External-Memory Network Analysis Algorithms for Naturally Sparse Graphs

Michael T. Goodrich

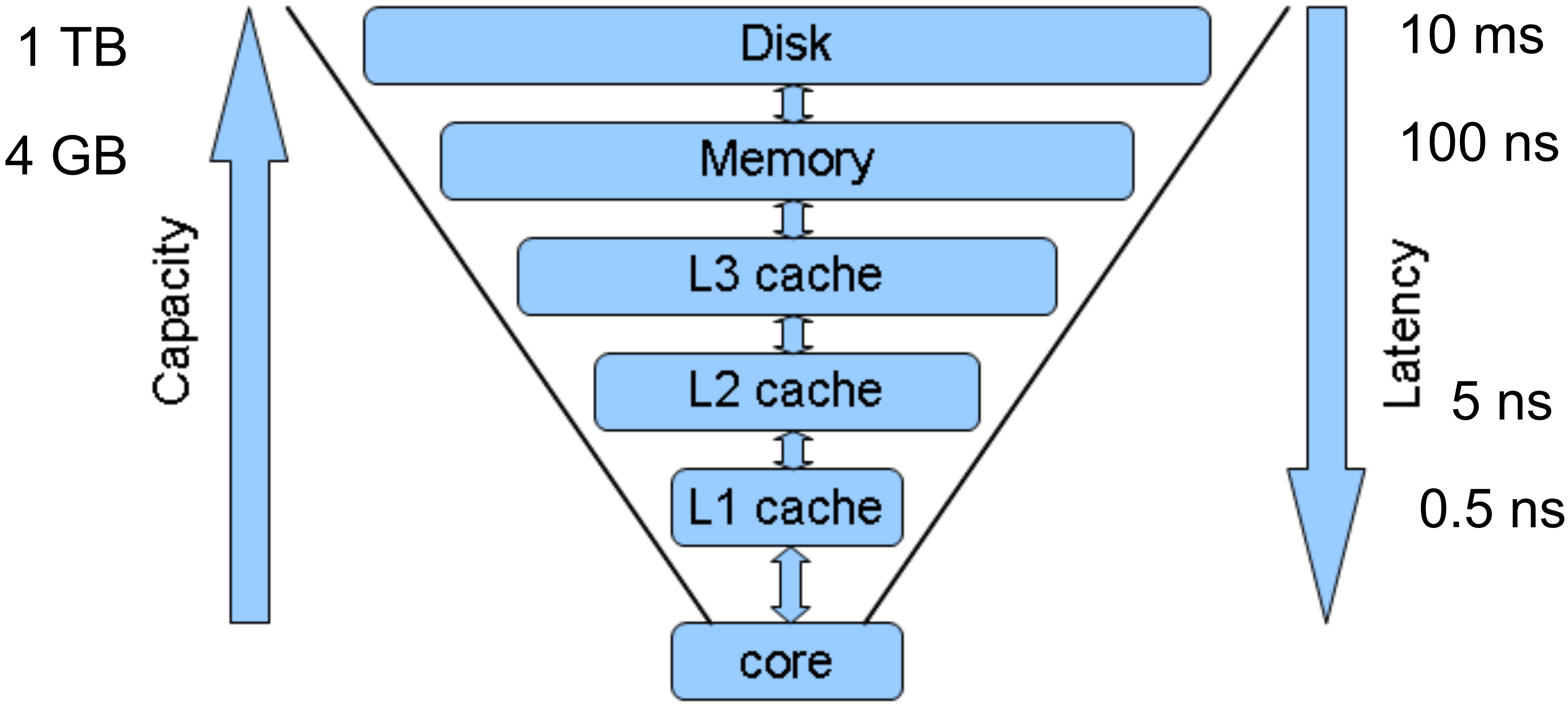
joint w/ Pawel Pszona

Dept. of Computer Science

UNIVERSITY of CALIFORNIA IRVINE

The Memory Hierarchy

- The trade-off of size and speed



External-Memory Model

- Problem: datasets too big to fit in memory
- External-Memory Model: manage disk/memory transfers manually! [Vitter]
- Parameters:
  - M – memory capacity
  - N – $\#$ stored items
  - D – $\#$ disks
  - B – block size
- We measure the number of I/O's between disks and memory
- Key bounds:
  - $scan(N) = \theta(\frac{N}{DB})$
  - $sort(N) = \theta(\frac{N}{DB} \log_{M/B} \frac{N}{B})$

