k-cores and Densest Subgraphs

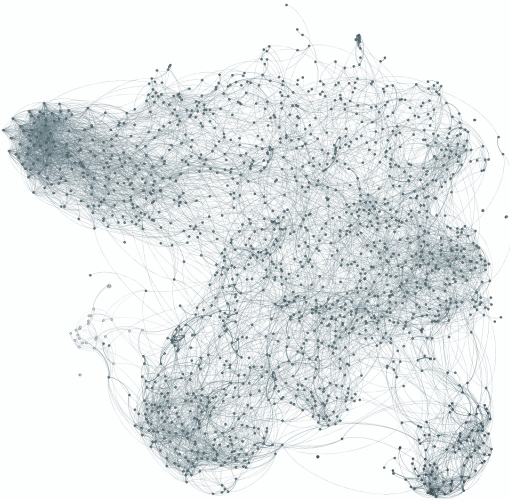
Mauro Sozio

Telecom ParisTech

November 15, 2018

Finding dense regions in a graph..

for community detection, spam detection, event detection...



k-cores

Definition (k-core)

Given a graph G and $k \geq 0$, a subgraph H of G is a k-core, if

- for every node $v \in V_H$, $\delta_H(v) \geq k$;
- $|V_H|$ is maximum;

k-cores

Definition (k-core)

Given a graph G and $k \geq 0$, a subgraph H of G is a k-core, if

- for every node $v \in V_H$, $\delta_H(v) \geq k$;
- $|V_H|$ is maximum;

A k-core can be computed in linear time in $|E_G|$ as follows.

While (at least one node has degree $< k$)

- remove all nodes with degree $< k$ from the current graph.

k-core decomposition

Note: a k-core might not be connected and is unique (possibly empty).

k-core decomposition

Note: a k-core might not be connected and is unique (possibly empty).

Note: If v belongs to a $k-$core then it belongs to a $\bar{k}-$core, $\bar{k} < k$.

k-core decomposition

Note: a k-core might not be connected and is unique (possibly empty).

Note: If v belongs to a $k-$core then it belongs to a $\bar{k}-$core, $\bar{k} < k$.

Definition (k-core decomposition)

A $k-$core decomposition specifies for each node v in G an integer k_v such that v is in the k_v-core and k_v is maximum.

k-core decomposition

Note: a k-core might not be connected and is unique (possibly empty).

Note: If v belongs to a $k-$core then it belongs to a $\bar{k}-$core, $\bar{k} < k$.

Definition (k-core decomposition)

A $k-$core decomposition specifies for each node v in G an integer k_v such that v is in the k_v-core and k_v is maximum.

Note: A $k-$core decomposition can be computed in linear time: a $\bar{k}$-core is obtained by iteratively removing all nodes with degree $< \bar{k}$, $\bar{k} = 1, \ldots, n$.

Graph: Definitions

Definition ((Undirected) Graph)

A graph G is a pair (V_G, E_G), where V_G is a set of *nodes*, while E_G is a set of *edges* (u, v) with $u, v \in V_G$.

Other important definitions:

Graph: Definitions

Definition ((Undirected) Graph)

A graph G is a pair (V_G, E_G), where V_G is a set of *nodes*, while E_G is a set of *edges* (u, v) with $u, v \in V_G$.

Other important definitions:

- A graph $H = (V_H, E_H)$ is a (induced) subgraph of $G = (V_G, E_G)$ if the following two conditions hold: $V_H \subseteq V_G$, moreover, $(u, v) \in E_H$ if and only if $u, v \in H$ and $(u, v) \in E_G$.

Graph: Definitions

Definition ((Undirected) Graph)

A graph G is a pair (V_G, E_G), where V_G is a set of *nodes*, while E_G is a set of *edges* (u, v) with $u, v \in V_G$.

Other important definitions:

- A graph $H = (V_H, E_H)$ is a (induced) subgraph of $G = (V_G, E_G)$ if the following two conditions hold: $V_H \subseteq V_G$, moreover, $(u, v) \in E_H$ if and only if $u, v \in H$ and $(u, v) \in E_G$.
- $\delta_G(v)$ denotes the number of edges incident to v in G, while $\delta_H(v)$ denotes the number of edges incident to v in H.

Density of a graph

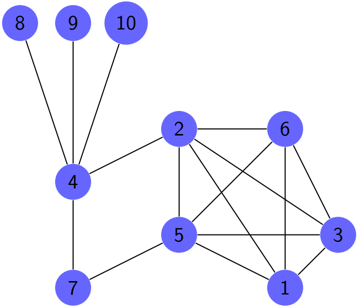
Definition (average degree density)

Given a graph $G = (E_G, V_G)$ its (average degree) density $\rho(G)$ is defined as $\rho(G) = \frac{|E_G|}{|V_G|}$.

