Densest Subgraph: Supermodularity, Iterative Peeling and Flow

Chandra Chekuri
Univ. of Illinois, Urbana-Champaign

Based on joint work with Kent Quanrud and Manuel Torres

Northwestern, May 4, 2022
Densest Subgraph (DSG)

\( G = (V, E) \) undirected graph

Find “dense” subgraph(s)

\[
density(S) = \frac{|E(S)|}{|S|}
\]

\[
\lambda^* = \max_{S \subseteq V} \frac{|E(S)|}{|S|}
\]
Example

\[ \lambda^* = \frac{6}{4} \]
Dense Subgraph Discovery

\[ \text{density}(S) = \frac{f(S)}{|S|} \]

- Triangle density: \( f(S) = \# \) of triangles in \( G[S] \) [Tsourakakis’14]
- k-clique density: \( f(S) = \# \) of k-cliques in \( G[S] \) [Tsourakakis’15]
- Hypergraphs: \( f(S) = \# \) of hyperedges in \( G[S] \) [folklore?]
- p-mean density: \( f(S) = \sum_{v \in S} \deg(v, S)^p \) [Benson-Kleinberg-Veldt’21]
- Constrained versions: [many authors]
  \[ \max f(S) \text{ s.t } |S| = k, |S| \leq k, |S| \geq k \]
- Directed graph version: [Kannan-Vinay’99, Charikar’00]
Polynomial Solvability

DSG is poly-time solvable

- Reduction to flow [Picard-Queyranne’82, Goldberg’84]
- Reduction to submodular function minimization [folklore]
- LP relaxation [Charikar’00]
Sub and Supermodularity

Real-valued set function $f: 2^V \rightarrow R$ is **submodular** if

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B) \quad \forall A, B$$

Equivalently:

$$f(A + v) - f(A) \geq f(B + v) - f(B) \quad A \subseteq B, v \notin B$$
Sub and Supermodularity

\[ f : 2^V \rightarrow R \text{ is supermodular} \iff -f \text{ is submodular} \]

\[ f(A) + f(B) \leq f(A \cap B) + f(A \cup B) \quad \forall A, B \]

Notation: \( f(v | S) = f(S + v) - f(S) \) marginal value

Supermodular:

\[ f(v | B) \geq f(v | A) \quad A \subseteq B, v \in B - A \]
Sub and Supermodularity

Given graph $G = (V, E)$

- $f(S) = |\delta(S)|$ is submodular and non-negative.
- $f(S) = |E(S)| = \frac{1}{2} \left( \sum_v \deg(v) - |\delta(S)| \right)$ is supermodular, non-negative and monotone.
Densest Supermodular Set (DSS)

Given supermodular $f : 2^V \rightarrow R_+$ find $\max_S \frac{f(S)}{|S|}$

Decision version: check if $\exists S \text{ s.t. } \frac{f(S)}{|S|} \geq \lambda$

Check if $\exists S \text{ s.t. } \lambda|S| - f(S) \leq 0$

Poly-time via submodular function minimization
Some Recent Directions on Densest Subgraph Discovery

- Fast *approximate* algorithms for *(very) large* graphs
- Variations in objective and applications
- Streaming *(approximate)* algorithms
- Parallel *(approximate)* algorithms
- Dynamic *(approximate)* algorithms
- ...

Motivation

- Faster approximations for mixed packing and covering LPs (DSG is a special case)
- Connections to supermodularity
- Discrete + continuous
Results at high-level

- Fast approximate algorithm: \((1 - \varepsilon)\) approximation for densest subgraph in \(O\left(\frac{m \cdot \text{polylog}(n)}{\varepsilon}\right)\) time

- Affirmative answer to conjecture of [Boob et al]

