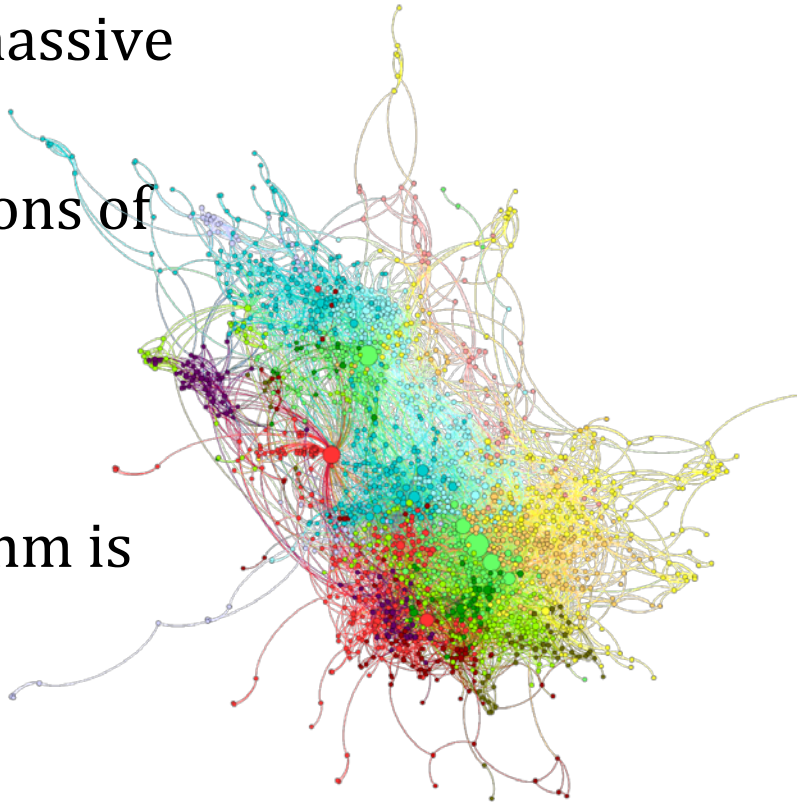
Parallel Graph Algorithms

Maleq Khan

Joint work with Maksudul Alam, Shaikh Arifuzzaman, Hasanuzzaman Bhuiyan, Jiangzhuo Chen, Madhav Marathe, Anil Vulikanti, and Zhao Zhao

Big Graph / Massive Graph

- Some patterns emerges only in massive networks
- Size: billions of vertices and trillions of edges and growing
- Along with time efficiency, space efficiency is also crucial
- Runtime: even $O(n^2)$ time algorithm is not useful



Storing Graph in Main Memory

- Matrix is not acceptable: $O(n^2)$ space
  - A graph with 1 million nodes may take 10-20 TB space.
- Adjacency list takes O(m) space
  - Which we use
- Memory of single machine may not be large enough to hold the entire network

Dealing with Space

- Sparsification / sampling based approximation algorithms
- Streaming algorithms
- External-memory algorithms
- Distributed memory parallel algorithms
  - MPI
  - MapReduce
  - Pregel

Shared Memory System

- Contention in reading/writing – difficult to achieve a good speed up
- Not readily available with large number of processors and memory – costly system
- Distributed-memory cluster of nodes are more commonly available

Distributed-Memory Parallel System

- P processors
- Each processors have local memory
- No shared memory
- Processors communicate with each other by exchanging messages
- Shared disk space

Desired Efficiency

- The graph does not fit in the memory of a single computing node
  - O(m) space is required for the entire network
- O(*n)* space (by each processor) can be acceptable
- Target space: O(m/P), which is best we can do
- Target speedup: P
  - Speedup facor = $T_{\text{sequential}} / T_{\text{parallel}}$
- Number of processors: $P << n$

General Challenges

- Dependencies in computation
- Partitioning the data
- Distributing the computation task among the processors
- Load balancing
- Communication cost and the issue of scaling to large number of processors

A Simple Example

Generate $G(n, p)$ graph:

```
for i = 1 to n do
    for j = i+1 to n do
        add edge (i, j) to the graph with prob. p
```

- $O(n^2)$ time
- Easy to parallelize – time $O(n^2/P)$

A Simple Example (cont.)

Generate $G(n, p)$ graph

- a sequence of Bernoulli trials with success prob. p
- lengths of the streaks of failures are geometric random variables
- generate a geometric random variable x
- Skip x edges and add the next edge

- $O(m)$ time
- Non-trivial to parallelize, but we can achieve a good speed up
  - $O(m/P + f(P))$ time

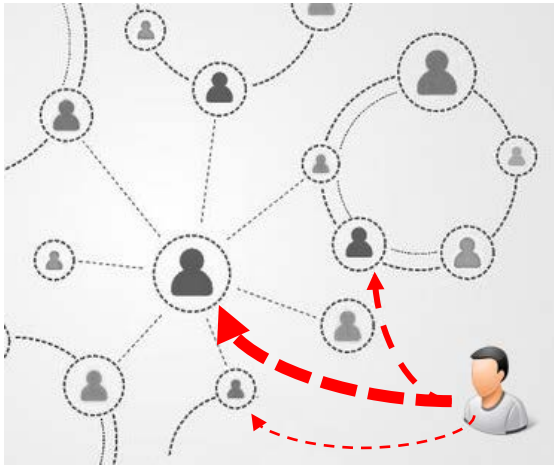
Our Parallel Algorithms

We developed parallel Algorithm for the following problems

- Generating random graphs using preferential attachment model
- Generating random graphs using Chung-Lu model
- Counting/enumerating subgraphs
  - MPI based
  - Hadoop based
- Counting/enumerating triangles
- Switching end points of the edges
- Converting edge list to adjacency list

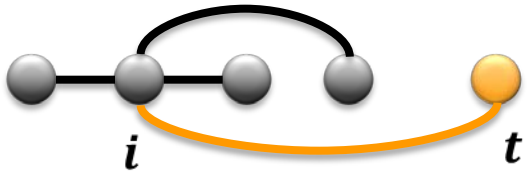
Preferential Attachment (PA) Model

- **Preferential Attachment**
  - A node connects with higher probability to a node which already has large number of connections.
    - WWW: New webpage add links to well known sites
    - Citation: Well cited papers are highly likely to be cited more
- Follows power-law degree distribution



Barabasi-Albert Model

- One of the first preferential attachment model
- Start with n_0 nodes at time $t = 0$
- Each time add a *new node* which creates x new edges with x *existing nodes*
- New node t connects to a node i with probability proportiaonal to its degree

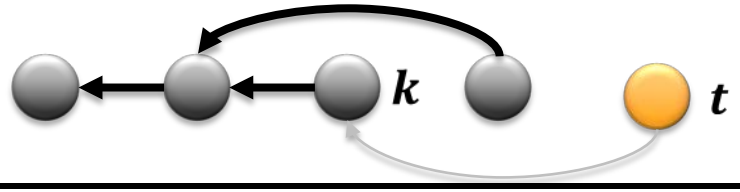


$$Pr(t \to i) = \frac{d_i}{\sum d_j} \qquad d_j = \text{degree of } j\text{-th node}$$