Graph Sparsity

- *k*-core

  maximal connected subgraph with all vertices having degree $\geq k$

- *k*-core number

  maximal k s.t. graph has a k-core

- degeneracy

  graph has degeneracy d, if d is the smallest number s.t. every subgraph has a vertex of degree $\leq d$

Graph Sparsity

- k-core

  maximal connected subgraph with all vertices having degree $\geq k$

- k-core number

  maximal k s.t. graph has a k-core

- degeneracy

  graph has degeneracy d, if d is the smallest number s.t. every subgraph has a vertex of degree $\leq d$

Fact

Graph has degeneracy d iff its k-core number is equal to d.
We call such graph d-degenerate

Naturally Sparse Graphs

Definition

We call a graph naturally sparse if it has constant degeneracy

Naturally Sparse Graphs

Definition

We call a graph naturally sparse if it has constant degeneracy

- Planar graphs ($d \leq 5$)
- Random graphs [Pittel *et al. (1996), Riordan (2008)]*
- Generated graphs [Barabasi and Albert (1999), Kleinberg (2000)]
- Real world graphs [Eppstein and Strash (2011)]

Degeneracy Ordering

Definition

d-degeneracy ordering of G – ordering L of vertices of G s.t. each vertex has at most d neighbors that are later in L

Fact

d-degenerate graph always has a d-degeneracy ordering

Some Notation

- d – graph degeneracy
- n – # vertices
- m – # edges ($m = O(dn)$)

Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**

  $v \leftarrow$ vertex of smallest degree in G

  remove v from G and place it at the end of ordering

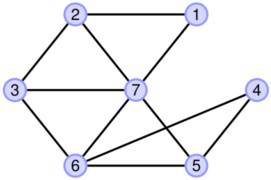
Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**
  $v \leftarrow$ vertex of smallest degree in G
  remove v from G and place it at the end of ordering

- Example:

G



L

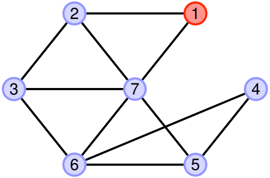
Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**
  $v \leftarrow$ vertex of smallest degree in G
  remove v from G and place it at the end of ordering

- Example:

G L

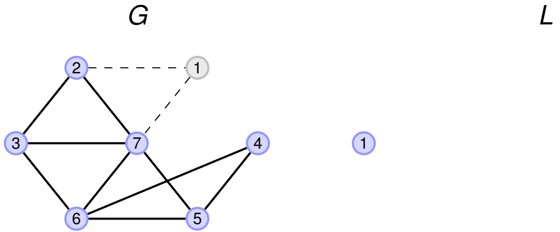


Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**
  $v \leftarrow$ vertex of smallest degree in G
  remove v from G and place it at the end of ordering

- Example:



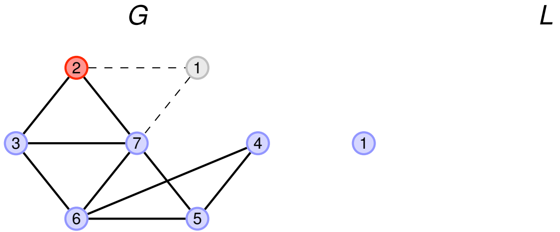
Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**

  $v \leftarrow$ vertex of smallest degree in G

  remove v from G and place it at the end of ordering
- Example:

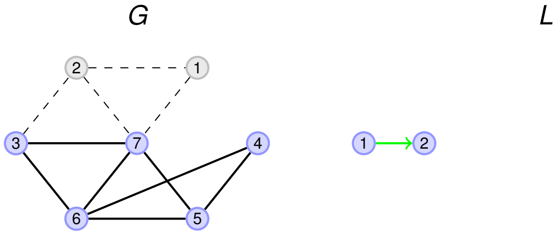


Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**
  $v \leftarrow$ vertex of smallest degree in G
  remove v from G and place it at the end of ordering