- Generalization to supermodular functions

- Other results …

*Mainly about connections which are simple in retrospect*
Rest of the talk

- Charikar’s LP Relaxation
- Flow based approximation algorithm
- Peeling and Iterative Peeling
- Relating iterative peeling to LP solving via MWU
Charikar’s LP Relaxation

\[ \text{max} \sum_{uv \in E} x_{uv} \]
\[ \sum_v z_v = 1 \]
\[ x_{uv} \leq \min(z_u, z_v) \quad uv \in E \]
\[ x, z \geq 0 \]

\[ \text{max} \sum_{uv \in E} x_{uv} \]
\[ \sum_v z_v = 1 \]
\[ x_{uv} \leq \min(z_u, z_v) \quad uv \in E \]
\[ x, z \geq 0 \]

**Theorem:** [Charikar’00] LP is optimal for DSG
# Charikar’s LP Relaxation

<table>
<thead>
<tr>
<th>Primal</th>
<th>Dual</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \max \sum_{uv \in E} x_{uv} ]</td>
<td>[ \min D ]</td>
</tr>
<tr>
<td>[ \sum_v z_v = 1 ]</td>
<td>[ \gamma_{uv,u} + \gamma_{uv,v} \geq 1 \quad uv \in E ]</td>
</tr>
<tr>
<td>[ x_{uv} \leq \min(z_u, z_v) \quad uv \in E ]</td>
<td>[ \sum_{uv \in E} \gamma_{uv,v} \leq D \quad v \in V ]</td>
</tr>
<tr>
<td>[ x, z \geq 0 ]</td>
<td>[ y \geq 0 ]</td>
</tr>
</tbody>
</table>

**Theorem:** [Charikar’00] LP is optimal for DSG
Solving LP Approximately

• Dual-LP is a *mixed packing and covering* LP

• Can obtain \((1 - \epsilon)\) approx. in \(O \left( \frac{m \cdot \text{polylog}(n)}{\epsilon^2} \right)\) time, even in parallel [Bahmani-Goel-Munagala’14]

• **Open question:** can we solve mixed packing and covering LPs in \(O \left( \frac{N \cdot \text{polylog}(n)}{\epsilon} \right)\) time? Known for pure packing and covering [AllenZhu-Orecchia’14, Wang-Rao-Mahoney’15]

• \(O \left( \frac{m \Delta \cdot \text{polylog}(n)}{\epsilon} \right)\) time for DSG [Boob-Sawlanini-Wang’19]
Flow Reduction via Dual

Claim: Max-flow in $H_\lambda = |E|$ iff $\lambda \geq \lambda^*$

Observed in [Boob et al]

\[
\begin{align*}
\min D \\
y_{uv,u} + y_{uv,v} & \geq 1 \quad uv \in E \\
\sum_{uv \in E} y_{uv,v} & \leq D \quad v \in V \\
y & \geq 0
\end{align*}
\]

Fractional perfect matching

Flow network $H_\lambda$
Flow based Approx Algorithm

Given value $\lambda$.

1. Construct $H_\lambda$

2. Run augmenting path algorithm: stop if shortest augmenting path length $\geq c \log n / \epsilon$

**Theorem:** If maxflow not reached then there exists subgraph in $G$ with density $\geq (1 - \epsilon)\lambda$
Flow based Approx Algorithm

**Theorem:** $(1 - \epsilon)$ approximation for DSG in $O\left(m \frac{\text{polylog}(n)}{\epsilon}\right)$ time

- Generalizes to hypergraphs
- Also yields faster approximation algorithm for densest directed subgraph via reduction
Peeling Algorithm

[Asahiro et al. 00, Charikar 00]

- For \( i = 1 \) to \( n \) do
  - \( v_i \) is in min-degree vertex in \( G \)
  - \( G \leftarrow G - v_i \)

- \( v_1, v_2, \ldots, v_n \) is ordering created by algorithm
- \( S_i \leftarrow \{v_i, v_{i+1}, \ldots, v_n\} \)
- Output \( \text{argmax}_i \frac{|E(S_i)|}{|S_i|} \)