- Does not lead to efficient parallelization

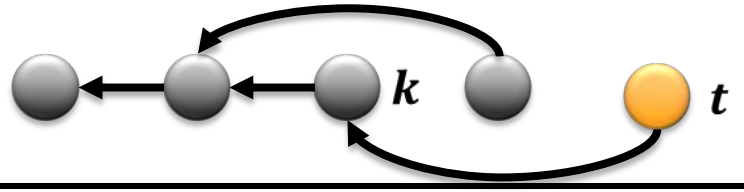
Copy Model

- Another preferential attachment model
- F_t = the node to which a new node t connects ($F_t < t$)
  - We say F_t is the **parent** of node t
- For a new node t
  - **Step 1 (Node Selection)**: a node $k \in [1, t-1]$ is chosen uniformly
  - **Step 2 (Edge Creation)**: Determine F_t as follow:



Copy Model

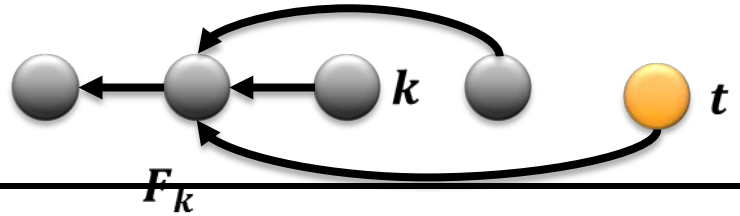
- Another preferential attachment model
- F_t = the node to which a new node t connects ($F_t < t$)
  - We say F_t is the **parent** of node t
- For a new node t
  - **Step 1 (Node Selection)**: a node $k \in [1, t-1]$ is chosen uniformly
  - **Step 2 (Edge Creation)**: Determine F_t as follow:
    - **Direct:** $F_t = k$ with probability p



Direct Edge

Copy Model

- Another preferential attachment model
- F_t = the node to which a new node t connects ($F_t < t$)
  - We say F_t is the **parent** of node t
- For a new node t
  - **Step 1 (Node Selection)**: a node $k \in [1, t-1]$ is chosen uniformly
  - **Step 2 (Edge Creation)**: Determine F_t as follow:
    - **Direct:** $F_t = k$ with probability p
    - **Copy:** $F_t = F_k$ with probability $1 - p$



Copy Edge

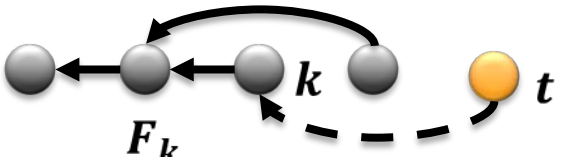
Parallel Copy Model (x=1)

- Each node create $x = 1$ new edge
- $V = \{1,2,3, \dots, n\}$ is the set of n nodes
- Nodes are distributed into $\boldsymbol{P}$ disjoint sets: $V_1, V_2, \dots, V_P$
  - $V_i \cap V_j = \emptyset$ for any $i \neq j$ and $\cup_i V_i = V$ for $1 \leq i \leq P$

- Processor $\boldsymbol{P_i}$ computes $\boldsymbol{F_t}$ for every node $\boldsymbol{t} \in \boldsymbol{V_i}$
  - $\boldsymbol{P_i}$ **independently** compute Step 1 (Node Selection)
  - **Direct edges** are also determined **independently**
  - Only **Copy edges** are dependent on previous network and require inter-processor communication

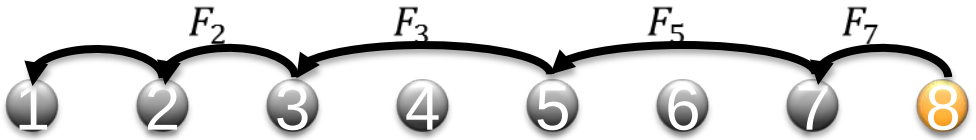
Dependency Chain

- To make a connection a node need to know the previous network



F_t connects to F_k, hence we say t is dependent on k

- Such dependency can form a chain



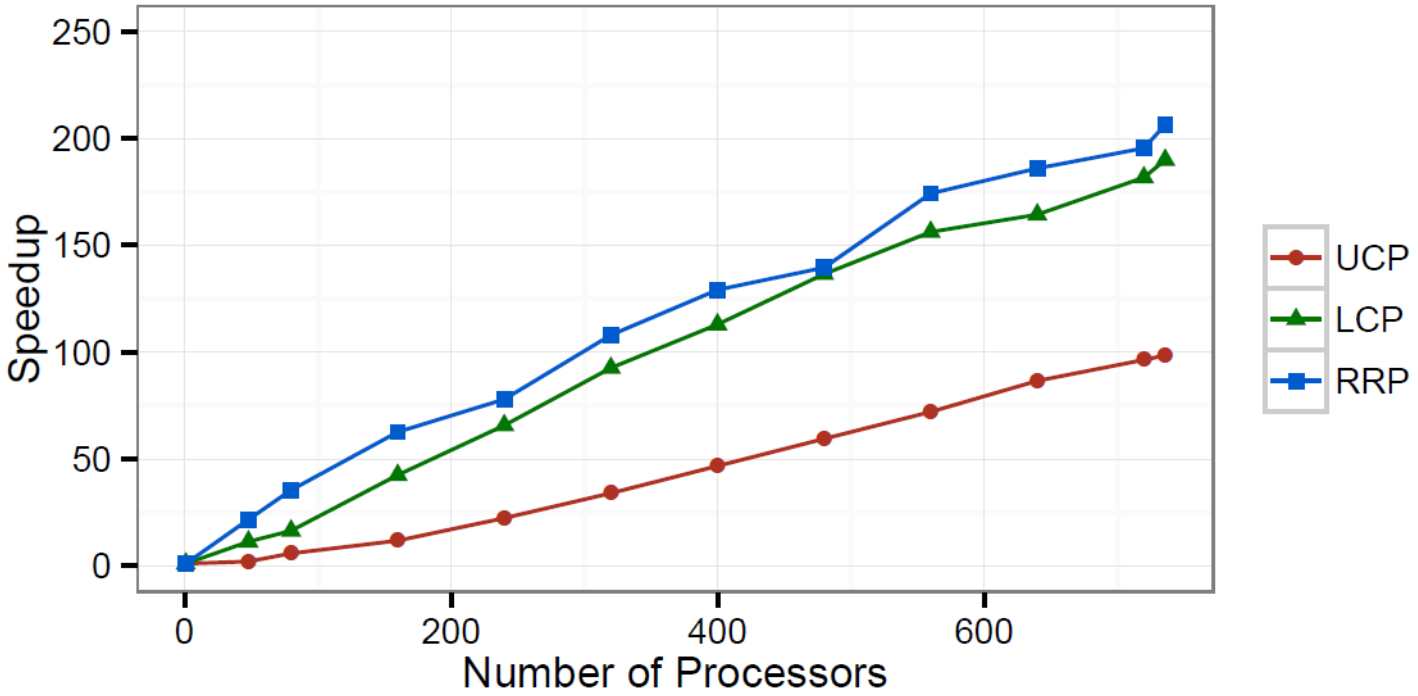
8 is dependent on 7, which is dependent on 5 and so on…

- A node has to wait until all the nodes in the dependency chain is computed
  - Might led to inefficient algorithm if the length of such chain is very long