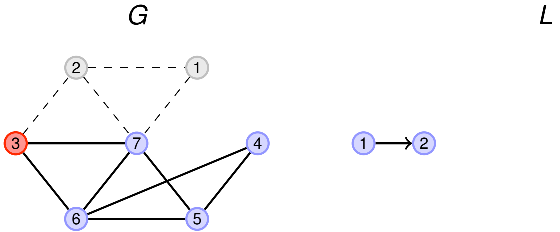
- Example:



Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**
  $v \leftarrow$ vertex of smallest degree in G
  remove v from G and place it at the end of ordering
- Example:

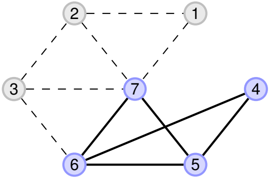


Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**
  $v \leftarrow$ vertex of smallest degree in G
  remove v from G and place it at the end of ordering
- Example:

G



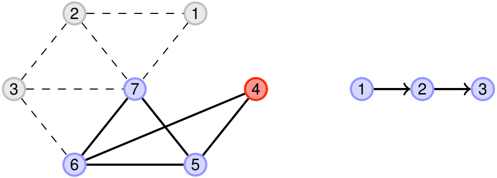
L

Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**
  $v \leftarrow$ vertex of smallest degree in G
  remove v from G and place it at the end of ordering
- Example:

G L

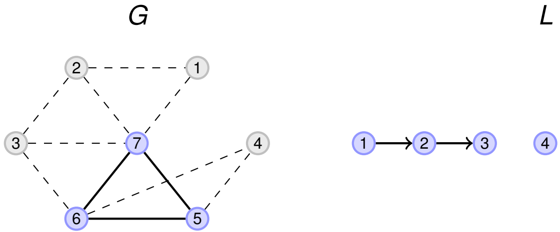


Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**
  $v \leftarrow$ vertex of smallest degree in G
  remove v from G and place it at the end of ordering

- Example:



Sequential Case

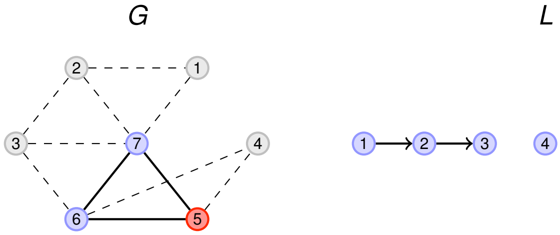
- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**

  $v \leftarrow$ vertex of smallest degree in G

  remove v from G and place it at the end of ordering

- Example:



Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:

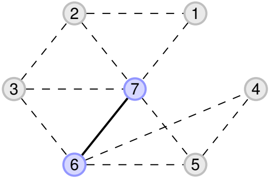
  while G is nonempty **do**

  $v \leftarrow$ vertex of smallest degree in G

  remove v from G and place it at the end of ordering

- Example:

G



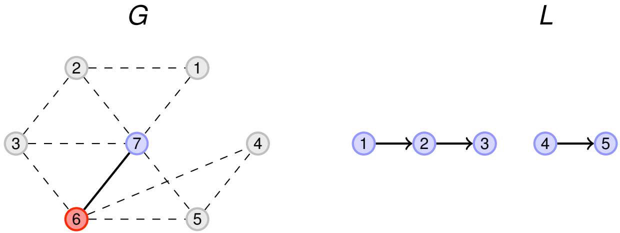
L

Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**
  $v \leftarrow$ vertex of smallest degree in G
  remove v from G and place it at the end of ordering

- Example:

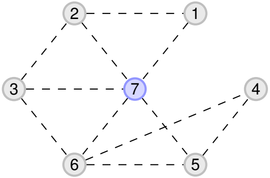


Sequential Case

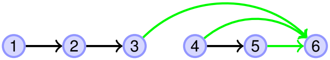
- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**
  $v \leftarrow$ vertex of smallest degree in G
  remove v from G and place it at the end of ordering
- Example:

G



L



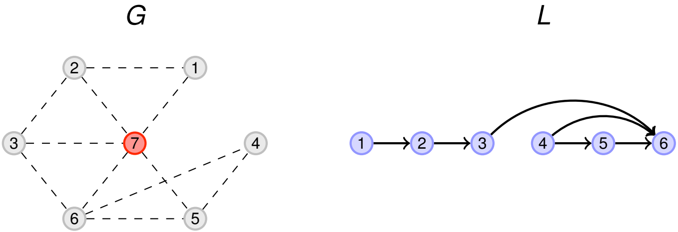
Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**