**Theorem:** [Charikar’00] Greedy peeling a \( \frac{1}{2} \) approximation for DSG (proof via LP)
(Tight) Example

\[ K_{d+1} \]

\[ \lambda^* \approx d \text{ via } K_{d,D} \]

\[ \lambda(G) \approx \frac{d}{2} \]
(Tight) Example

\[ K_{d+1} \]

\[ K_{d,D} \]

\[ D \gg d \]

\[ \lambda^* \approx d \text{ via } K_{d,D} \]

\[ \lambda(G) \approx \frac{d}{2} \]

Peeling order
Given supermodular function $f: 2^V \rightarrow R_+$

- For $i = 1$ to $n$ do
  - $v_i \leftarrow \text{argmin}_v f(v | V - v)$
  - $V \leftarrow V - v_i$
  - Restrict $f$ to $V - v_i$

- $v_1, v_2, \ldots, v_n$ is ordering created by algorithm
- $S_i \leftarrow \{v_i, v_{i+1}, \ldots, v_n\}$
- Output $\text{argmax}_i \frac{|f(S_i)|}{|S_i|}$
**Peeling and DSS**

**Question:** How can we characterize for general $f$?

$$c_f = \max_S \frac{\sum_{v \in S} f(v | S - v)}{f(S)}$$

Supermodularity: $\sum_{v \in S} f(v | S - v) \geq f(S) \Rightarrow c_f \geq 1$
Peeling and DSS

\[ c_f = \max_S \frac{\sum_{v \in S} f(v | S - v)}{f(S)} \]

**Theorem:** Peeling is a \( \frac{1}{c_f} \) approximation for DSS

Proof is a simple adaptation of the combinatorial proof of [Khuller-Saha’09]

Can also do it via relaxation ala [Charikar’00]
Theorem: Peeling is a $\frac{1}{c_f}$ approximation for DSS

- Graphs: $c_f = \max_S \frac{\sum_{v \in S} \deg(v,S)}{|E(S)|} = 2$
- Hypergraphs: $c_f = r$ where $r$ is rank
- $p$-th mean in graphs: $c_f = p + 1$
Iterative Peeling

[BGPSTWW’20]

- Heuristic inspired by Dual-LP and MWU
- Goal: improve $\frac{1}{2}$ approx to $(1 - \epsilon)$ approx.
- Peel several times by adjusting ”load”
- Creates a new ordering in each iteration
- Pick best suffix among all orderings
Iterative Peeling

[BGPSTWW’20]

**Greedy++**
- \( \text{load}(v, 0) = 0 \) for all \( v \)
- For \( t = 1 \) to \( T \) do
  - \( G' \leftarrow G \)
  - For \( i = 1 \) to \( n \) do
    - \( v_{t,i} \leftarrow \text{argmin}_v \text{deg}(v) + \text{load}(v, t - 1) \)
    - \( \text{load}(v_{t,i}, t) = \text{load}(v_{t,i}, t - 1) + \text{deg}(v_{t,i}) \)
    - \( G' \leftarrow G' - v_{i,t} \)
- \( S_{t,i} \leftarrow \{v_{t,i}, \ldots, v_{t,n}\} \)
- Output \( \text{argmax}_{i,t} \frac{|E(S_{t,i})|}{|S_{t,i}|} \)
Example

\[ K_{d,D} \quad D \gg d \]

\[ \lambda^* \approx d \]

Peeling: \( \lambda \approx \frac{d}{2} \)
Example

$K_d, f \gg d$  

$K_{d+1}$  

$\lambda^* \approx d$  

Peeling: $\lambda \approx \frac{d}{2}$
[BGPSTWW’20]

**Conjecture:** Greedy++ is a \((1 - \epsilon)\) approximation after \(O\left(\frac{1}{\epsilon^2}\right)\) iterations for DSG

Seems to work very well in practice. Implementation runs very fast even on large graphs and converges quickly on many real-world graphs.
Iterative Peeling for DSS

Given supermodular $f : 2^V \rightarrow \mathbb{R}_+$ find $\max_S \frac{f(S)}{|S|}$