Length of Dependency Chain

- **Theorem:** Let L_t be the length of the dependency chain starting at node t and $L_{max} = \max_t L_t$. Then the expected length $E[L_t] \leq \log n$ and $L_{max} = O(\log n)$ w.h.p., where n is the number of nodes.
- The maximum length of dependency chain is bound by $O(\log \boldsymbol{n})$
  - The average length of dependency chain is $\frac{1}{p}$
- Leads to efficient parallel algorithm, as there is **less** dependency
- A processor hardly remains idle as it has other nodes to work with

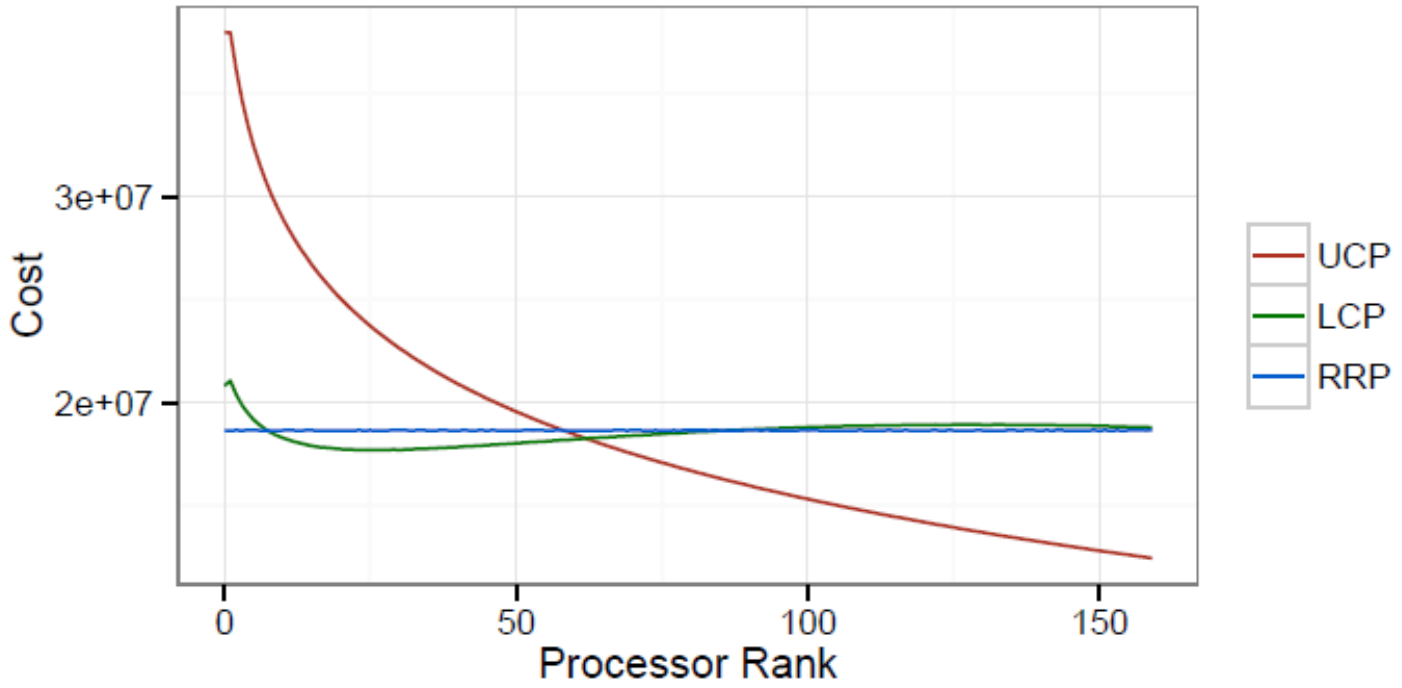
Algorithm for PA Model: Strong Scaling



- $n=10^9$, x=6
- Linear speed-up using 768 processors
- Able to generate **400 Billion edges** within **5 minutes** using **768** procs.

Comparison of Partitioning Algorithms

- Computational Cost:
  - LCP and RRP shows good load balancing

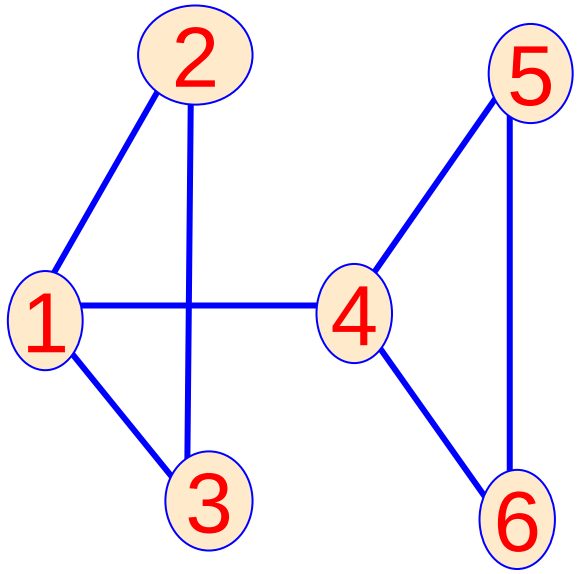


Publications

- Distributed-Memory Parallel Algorithms for Generating Massive Scale-free Networks Using Preferential Attachment Model
  Maksudul Alam, Maleq Khan, and Madhav V. Marathe
  Intl. Conf. for High Performance Computing, Networking, Storage and Analysis(SuperComputing), Denver, Nov. 2013.

Edge List and Adjacency List

- Edge list
  - In most cases, graphs are generated as list of edges
  - Edge denotes a link between a pair of entities
- Adjacency list
  - Graph algorithms work efficiently if information of adjacent nodes for each node is readily available.



{1,2}
{1,3}
{1,4}
{2,3}
{4,5}
{4,6}
{5,6}

Edge List

N_1={2,3,4}
N_2={1,3}
N_3={1,2}
N_4={1,5,6}
N_5={4,6}
N_6={4,5}

Adjacency List

- Scanning all neighbors of node v:
  - Edge list: $O(m)$ time
  - Adjacency list: $O(d_v)$time

Sequential Conversion

Conversion is trivial in a sequential setting

for each v ∈ V, $N_v \leftarrow \varnothing$

for each edge (u, v) ∈ E **do**

$N_v \leftarrow N_v \cup \{u\}$

$N_u \leftarrow N_u \cup \{v\}$

How to Parallelize

- Phase 1- Local adjacency list:
  - Set of edges E is partitioned into P initial partitions E_i, having almost m/P edges in each partition
  - Processor i works on E_i and construct local adjacency lists
  - Runtime and space complexity of Phase 1 is O(m/P).
  - Computational loads are balanced.