  $v \leftarrow$ vertex of smallest degree in G

  remove v from G and place it at the end of ordering
- Example:



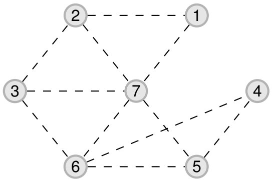
Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:

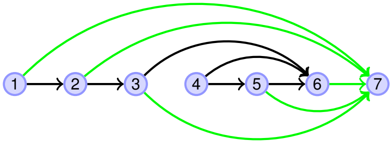
  while G is nonempty **do**
  $v \leftarrow$ vertex of smallest degree in G
  remove v from G and place it at the end of ordering

- Example:

G



L



Sequential Case

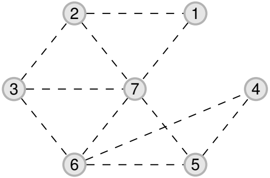
- Degeneracy ordering easy to compute
- Algorithm:

  while G is nonempty **do**

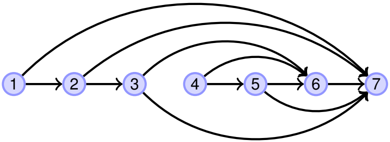
  $v \leftarrow$ vertex of smallest degree in G

  remove v from G and place it at the end of ordering
- Example:

G



L



External-Memory Case

- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):

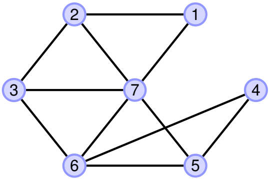
  while G is nonempty **do**
  $S \leftarrow n\epsilon/(2+\epsilon)$ vertices of smallest degree in G
  remove S from G and place at the end of ordering

External-Memory Case

- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):

  while G is nonempty **do**
  $S \leftarrow n\epsilon/(2+\epsilon)$ vertices of smallest degree in G
  remove S from G and place at the end of ordering
- Example ($\epsilon = 1$):

G L

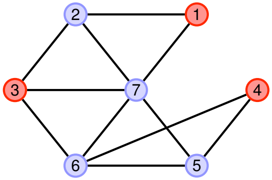


External-Memory Case

- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):

  while G is nonempty **do**
  $S \leftarrow n\epsilon/(2+\epsilon)$ vertices of smallest degree in G
  remove S from G and place at the end of ordering
- Example ($\epsilon = 1$):

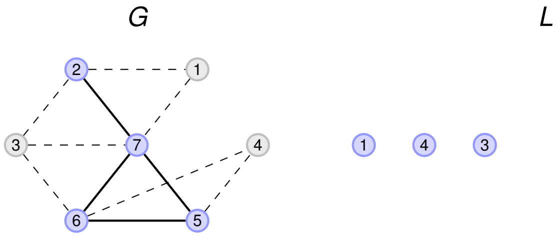
G L



External-Memory Case

- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):

  while G is nonempty **do**
  $S \leftarrow n\epsilon/(2+\epsilon)$ vertices of smallest degree in G
  remove S from G and place at the end of ordering
- Example ($\epsilon = 1$):

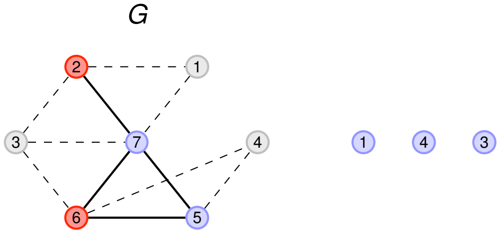


External-Memory Case

- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):

  while G is nonempty **do**
  $S \leftarrow n\epsilon/(2+\epsilon)$ vertices of smallest degree in G
  remove S from G and place at the end of ordering

- Example ($\epsilon = 1$):