**SuperGreedy++**
- $\text{load}(v, 0) = 0$ for all $v$
- For $t = 1$ to $T$ do
  - $S_{t,0} \leftarrow V$
  - For $i = 1$ to $n$ do
    - $v_{t,i} \leftarrow \arg\min_{v \in S_{t,i}} f(v | S_{t,i} - v) + \text{load}(v, t - 1)$
    - $\text{load}(v_{t,i}, t) = \text{load}(v_{t,i}, t - 1) + f(v_{t,i} | S_{t,i} - v_{t,i})$
    - $S_{t,i+1} \leftarrow S_{t,i} - v_{t,i}$
- Output $\arg\max_{t,i} \frac{|f(S_{t,i})|}{|S_{t,i}|}$
Iterative Peeling for DSS

**Theorem:** SuperGreedy++ converges to a \((1 - \epsilon)\) approximation in \(O\left(\frac{1}{\epsilon^2} \frac{\max f(v)}{\lambda^*} \log n\right)\) iterations

**Corollary:** Greedy++ converges to a \((1 - \epsilon)\) approximation for DSG in \(O\left(\frac{1}{\epsilon^2} \frac{\Delta(G)}{\lambda^*} \log n\right)\) iterations
Proof Idea

- Express DSS as an LP relaxation
- Relate SuperGreedy++ iterations to a multiplicative-weight update (MWU) algorithm via LP
Proof Idea

• Express DSS as an LP relaxation
  • Generalize Charikar’s LP for DSG via Lovasz-extension of supermodular/submodular functions
  • Rewrite as LP via an ordering based view of Lovasz-extension

• Relate SuperGreedy++ iterations to a multiplicative-weight update (MWU) algorithm via LP
  • SuperGreedy++ iterations are not MWU iterations but can show approximate relationship
Charikar’s LP

**LP**

\[
\begin{align*}
\text{max} \sum_{uv \in E} x_{uv} \\
\sum_v z_v &= 1 \\
x_{uv} &\leq \min(z_u, z_v) \quad uv \in E \\
x, z &\geq 0
\end{align*}
\]

**Concave Program**

\[
\begin{align*}
\text{max} \sum_{uv \in E} \min(z_u, z_v) \\
\sum_v z_v &= 1 \\
z &\geq 0
\end{align*}
\]
Lovasz Extension

\[ f: 2^V \rightarrow \mathbb{R} \] real valued set function

want to extend to continuous function \( f: [0,1]^V \rightarrow \mathbb{R} \)

Example: \( V = \{v_1, v_2, v_3, v_4\} \) What is \( f(0.3, 0.7, 0, 0.1) \)?

Sort according to decreasing \( x \) values: \( v_2, v_1, v_4, v_3 \)

\[
\begin{align*}
f(0.3, 0.7, 0, 0.1) &= x_2 f(v_2 \mid \emptyset) + x_1 f(v_1 \mid \{v_2\}) + x_4 f(v_4 \mid \{v_2, v_1\}) \\
&\quad + x_3 f(v_3 \mid \{v_2, v_1, v_4\})
\end{align*}
\]
Lovasz Extension

\( f : 2^V \to \mathbb{R} \) real valued set function

A rounding interpretation:

\[
\hat{f}(x) = \mathbf{Ex}_{\theta \sim [0,1]} [f(x^\theta)]
\]

where \( x^\theta = \{ v | x_v \geq \theta \} \)

**Theorem:** [Lovasz] \( \hat{f} \) is convex iff \( f \) is submodular. \( \hat{f} \) is concave iff \( f \) is supermodular.
Convex Relaxation for DSS

Supermodular func: \( f: 2^V \to R_+ \). Want \( \max_S \frac{f(S)}{|S|} \)

\[
\max \hat{f}(z) \\
\sum_v z_v = 1 \\
z \geq 0
\]

Example: \( G = (V, E), f(S) = |E(S)| \)
\[
\hat{f}(x) = \sum_{uv \in E} \min(x_u, x_v)
\]
Convex Relaxation for DSS