Local computation

each processor *i* executes in parallel:

for each edge (u, v) $\in E_i$ **do**

$$N_v^i \leftarrow N_v^i \cup \{u\}$$

$$N_u^i \leftarrow N_u^i \cup \{v\}$$

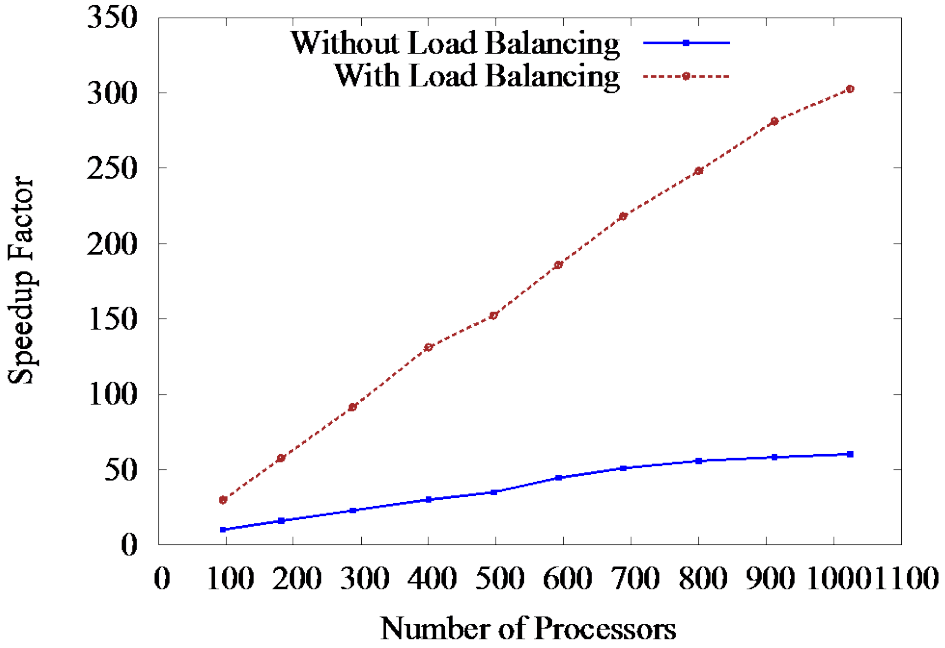
How to Parallelize

- Phase 2- Merging:
  - Dedicated merger: for each node v, runtime $O(d_v)$. A total runtime of $O(m)$ which is as good as sequential algorithm.
  - Requires parallel merging.

$$N_v = \bigcup_j N_v^j$$

  - Load balancing is a non-trivial problem in this phase
  - Have each processor merge for different set of nodes
  - Require a new partitioning to have balanced load.

Conversion: Speedup



Performance on Twitter network

- Our algorithm achieves a speedup factor of ~300 with 1024 processors.
- Almost linear speedup up to a large number of processors.
- Load balancing improves performance significantly.

Conversion

Publications

• Fast Parallel Conversion of Edge List to Adjacency List for Large-Scale Graphs

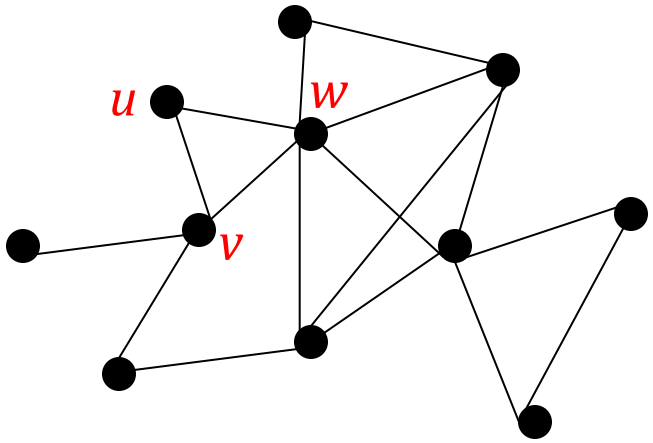
Shaikh Arifuzzaman and Maleq Khan

23rd High Performance Computing Symposium (HPC), Alexandria, VA, USA, April 2015.

Triangles in a Network

Given a network $G(V, E)$,

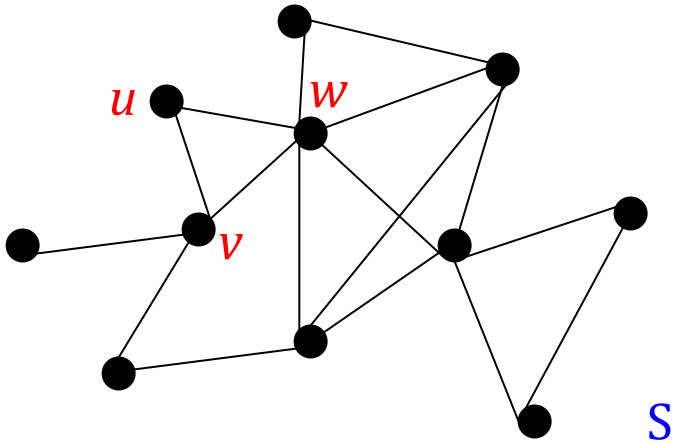
(u, v, w) is a triangle if (u, v), (v, w), (w, u) are edges in E.



Triangles in a network

Given a network $G(V, E)$,

(u, v, w) is a triangle if (u, v), (v, w), (w, u) are edges in E.



Set of neighbors of v

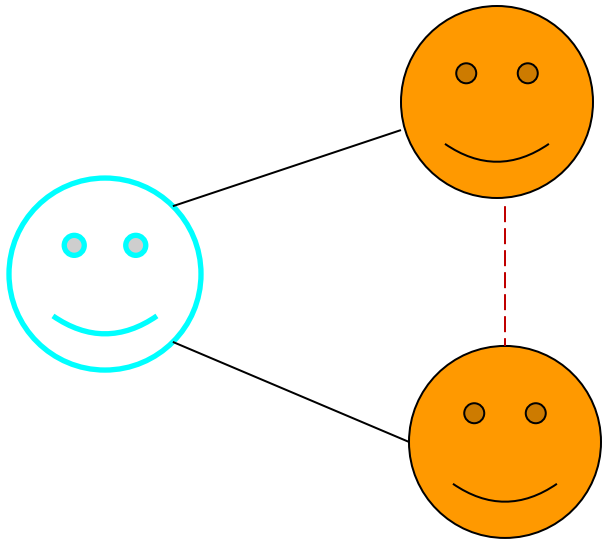
Number of triangles incident on v

$$T_v = |\{(u,w) \in E \mid u, w \in N(v)\}|$$

$T_v = 1$
$T_w = 5$
Total, $T = 7$

Social Theory: Transitivity

Friends of a friend tend to become friends themselves and form triangles! [Wasserman Faust '94]



P Erdös, R Graham, and F Chung

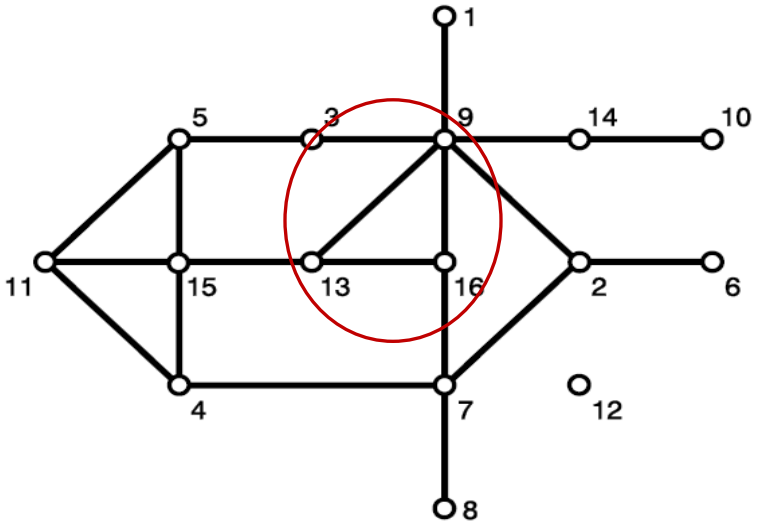
Applications

- Analysis of complex networks: clustering coefficients and transitivity ratio [Watts,Strogatz'98]
- Modeling microscopic evolution of social networks by triangle closing [Leskovec et.al., KDD '08]
- Solving systems of geometric constraints involves triangle counting [Fudos, Hoffman 1997]
- Many other applications: Motif Detection/ Frequent Subgraph Mining (e.g., Protein-Protein Interaction Networks), Community Detection [Berry et al. '09], Outlier Detection [Tsourakakis '08]