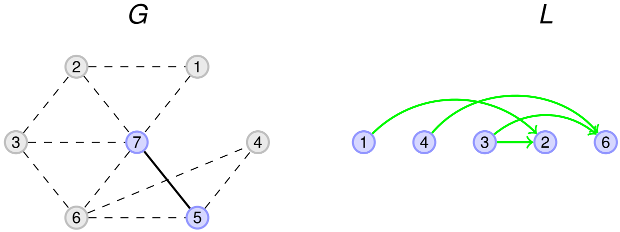


External-Memory Case

- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):

  while G is nonempty **do**
  $S \leftarrow n\epsilon/(2+\epsilon)$ vertices of smallest degree in G
  remove S from G and place at the end of ordering

- Example ($\epsilon = 1$):

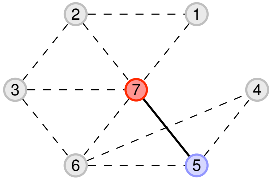


External-Memory Case

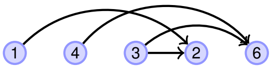
- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):

  while G is nonempty **do**
  $S \leftarrow n\epsilon/(2+\epsilon)$ vertices of smallest degree in G
  remove S from G and place at the end of ordering
- Example ($\epsilon = 1$):

G



L

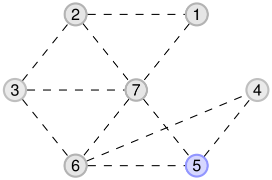


External-Memory Case

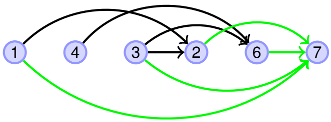
- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):

  while G is nonempty **do**
  $S \leftarrow n\epsilon/(2+\epsilon)$ vertices of smallest degree in G
  remove S from G and place at the end of ordering
- Example ($\epsilon = 1$):

G



L

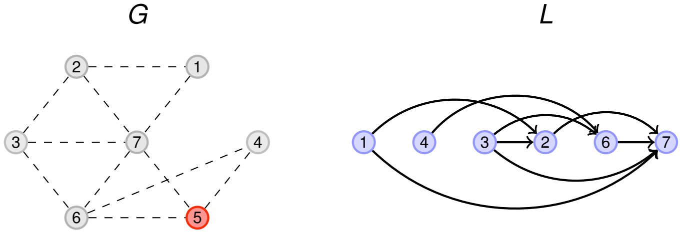


External-Memory Case

- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):

  while G is nonempty **do**
  $S \leftarrow n\epsilon/(2+\epsilon)$ vertices of smallest degree in G
  remove S from G and place at the end of ordering

- Example ($\epsilon = 1$):

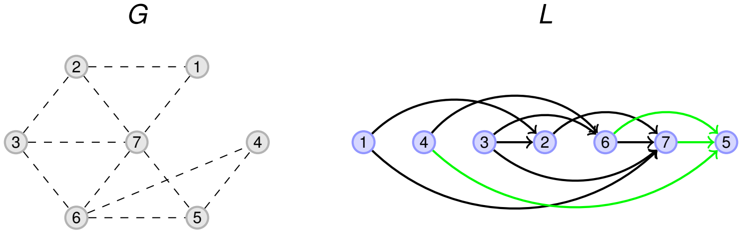


External-Memory Case

- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):

  while G is nonempty **do**
  $S \leftarrow n\epsilon/(2+\epsilon)$ vertices of smallest degree in G
  remove S from G and place at the end of ordering

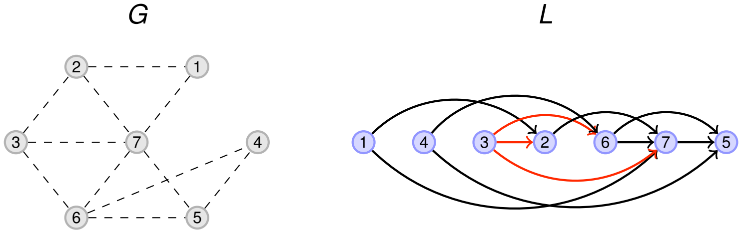
- Example ($\epsilon = 1$):



External-Memory Case

- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):

  while G is nonempty **do**
  $S \leftarrow n\epsilon/(2+\epsilon)$ vertices of smallest degree in G
  remove S from G and place at the end of ordering
- Example ($\epsilon = 1$):



