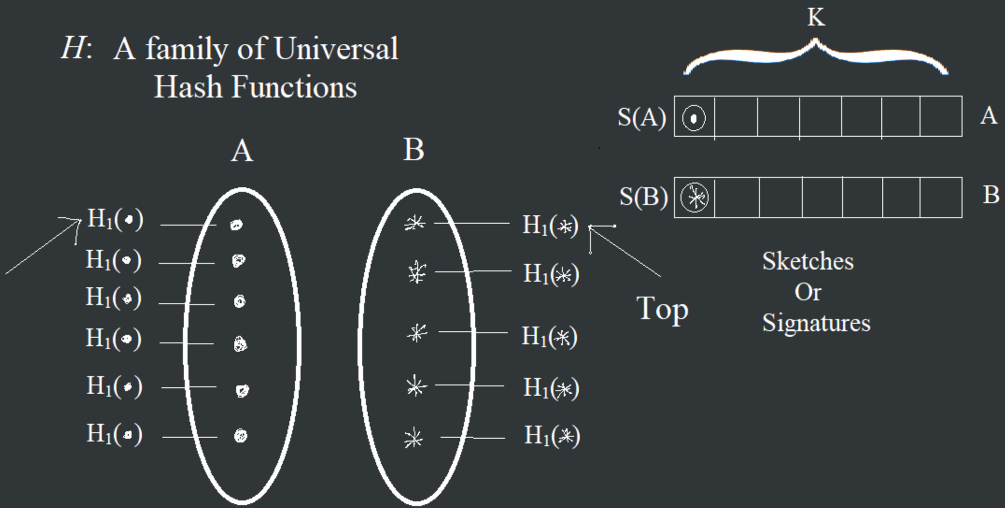
# MinHash Algorithm: Variant 2



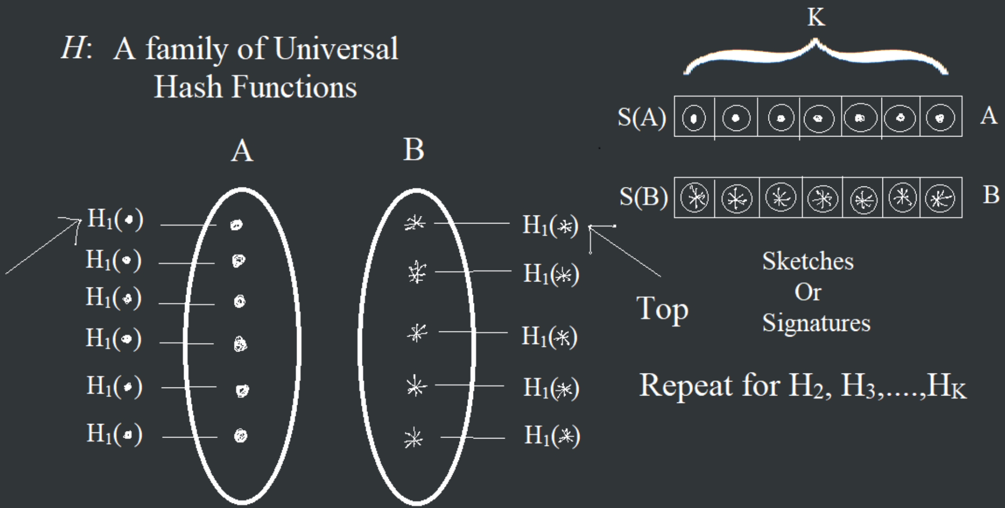
# MinHash Algorithm: Variant 2



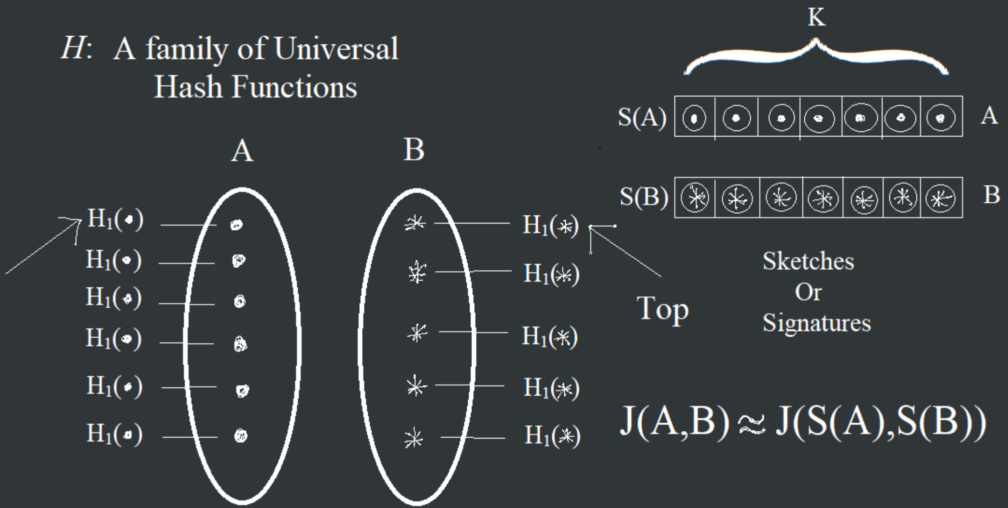
# MinHash Algorithm: Variant 2



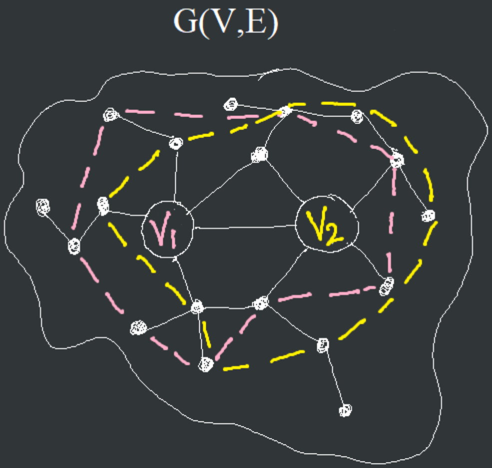
# MinHash Algorithm: Variant 2



# MinHash Algorithm: Variant 2



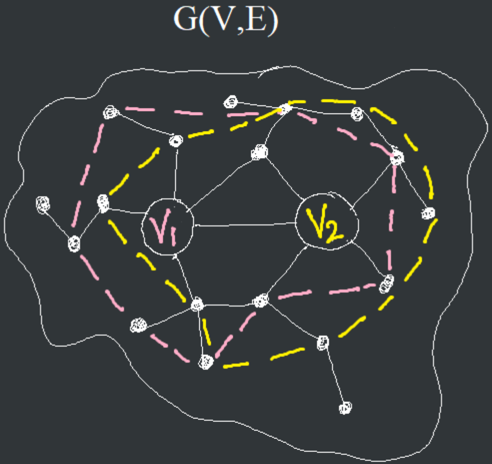
# Preserving graph neighborhoods via MinHash




In a graph $G(V, E)$ where $V$ is the set of vertices, and $E$ is the set of edges, $\forall v \in V$, do a BFS from $v$ going up to a depth $d$, where $d$ is a hyperparamter that can be tuned.

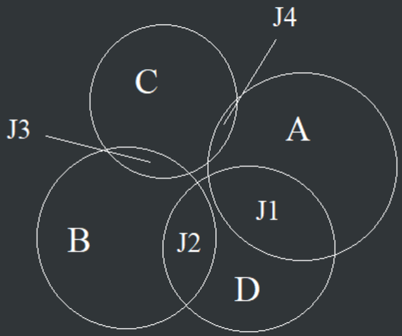
The neighborhood of a vertex $v$ is defined by the set $S(v)$ which contains the vertices visited by the above mentioned BFS traversal.