Sequential Algorithm: NodeIterator++

- NodeIterator++ (Latapy[2008], Shank[2007], Suri[2011]) uses a total order $\prec$ of nodes to avoid duplicate count of triangles.
  - A degree-based order reduces running time significantly.

$$u \prec v \Leftrightarrow (d_u < d_v) \vee (d_u = d_v \wedge u < v)$$



$13 \prec 16 \prec 9$

for $v \in V$ ***do***

  for $u \in N(v)$ and $v \prec u$ ***do***

    for $w \in N(v)$ and $u \prec w$ ***do***

      if $(u,v) \in E$ ***then***

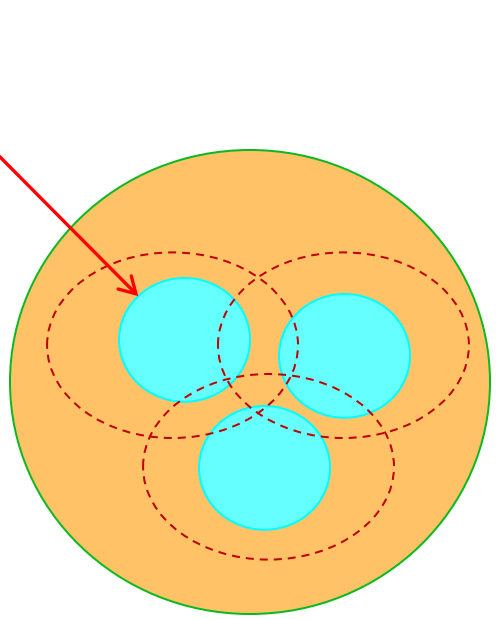
        $T \leftarrow T + 1$

Partitioning the Network

- V is partitioned into P disjoint subsets V_i^c (core nodes in proc. i)

$$V_i^c \cap V_j^c = \emptyset, \quad \text{for } i \neq j$$

$$\bigcup_i V_i^c = V$$



Partitions of a network

Partitioning the Network

- V is partitioned into P disjoint subsets V_i^c

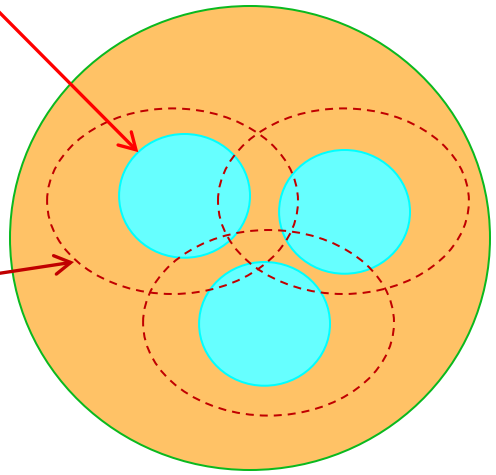
$$V_i^c \cap V_j^c = \varnothing, \quad \text{for } i \neq j$$

$$\bigcup_i V_i^c = V$$

- Partition i is subgraph $G_i(V_i, E_i)$, where

$$V_i = V_i^c \cup \bigcup_{v \in V_i^c} N_v$$

$$E_i = \{(u,v) \mid u,v \in V_i \text{ and } (u,v) \in E\}$$



Partitions of a network

Partitioning the Network

- V is partitioned into P disjoint subsets V_i^c

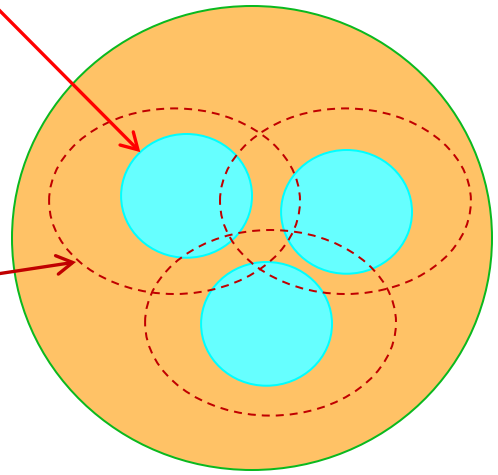
$$V_i^c \cap V_j^c = \emptyset, \quad \text{for } i \neq j$$

$$\bigcup_i V_i^c = V$$

- Partition i is subgraph $G_i(V_i, E_i)$, where

$$V_i = V_i^c \cup \bigcup_{v \in V_i^c} N_v$$

$$E_i = \{(u,v) \mid u,v \in V_i \text{ and } (u,v) \in E\}$$



Partitions of a network

Partitioning of V crucially affects load balancing

Load balancing schemes

Define a cost function $f(v)$ = cost to count triangles incident on node v

Now partition V such that

$$\sum_{v \in V_i^c} f(v) \approx \frac{1}{P} \sum_{v \in V} f(v)$$

- Exact computation of $f(v)$ may not be possible
- We estimate $f(v)$ with various functions

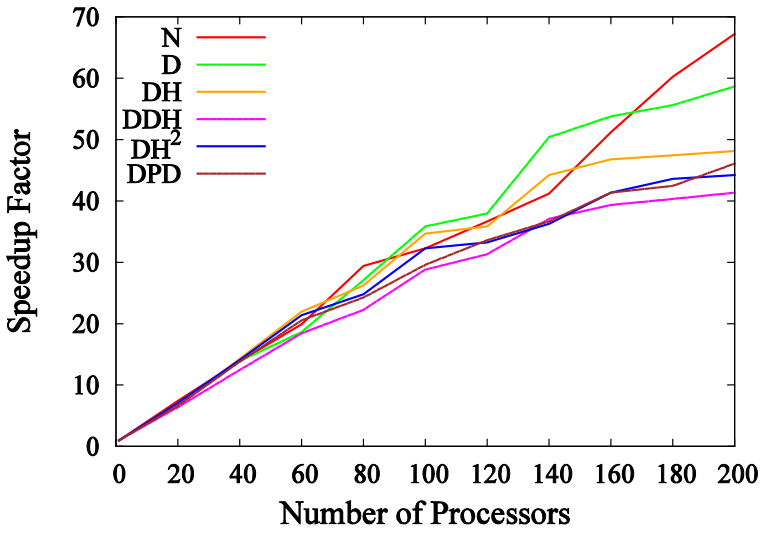
Estimating Computing Load

How to estimate *f(v)*?