External-Memory Case: Analysis

- Computes a $(2+\epsilon)d$-degeneracy ordering of a d-degenerate graph
- Does not require prior knowledge of d
- $O(\lg n)$ `while` iterations
- $O(sort(dn))$ overall I/O complexity

Cycles of Given Length

- Problem: find a cycle of length c in graph G
- *NP*-complete in general case, feasible for small c
- Our algorithm – external-memory adaptation of a sequential algorithm [Alon *et al. (2009)]*

All Maximal Cliques

- Problem: given graph G, list all its maximal cliques
- Basic version of the algorithm [Bron and Kerbosch (1973)]: recursive search maintaining the following sets:
  - R – current clique (possibly non-maximal)
  - P – vertices to be considered for adding to clique
  - X – forbidden vertices (not to be added to clique)
- Further improved [Tomita *et al. (2006)]*

Algorithm

- [Eppstein *et al. (2010)]*: Run the improved version of the algorithm with initial values:
  - R – vertex v
  - P – later neighbors of v in degeneracy ordering
  - X – earlier neighbors of v in degeneracy ordering

  (for every v)

Algorithm

- [Eppstein *et al. (2010)*]: Run the improved version of the algorithm with initial values:
    - R – vertex v
    - P – later neighbors of v in degeneracy ordering
    - X – earlier neighbors of v in degeneracy ordering

  (for every v)
- Runs in $O(3^{d/3}dn)$ time

Algorithm

- [Eppstein *et al. (2010)*]: Run the improved version of the algorithm with initial values:
  - R – vertex v
  - P – later neighbors of v in degeneracy ordering
  - X – earlier neighbors of v in degeneracy ordering

  (for every v)
- Runs in $O(3^{d/3}dn)$ time
- Our external-memory version of this algorithm: $O(3^{\delta/3} sort(\delta n))$ I/O's ($\delta = (2+\epsilon)d$)

Summary

- Approximate degeneracy ordering can be efficiently computed even for huge graphs by external-memory algorithm

Summary

- Approximate degeneracy ordering can be efficiently computed even for huge graphs by external-memory algorithm
- Existing sequential algorithms utilizing degeneracy ordering can be adapted into the external-memory model

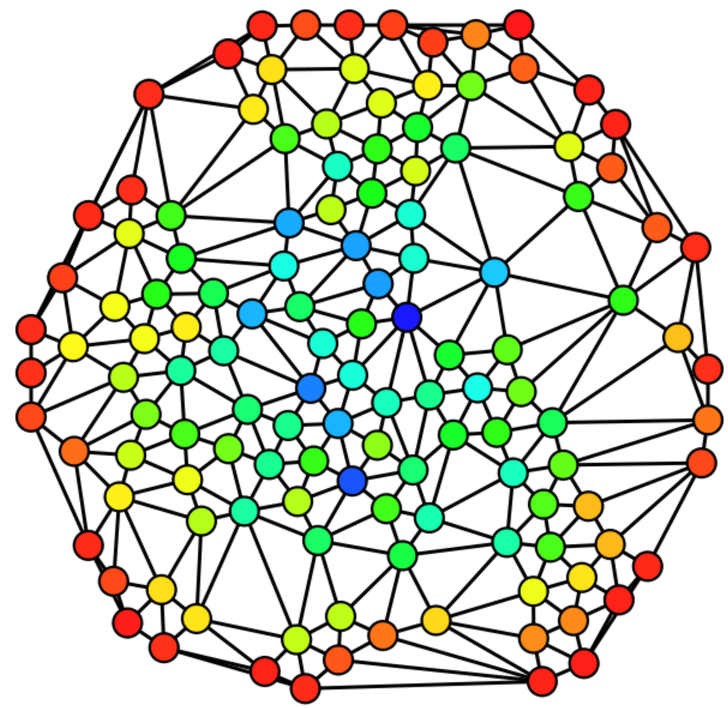
Betweenness Centrality

The betweenes of a vertex v in a graph G: = (V,E) with V vertices is defined as follows:

1. For each pair of vertices (s,t), consider the shortest paths between them.

2. For each pair of vertices (s,t), determine the fraction of shortest paths that pass through the vertex in question (here, vertex v).

3. Sum this fraction over all pairs of vertices (s,t).



Thank you!