# Preserving graph neighborhoods via MinHash



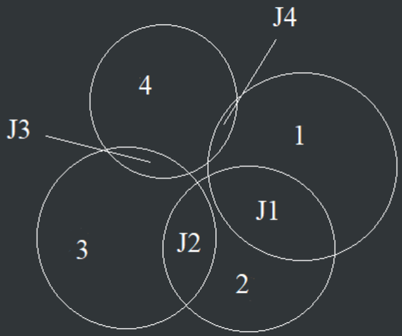

Use MinHash to estimate the similarity between these sets which implicitly gives an estimate of the similarity between the neighborhoods that these sets represent.

# Preserving graph neighborhoods via MinHash



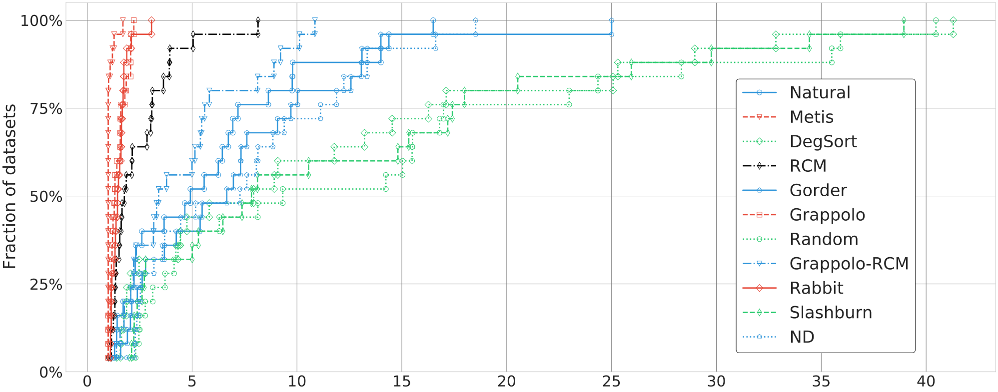

Greedily order these sets based on the estimated Jaccard coefficient of each pair.
Here, $J1 \approx J(A, D)$, $J2 \approx J(D, B)$, $J3 \approx J(C, B)$, and $J4 \approx J(A, C)$ such that, $J1 > J2 > J3 > J4$

# Preserving graph neighborhoods via MinHash



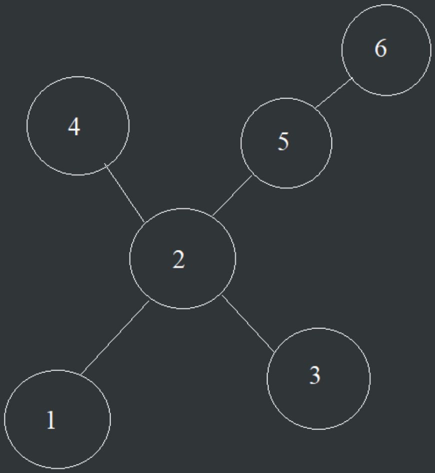

This leads to the ordering where $A$ is followed by $D$, $B$, and then $C$.
The ordering of nodes within each set can be done conserving the BFS traversal order. This further helps to preserve the smaller neighborhoods while reordering inside the bigger sets.

# Using MinHash instead of Community Detection



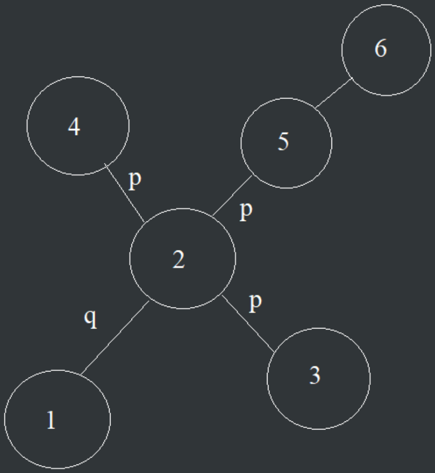

[Taken from a recent submission under review at IISWC 2020]

# Using MinHash instead of Community Detection




In a graph $G(V, E)$ where $V$ is the set of vertices, and $E$ is the set of edges, $\forall v \in V$, do a random walk of length $l$, where $l$ is a hyperparameter than can be tuned.