Supermodular func: \( f: 2^V \to R_+ \). Want \( \max_S \frac{f(S)}{|S|} \)

\[
\begin{align*}
\max \hat{f}(z) \\
\sum_v z_v &= 1 \\
z &\geq 0
\end{align*}
\]

**Theorem:** Relaxation is exact for DSS
Supermodular func: \( f : 2^V \rightarrow R_+ \)

Consider all orderings/permutations of \( V \)

Given an ordering \( \sigma \) define a vector

\[
q(\sigma) \in R^V \text{ where } q_v(\sigma) = f(v \mid \{w \mid w <_\sigma v\})
\]

Example: \( \sigma = v_2, v_4, v_3, v_1 \)

\[
q_{v_3}(\sigma) = f(v_2, v_4, v_3) - f(v_2, v_4)
\]
Supermodular func: \( f : 2^V \to R_+ \)

Consider all orderings/permutations of \( V \)

Given an ordering \( \sigma \) define a vector

\[
q(\sigma) \in R^V \text{ where } q_v(\sigma) = f(v | \{w \mid w <_\sigma v\})
\]

Fact: \( \hat{f}(x) = \min_\sigma x^T q(\sigma) \).

Given \( x \), the optimum ordering \( \sigma_x \) is to sort coordinates of \( x \) in decreasing order of \( x_v \).
Rewriting Relaxations

\[
\begin{align*}
\text{max } \hat{f}(z) \\
\sum_{\nu} z_{\nu} &= 1 \\
z &\geq 0
\end{align*}
\]

\[OPT \text{ val } = \lambda^*\]

\[
\begin{align*}
\text{min } \sum_{\nu} z_{\nu} \\
\hat{f}(z) &\geq 1 \\
z &\geq 0
\end{align*}
\]

\[OPT \text{ val } = 1/\lambda^*\]
Rewriting Relaxations

\[
\min \sum_{\nu} z_\nu \\
\hat{f}(z) \geq 1 \\
z \geq 0
\]

\[
\text{OPT val} = \frac{1}{\lambda^*}
\]

\[
\min \sum_{\nu} z_\nu \\
z^T q(\sigma) \geq 1 \quad \text{for all } \sigma \\
z \geq 0
\]

\[
\text{OPT val} = \frac{1}{\lambda^*}
\]

Exponential sized LP
Rewriting Relaxations

\[
\min \sum_{\nu} z_{\nu} \\
\text{subject to:} \\
z^{T} q(\sigma) \geq 1 \text{ for all } \sigma \\
z \geq 0
\]

\[
\max \sum_{\sigma} y_{\sigma} \\
\text{subject to:} \\
\sum_{\sigma} q_{\nu}(\sigma) y_{\sigma} \leq 1 \text{ for all } \nu \in V \\
y \geq 0
\]

OPT val = $1/\lambda^*$

Dual LP

Exponential sized LP
Ordering LP Relaxation

\[
\begin{align*}
\max & \sum_{\sigma} y_{\sigma} \\
\sum_{\sigma} q_v(\sigma) y_{\sigma} & \leq 1 \quad \text{for all } v \in V \\
y & \geq 0
\end{align*}
\]

- Packing LP
- Exponential # of variables but only \( n \) non-trivial constraints
- Amenable to MWU techniques
Solving Ordering LP via Multiplicative Weight Updates

• MWU: iterative algorithm for solving LPs
• Maintain (exponential) weights on constraints (dual variables)
• In each iteration solve a Lagrangean relaxation and take a small step along solution
$f: 2^V \to R$ is supermodular

