External-Memory Network Analysis Algorithms for Naturally Sparse Graphs

Michael T. Goodrich
joint w/ Pawel Pszona
Dept. of Computer Science
The Memory Hierarchy

- The trade-off of size and speed

1 TB

4 GB

Capacity

Disk

Memory

L3 cache

L2 cache

L1 cache

core

Latency

10 ms

100 ns

5 ns

0.5 ns
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:
  - $\text{scan}(N) = \theta\left(\frac{N}{DB}\right)$
  - $\text{sort}(N) = \theta\left(\frac{N}{DB} \log_{M/B} \frac{N}{B}\right)$
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*. 

*Planar graphs* and *random graphs* (Pittel et al. 1996, Riordan 2008) can be naturally sparse. *Generated graphs* (as in Kleinberg et al.) and *real-world graphs* (Eppstein and Strash) can also be naturally sparse.
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:
  
  ```plaintext
  while G is nonempty do
    ν ← vertex of smallest degree in G
    remove ν from G and place it at the end of ordering
  ```
Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:
  
  ```
  while G is nonempty do
  v ← 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:
  
  \[
  \text{while } G \text{ is nonempty do }
  \]
  
  \[
  v \leftarrow \text{vertex of smallest degree in } G
  \]
  
  remove \( v \) from \( G \) and place it at the end of ordering

- Example:

\[
G \quad L
\]
Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:
  
  ```
  while G is nonempty do
    v ← vertex of smallest degree in G
    remove v from G and place it at the end of ordering
  ``

- Example:

```text
G
```

```
L
```

```text
2
3
6

4
5
7

1
1
```
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
```

![Graph example](image)
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:

  ![Graph](image)

  $G$

  $L$

  $1 \rightarrow 2$
Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:
  
  ```
  while G is nonempty do
      v ← vertex of smallest degree in G
      remove v from G and place it at the end of ordering
  ```
- Example:

```
G

1  2  3  4  5  6  7
```

```
L

1  2
```
Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:
  
  ```
  while G is nonempty do
  \( v \leftarrow \text{vertex of smallest degree in } G \)
  remove \( v \) from \( G \) and place it at the end of ordering
  ```

- Example:

  ![Graph](image)

  ```
  \begin{align*}
  G &\quad & L \\
  1 &\quad & 2 \\
  3 &\quad & 7 \\
  6 &\quad & 5 \\
  \end{align*}
  ```
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:
  ```

  ![Graph](image)

  - $G$
  - $L$
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 ← vertex of smallest degree in G
    remove v from G and place it at the end of ordering
  ```
- Example:

```
G

1 2 3 4 5 6 7

L

1 → 2 → 3 → 4
```
Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:
  
  ```
  while G is nonempty do
    v ← 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
    \( \nu \leftarrow \) vertex of smallest degree in \( G \)
    remove \( \nu \) from \( G \) and place it at the end of ordering
  ```
- Example:

![Graph Diagram](image)

\( G \)  \( L \)
Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:
  \[
  \text{while } G \text{ is nonempty do} \]
  \[
  \begin{align*}
  v &\leftarrow \text{vertex of smallest degree in } G \\
  \text{remove } v \text{ from } G \text{ and place it at the end of ordering}
  \end{align*}
  \]
- Example:

\[
\begin{align*}
G & \\
L & \\
\end{align*}
\]
Sequential Case

- Degeneracy ordering easy to compute
- Algorithm:
  
  ```
  while G is nonempty do
    v ← 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 ← vertex of smallest degree in G
      remove v from G and place it at the end of ordering
  ```

- Example:

```
G

1 ——— 2 ——— 3 ——— 4 ——— 5
    |      |      |
    |      |      |
    6 ——— 7 ——— 1
```

```
L

1 ——— 2 ——— 3 ——— 4 ——— 5 ——— 6 ——— 7
```

Diagram:

- Vertices marked with numbers are vertices in the ordering $L$.
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:

  ![Diagram](image)
External-Memory Case

- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):
  
  ```
  while $G$ is nonempty do
    $S \leftarrow \frac{n\epsilon}{2 + \epsilon}$ 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 ($\epsilon > 0$):
  
  \[
  \textbf{while} \ G \text{ is nonempty} \ \textbf{do} \\
  \quad S \leftarrow n\epsilon/(2 + \epsilon) \text{ vertices of smallest degree in } G \\
  \quad \text{remove } S \text{ from } G \text{ and place at the end of ordering}
  \]

- Example ($\epsilon = 1$):
  
  \[
  G \\
  \begin{array}{c}
  2 \\
  3 \\
  6 \\
  \end{array} \\
  \begin{array}{c}
  1 \\
  7 \\
  5 \\
  \end{array} \\
  \begin{array}{c}
  4 \\
  \end{array} \\
  L
  \]
External-Memory Case

- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):
  
  \[
  \text{while } G \text{ is nonempty do} \\
  \quad S \leftarrow n\epsilon/(2 + \epsilon) \text{ vertices of smallest degree in } G \\
  \quad \text{remove } S \text{ from } G \text{ and place at the end of ordering}
  \]
- Example ($\epsilon = 1$):
  
  $G$

\[
\begin{array}{c}
  2 \\
  3 \\
  6 \\
  7 \\
  5 \\
  4 \\
  1 \\
\end{array}
\]

$L$
External-Memory Case

- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):
  
  ```
  while G is nonempty do
    S ← $n\epsilon/(2 + \epsilon)$ vertices of smallest degree in G
    remove S from G and place at the end of ordering
  ```

- Example ($\epsilon = 1$):
  
  $$G \rightarrow L$$
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 ← $n\epsilon / (2 + \epsilon)$ 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 ($\epsilon > 0$):
  
  \[
  \text{while } G \text{ is nonempty do}
  \]
  
  \[
  S \leftarrow n\epsilon/(2 + \epsilon) \text{ vertices of smallest degree in } G
  \]
  
  remove $S$ from $G$ and place at the end of ordering

- Example ($\epsilon = 1$):

```
G

2 - 1
3 - 7
4

6
```

```
L

1 4
3
2
6
```
External-Memory Case

- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):
  
  ```
  while $G$ is nonempty do
  S ← $n\epsilon/(2 + \epsilon)$ 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 ($\epsilon = 1$):

![Diagram](image_url)
Not suitable for the external-memory case

Our solution: approximate degeneracy ordering

Algorithm ($\epsilon > 0$):

\begin{itemize}
    \item \textbf{while} $G$ is nonempty \textbf{do}
    \item \hspace{1em} $S \leftarrow \frac{n\epsilon}{(2 + \epsilon)}$ vertices of smallest degree in $G$
    \item \hspace{1em} remove $S$ from $G$ and place at the end of ordering
\end{itemize}

Example ($\epsilon = 1$):

\begin{itemize}
    \item $G$
    \item $L$
\end{itemize}
External-Memory Case

- Not suitable for the external-memory case
- Our solution: approximate degeneracy ordering
- Algorithm ($\epsilon > 0$):
  
  
  \[
  \text{while } G \text{ is nonempty do}
  \]
  
  \[
  S \leftarrow n\epsilon/(2 + \epsilon) \text{ vertices of smallest degree in } G
  \]
  
  remove $S$ from $G$ and place at the end of ordering

- Example ($\epsilon = 1$):
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
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)]
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 ($\delta = (2 + \epsilon)d$)
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 betweenness 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!