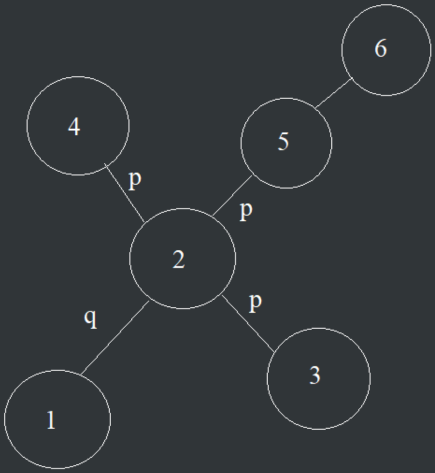
# Using MinHash instead of Community Detection




Like 'Node2vec' the random walks can be parameterized as follows:

- During a random walk when a transition is made from a vertex $u$ to $v$, the parameter $q$ could be used to generate the probability of making the transition back from $v$ to $u$.
- During a random walk, the parameter $p$ can be used to influence whether the transitions made one hop away will be more in a BFS, or a DFS fashion.

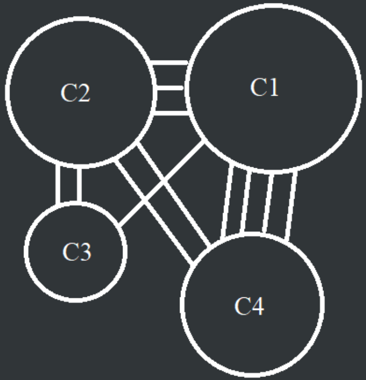
# Using MinHash instead of Community Detection




Each random walk gives a set of vertices (large enough to make MinHash not be an overkill). These sets can be put through the MinHash treatment so as to cluster the random walks together based on similarity.

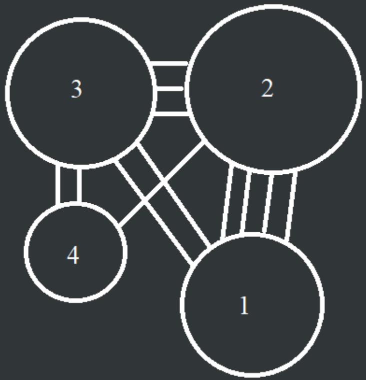
For pairs of sets representing random walks sharing an estimated Jaccard Coeffient higher than a set threshold, the intersection of such sets have a high probability of having vertices that might otherwise have belonged to the same community generated by a traditional community detection algorithms.

# Using MinHash instead of Community Detection




Ordering of communities could be formulated as a matching problem based on the strength of connections between communities.
Here, C1 and C4 are connected most strongly. Followed by the pair of (C2,C1). Next up are (C2,C3) and (C2,C4). Since, C4 is already matched up, the pair (C2,C3) is chosen.

# Using MinHash instead of Community Detection




This gives the order: C4,C1,C2,C3.
For intra-community vertex reordering, the natural order could be preserved, or each community could be used as an input graph to this whole process in a recursive fashion.

# RADAR (Reordering Assisted Duplication/Duplication Assisted Reordering): A Review

# Background

## Types of Graph Kernels

**Algorithm 1** Typical graph processing kernel

```
par_for src in Frontier do
    for dst in out_neigh(src) do
        AtomicUpd (vtxData[dst]), auxData[src])
```

Push-style execution

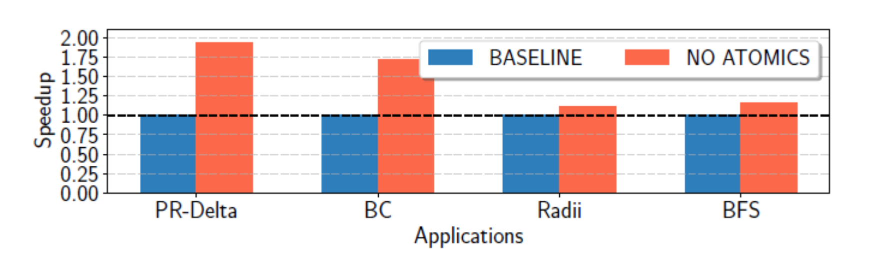
**Algorithm 2** Pull version of graph kernel

```
par_for dst in G do
    for src in in_neigh(dst) do
        if src in Frontier then
            Upd (vtxData[dst]), auxData[src])
```