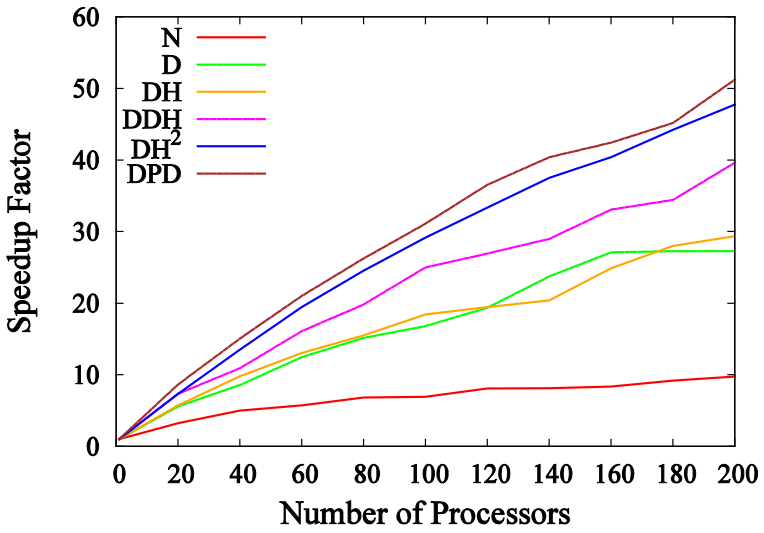
Time complexity: $$O\left(\sum_{v\in V}\sum_{u\in N_v}\left(\hat{d}_v+\hat{d}_u\right)\right)=O\left(\sum_{v\in V}d_v\hat{d}_v\right)$$

Cost Functions	Notations	
$f(v)=1$	N	← Equal # of nodes/proc.
$f(v)=d_v$	D	← Equal # edges / proc.
$f(v)=\hat{d}_v$	DH	
$f(v)=d_v\hat{d}_v$	DDH	
$f(v)=\hat{d}_v^2$	DH^2	
$f(v)=\sum_{u\in N_v}\left(\hat{d}_v+\hat{d}_u\right)$	DPD	← Best load balancing, but costly to compute

Counting Triangles: Speedup



Miami Network



LiveJournal Network

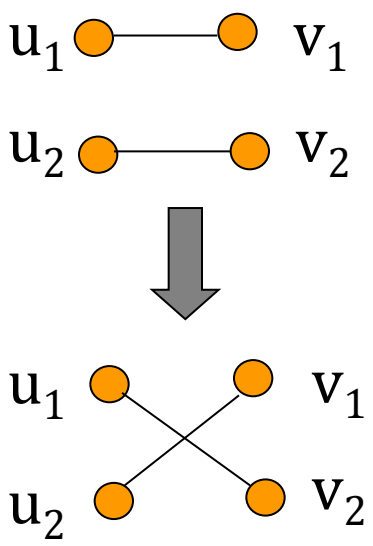
- Good speedup factor and scales to large number of processors
- 16 minutes for a network with 10 billions edges

Publications

PATRIC: A Parallel Algorithm for Counting Triangles in Massive Networks
Shaikh Arifuzzaman, Maleq Khan, and Madhav V. Marathe
ACM Conference on Information and Knowledge Management (CIKM), San Francisco, Oct. 2013.

Edge Switching Operation

Switching operations



Replace edges (u_1, v_1) and (u_2, v_2) with (u_1, v_2) and (u_2, v_1)

1. Randomly pick two edges of the graph and switch their end nodes
2. Repeat the above step until the desired number of edges are switched

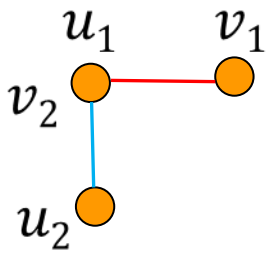
- Preserves degree distribution
- Allow us to study the space of networks with the same degree distribution

Edge Switching Problem

- Input:
  - A simple graph
    - No loops and parallel edges
  - The number of edge switches, t
- Sequential Processing:
  - Select pair of edge uniformly at random
  - Edge switching is performed only if the graph remains simple
  - This process is repeated until t number of edge switches are done
- Output:
  - A simple graph

Constraints of Edge Switch

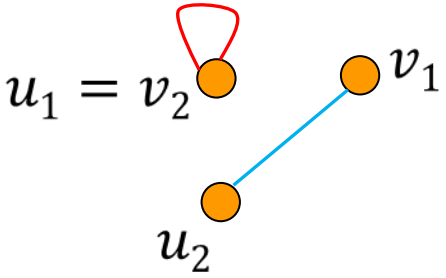
- The output remains simple graph after switching
  - **No Loop**
  - No Parallel Edge



Before switch

Conditions:

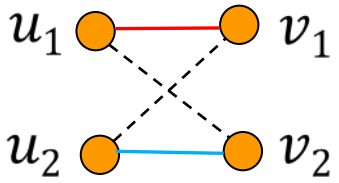
- $u_1 \neq v_2$
- $u_2 \neq v_1$



After switch

Constraints of Edge Switch

- The output remains simple graph after switching
  - No Loop
  - **No Parallel Edge**

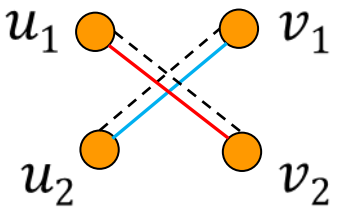


Before switch

Conditions:

- $u_1 \notin N(v_2)$
- $u_2 \notin N(v_1)$

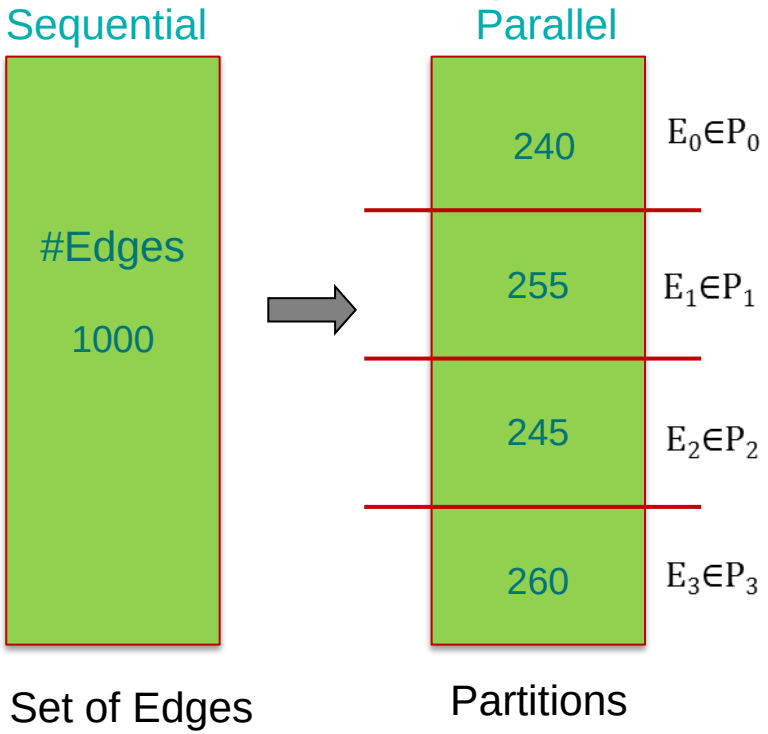
- $N(v)$ is the set of neighbors of v



After switch

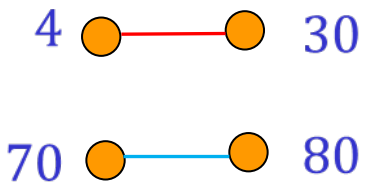
Parallel Edge Switching

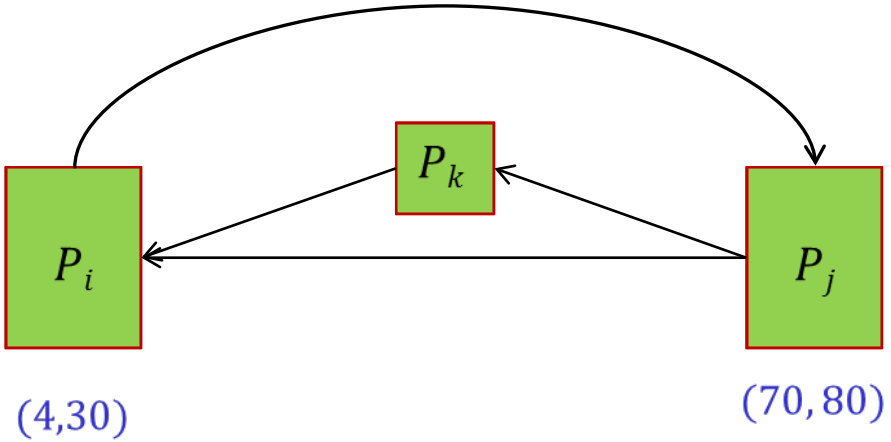
- Partitioning:
  - Nodes are sorted according to their node ids
  - Each processor P_i contains a disjoint set of edges E_i
  - Each partition contains almost equal number of edges
  - Ensure that one node's adjacency list belongs to only one processor



Parallel Edge Switching

- An edge getting replaced after edge switch may belong to a different processor P_k
  - Because of keeping only one copy of each edge (u, v) such that $u < v$

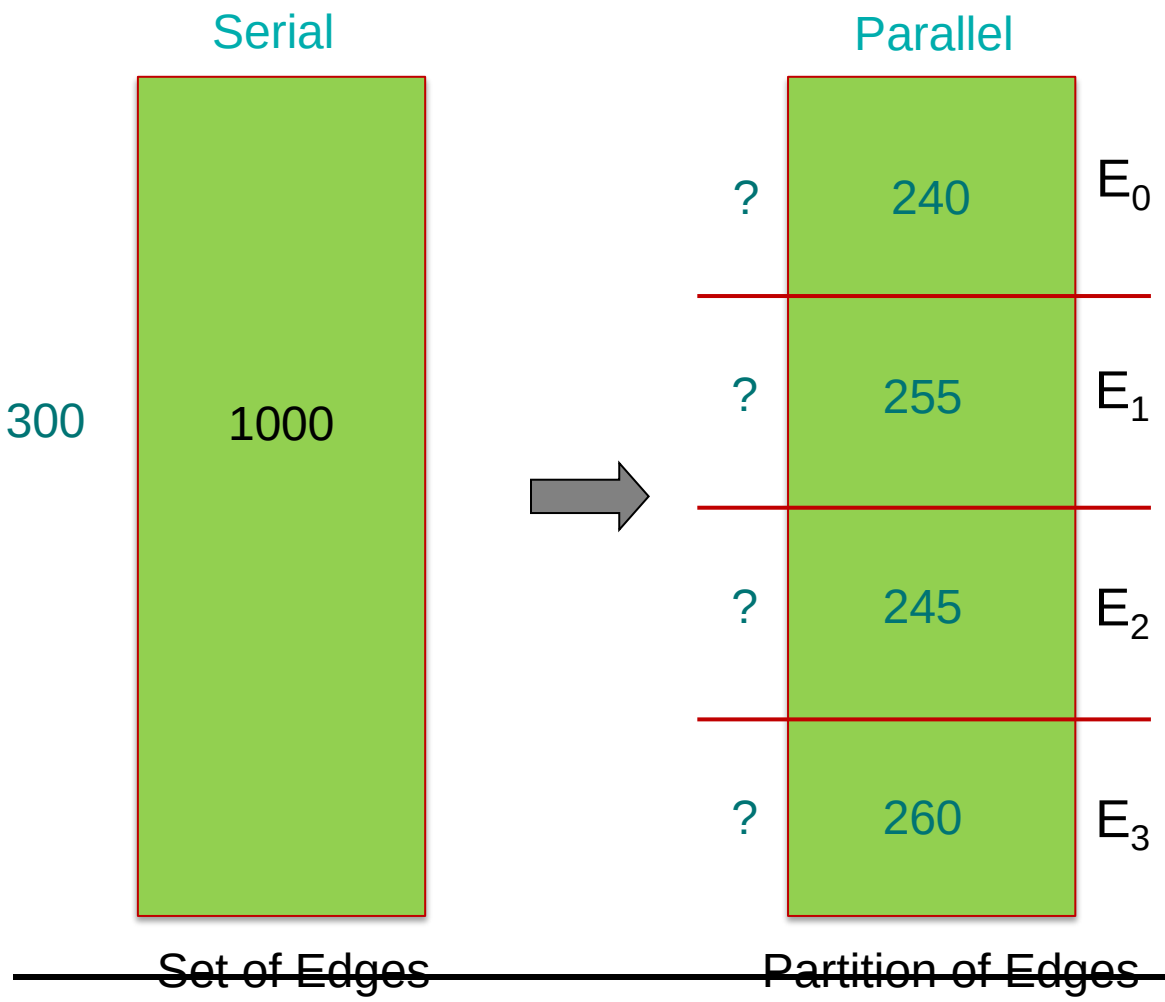




Parallel Edge Switching

- Challenges
  - Same new edge can be created by different pairs of processors at the same time
    - Example:
    - Consider the following switch
    - $(u_1, v_1), (u_2, v_2)$ ⇒ $(u_1, v_2), (u_2, v_1)$
    - The edge (u_1, v_2) can be created by following ways
      - $(u_1, \_), (\_, v_2)$
      - $(\_, u_1), (v_2, \_)$
    - The edge (u_2, v_1) can be created by following ways
      - $(u_2, \_), (\_, v_1)$
      - $(\_, u_2), (v_1, \_)$

Picking Edges Uniformly at Random

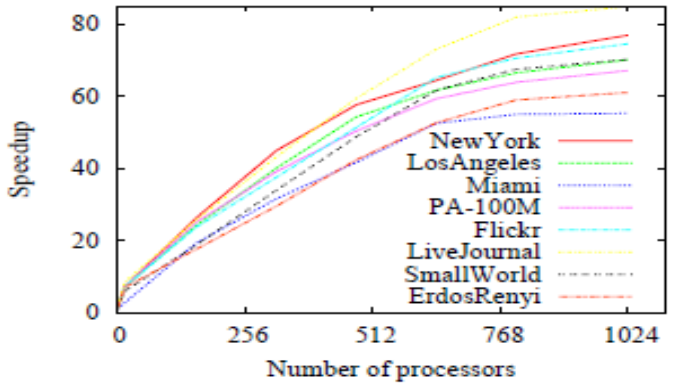


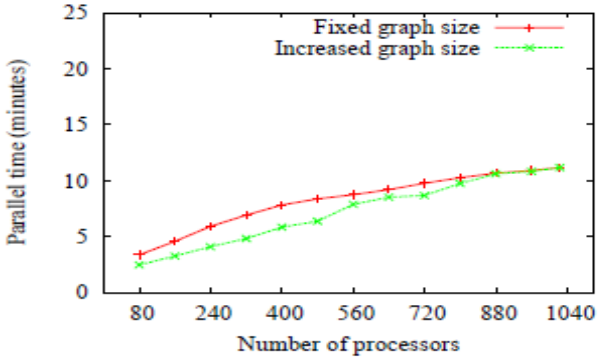
Question:
How do we know the number of first edges that will be picked from P_i in advance without actually picking the edges?