For ordering $\sigma$ of $V$, $q(\sigma)$ is a vector where

$q_v(\sigma) = f(v | \{u | u <_\sigma v \})$

$max \sum_{\sigma} y_\sigma$

$\sum_{\sigma} q_v(\sigma) y_\sigma \leq 1$ for all $v \in V$

$y \geq 0$

1. $y^0 = 0$
2. $load^0(v) = 1$ for all $v$
3. $\eta = \frac{1}{\epsilon} \log n$
4. For $t = 1$ to $T$ do
   - $\sigma_t = \text{argmin}_\sigma \langle load^{t-1}, q(\sigma) \rangle$
   - $y^t = y^{t-1} + \frac{1}{\lambda^* T} 1_{\sigma_t}$
   - For each $v$ set $load^t(v) \leftarrow \exp(\eta \sum_{\sigma} y^t_\sigma q_v(\sigma))$
5. Output $y^T = \frac{1}{\lambda^* T} \sum_t 1_{\sigma_t}$
\[ f : 2^V \to R \text{ is supermodular} \]

For ordering \( \sigma \) of \( V \), \( q(\sigma) \) is a vector where
\[ q_v(\sigma) = f(v \mid \{u \mid u <_\sigma v\}) \]

\[ \max \sum_{\sigma} y_\sigma \]
\[ \sum_{\sigma} q_v(\sigma) y_\sigma \leq 1 \quad \text{for all } v \in V \]
\[ y \geq 0 \]

1. \( y^0 = 0 \)
2. \( \text{load}^0(v) = 1 \) for all \( v \)
3. \( \eta = \frac{1}{\epsilon} \log n \)
4. For \( t = 1 \) to \( T \) do
   - \( \sigma_t = \arg\max_{\sigma} \langle \text{load}^{t-1}, q(\sigma) \rangle \)
   - \( y^t = y^{t-1} + \frac{1}{\lambda^* T} 1_{\sigma_t} \)
   - For each \( v \) set \( \text{load}^t(v) \leftarrow \exp(\eta \sum_{\sigma} y^t_\sigma q_v(\sigma)) \)
5. Output \( y^T = \frac{1}{\lambda^* T} \sum_{t} 1_{\sigma_t} \)

**MWU Analysis:** Algorithm outputs \((1 - \epsilon)\) approx if \( T = \Omega(\frac{\Delta}{\epsilon^2 \lambda^* \log n}) \)
Iterative Peeling and MWU

- MWU algorithm with LP naturally works with orderings of $V$ which we see in SuperGreedy++
- SuperGreedy++ is *not* implementing standard MWU algorithm
- Why?
  - For graphs, given $\text{load}(v)$ for each $v$
    - Output ordering according to decreasing order of loads
    - Static and does not add $\text{deg}(v)$ correction term
    - Hence in first iteration *any* ordering is ok for MWU
Iterative Peeling and MWU

- SuperGreedy++ is *not* implementing standard MWU algorithm

- **Technical Lemma:** For appropriate parameter setting, each iteration of SuperGreedy++ yields a \((1 + \epsilon)\) approximate ordering in MWU algorithm

- **Intuition:** \(\text{deg}\) is static while loads are increasing so initial Greedy step washes out eventually. Advantage of initial Greedy is its performance even after one iteration
Iterative Peeling and MWU

- SuperGreedy++ is *not* implementing standard MWU algorithm

- **Technical Lemma**: For appropriate parameter setting each iteration of SuperGreedy++ yields a $(1 + \epsilon)$ approximate ordering in MWU algorithm

- MWU analysis is robust to approximate oracle

- Putting together yields convergence analysis
Summary

• **Fast approximate algorithm:** $(1 - \epsilon)$ approximation for densest subgraph in $O\left( m \frac{\text{polylog}(n)}{\epsilon} \right)$ time. *Short augmenting paths suffice for density calculation*

• **SuperGreedy++:** simple iterative algorithm that converges for *any supermodular function*

• Other results in paper showcasing utility of supermodular perspective
Open Problems

- Tight analysis of iterative peeling
  - Worst example known to us: $\Omega\left(\frac{1}{\epsilon}\right)$ iterations for $(1 - \epsilon)$ approximation
  - Is dependence on $\Delta, n, \lambda^*$ necessary? What about for DSS?

- Improved sequential, dynamic and parallel algorithms for DSG, DSS, and variants
Thanks!