Pull-style execution

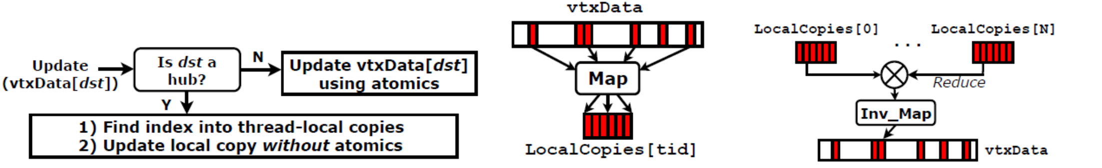
# Motivation

## Slowdown by Atomics



# Motivation

## HUBDUP: Data Duplication as an optimization



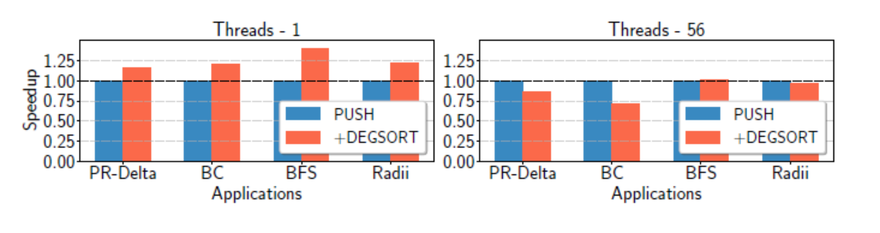

(a) Identifying hub vertices

(b) Gathering hub vertex updates in thread-local copies

(c) Reduction of values in thread-local copies

# Motivation

## Locality preservation by Graph Reordering



# RADAR: Reordering Assisted Duplication/Duplication Assisted Reordering

## Algorithm Description

The main algorithm of RADAR can be summarized as follows:

$$RADAR = DegSort + HUBDUP$$

The steps for RADAR can be formulated as follows:

1. Apply DegSort on the input graph such that hub vertices get the lowest contiguous vertex IDs in the vertex array. This addresses the issue of **identifying** and **locating** the hub vertices for HUBDUP.
2. Do data duplication for HUB vertices. This is essentially the same as carrying out HUBDUP on the reordered input graph. But this time without having to *map* in Step (b) or *Inv_Map* in Step (c). This is because the hub vertex IDs can be used directly as indexes into the thread-local copies (as a simple look-up).

# RADAR: A Critical Evaluation

## Advantages

- Easy identification of hub vertices.
- Easy location of hub vertices.
- Lightweight Reordering by DegSort.
- Easy adaptability to varying cache sizes.

# RADAR: A Critical Evaluation

## Advantages

- Easy identification of hub vertices.
- Easy location of hub vertices.
- Lightweight Reordering by DegSort.
- Easy adaptability to varying cache sizes.

# RADAR: A Critical Evaluation

## Advantages

- Easy identification of hub vertices.
- Easy location of hub vertices.
- Lightweight Reordering by DegSort.
- Easy adaptability to varying cache sizes.