Answer:
Multinomial Distribution

Parallel Edge Switching: Performance

- Speedup of 85 using 1024 processors
- Can switch more than 115B edge switches on a Pref. Attachment graph with 10B edges in less than 3 hours





Strong scaling

$$x = 1, stepsize = \frac{t}{100}$$

Weak scaling

using Pref. Attch. Graph

$$t = p \times 10M, stepsize = t/1000$$

Publications

Fast Parallel Algorithms for Edge-Switching to Achieve a Target Visit Rate in Heterogeneous Graphs

Hasanuzzaman Bhuiyan, Jiangzhuo Chen, Maleq Khan, and Madhav V. Marathe

International Conference on Parallel Processing (ICPP), Minneapolis, Sep. 2014.

Chung-Lu Model

- Generate a Random Graph from a given degree sequence
- a set of $\boldsymbol{n}$ nodes $\boldsymbol{V} = \{0,1,2,\dots,n-1\}$
- A set of weights $\boldsymbol{w} = \{w_0, w_1, w_2, \dots, w_{n-1}\}$
  - Weight $\boldsymbol{w_i}$ defines the expected degree of node $\boldsymbol{i}$
- Probability of an edge between nodes $\boldsymbol{i}$ and $\boldsymbol{j}$ is defined as:

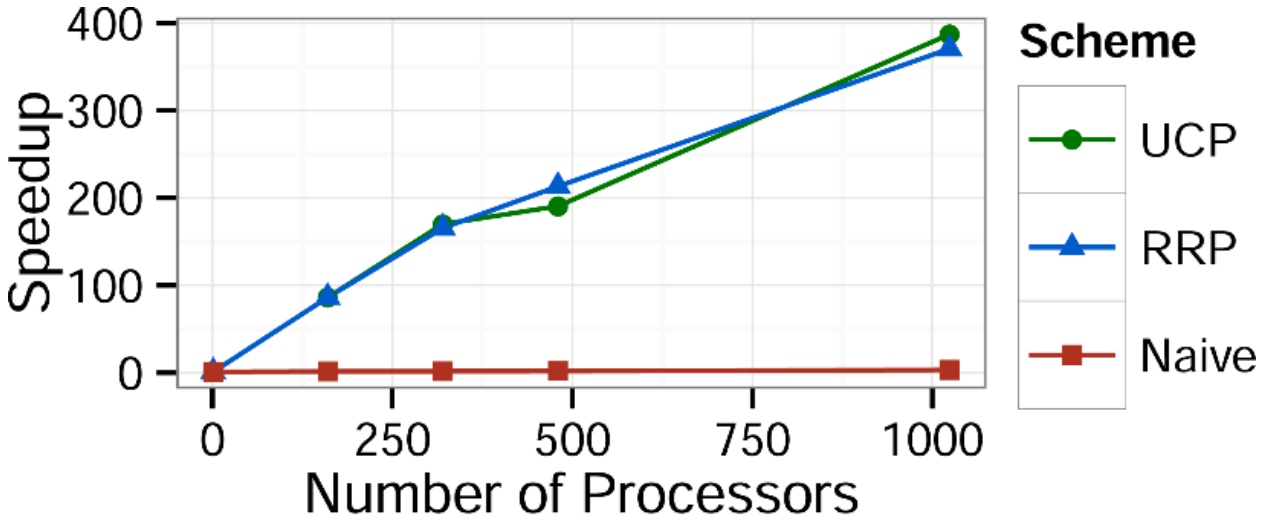
$$p_{i,j} = \frac{w_i w_j}{\sum_{k=0}^{n-1} w_k}$$

- If no self-loop is allowed ($i \neq j$), **expected degree** of node $\boldsymbol{i}$ is:

$$\deg(i) = \sum_j \frac{w_i w_j}{\sum_k w_k} = w_i - \frac{w_i^2}{\sum_k w_k}$$

Chung-Lu Model: Strong Scaling

- Twitter Network
- $n = 41.65 \times 10^6, m = 1.37 \times 10^9$
- Avg. degree = 33

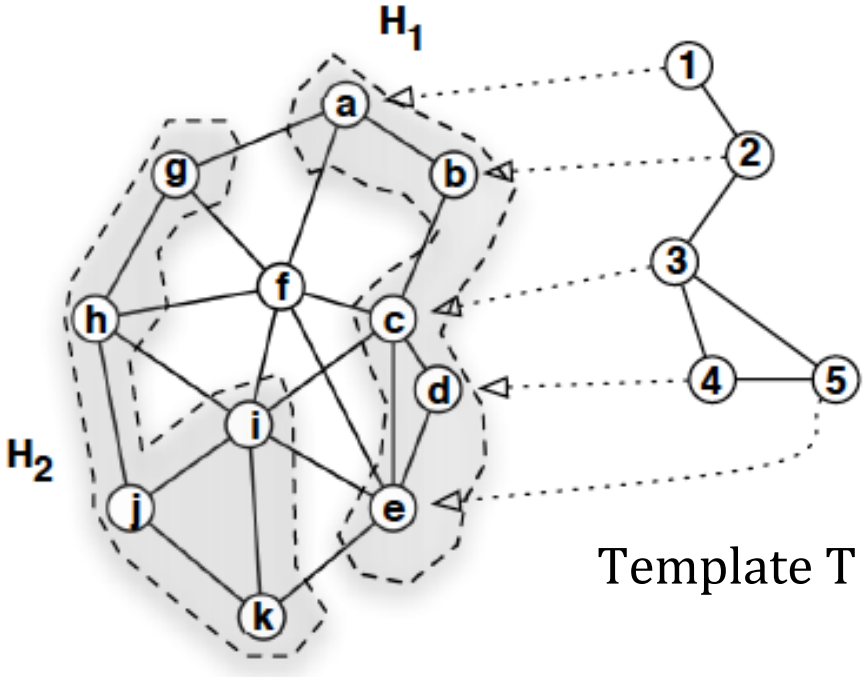


Subgraph Enumeration

Given a large graph G(V, E) and a smaller template $T(V_T, E_T)$, find the number of subgrahs $H(V_H, E_H)$ of G such that H is isomorphic to T.

H is isomorphic to T if there is a one-to-one mapping $f: V_H \rightarrow V_T$ such that
$(u, v) \in V_H$ if and only if $(f(u), f(v)) \in V_T$

Subgraph Enumeration (cont.)



H_1 is an induced and H_2 is a non-induced subgraphs

Subgraph Enumeration - Publications

SAHAD: Subgraph Analysis in Massive Networks Using Hadoop
Zhao Zhao, Guanying Wang, Ali Butt, Maleq Khan, V.S. Anil Kumar, and Madhav Marathe.
26th IEEE International Parallel & Distributed Processing Symposium (IPDPS), Shanghai, China, May 2012.

Subgraph Enumeration in Large Social Contact Networks using Parallel Color Coding and Streaming
Zhao Zhao, Maleq Khan, V.S. Anil Kumar and Madhav Marathe.
39th International Conference on Parallel Processing (ICPP), San Diego, California, Sep. 2010.