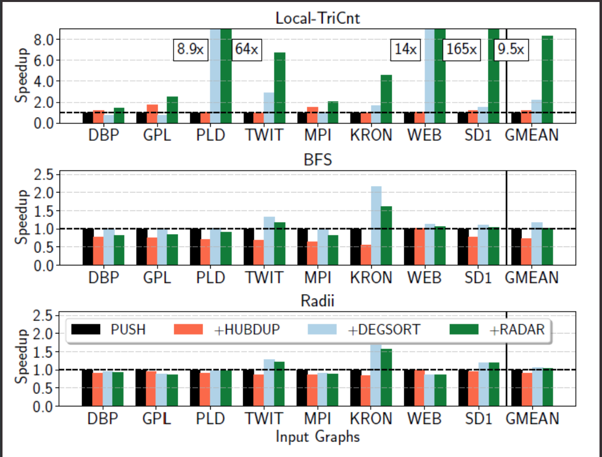
# RADAR: A Critical Evaluation

## Advantages

- Easy identification of hub vertices.
- Easy location of hub vertices.
- Lightweight Reordering by DegSort.
- Easy adaptability to varying cache sizes.

# RADAR: A Critical Evaluation

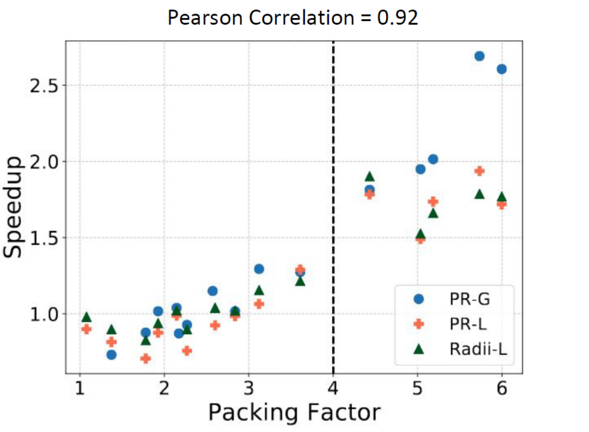
## Disadvantages




RADAR is NOT end-application agnostic.

# RADAR: A Critical Evaluation

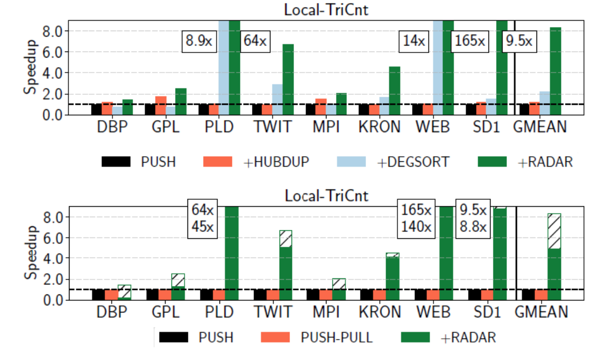
## Disadvantages




[Taken from Balaji et. al, IISWC 2018]

# RADAR: A Critical Evaluation

## Disadvantages




RADAR is NOT input-graph agnostic.

# RADAR: A Critical Evaluation

## Disadvantages

- Another skew-aware reordering technique that is relatively as lightweight as DegSort and employs coarsegrain reordering to preserve graph structure while reducing the cache footprint of hub vertices by binning based on degree is given in *Faldu et. al. IISWC 2019* and could be used as an alternative to DegSort where the natural order of the input graph has been known to already preserve some form of locality.
- The end-applications that are experimented on as part of this work are traditional ones like PageRank, BFS, Triangle Counting, Radii, etc. The list of applications might be inadequate to capture the usefulness of the algorithm. Experiments on more practical end applications in the field of community detection or influence maximization might present a clearer picture of the utility of the proposed work.

# RADAR: A Critical Evaluation

## Disadvantages

- Another skew-aware reordering technique that is relatively as lightweight as DegSort and employs coarsegrain reordering to preserve graph structure while reducing the cache footprint of hub vertices by binning based on degree is given in *Faldu et. al. IISWC 2019* and could be used as an alternative to DegSort where the natural order of the input graph has been known to already preserve some form of locality.
- The end-applications that are experimented on as part of this work are traditional ones like PageRank, BFS, Triangle Counting, Radii, etc. The list of applications might be inadequate to capture the usefulness of the algorithm. Experiments on more practical end applications in the field of community detection or influence maximization might present a clearer picture of the utility of the proposed work.

# Thank You