Definition (clique density)

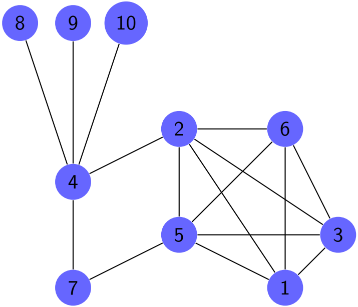
Given a graph $G = (E_G, V_G)$ its (clique) density $\phi(G)$ is defined as $\phi(G) = \frac{2 \cdot |E_G|}{|V_G| \cdot (|V_G| - 1)}$.

Example



$H = (\{4, 8, 9, 10\}, \{(4, 8)(4, 9)(4, 10)\})$
$\delta_G(4) = 5, \delta_H(4) = 3$

Example



$H = (\{4, 8, 9, 10\}, \{(4, 8)(4, 9)(4, 10)\})$
$\delta_G(4) = 5, \delta_H(4) = 3$
$\rho(G) = \frac{16}{10}, \rho(H) = \frac{3}{4}, \phi(H) = \frac{2 \cdot 3}{12}$

Simple lemma

Lemma

Given a graph $G = (V_G, E_G)$, we have:

$$\sum_{v \in V_G} \delta_G(v) = 2|E(G)|.$$

Proof.

Every edge $(u, v) \in E(G)$ is counted exactly twice in the summation: Once in $\delta_G(u)$ and the second time in $\delta_G(v)$. $\square$

Our main problem

Definition (Densest subgraph problem)

Given a graph $G = (V_G, E_G)$, find a subgraph H of G with maximum average degree density.

Facts: A global optimum can be computed in polynomial time. There is a linear-time algorithm that computes an approximation to the problem..

Densest Subgraph Algorithm

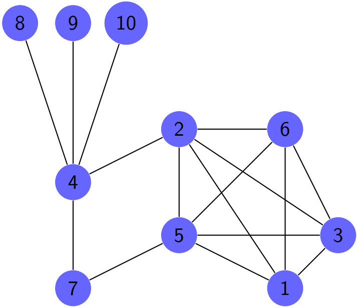
$H = G$;

while (G contains at least one edge)

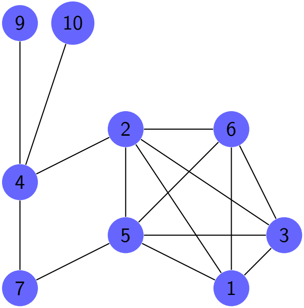
- let v be the node with minimum degree $\delta_G(v)$ in G;
- remove v and all its edges from G;
- if $\rho(G) > \rho(H)$ then $H \leftarrow G$;

return H;

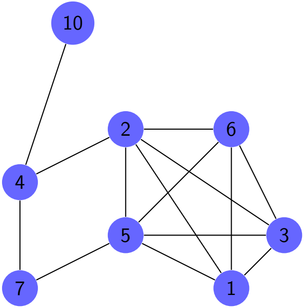
Densest Subgraph Algorithm:Example



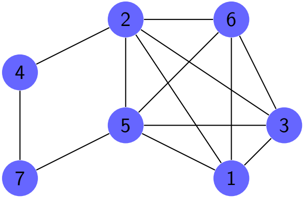
Densest Subgraph Algorithm:Example



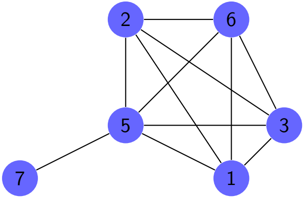
Densest Subgraph Algorithm:Example



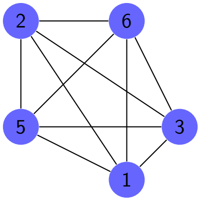
Densest Subgraph Algorithm:Example



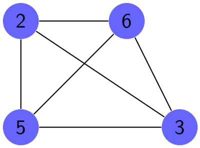
Densest Subgraph Algorithm:Example



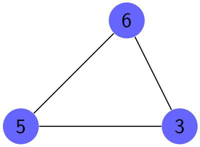
Densest Subgraph Algorithm:Example



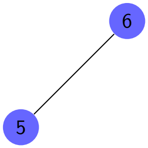
Densest Subgraph Algorithm:Example



Densest Subgraph Algorithm:Example

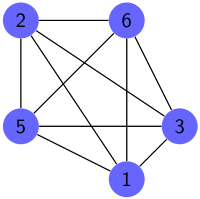


Densest Subgraph Algorithm:Example



Densest Subgraph Algorithm:Example

Densest Subgraph Algorithm:Example



Approximation Guarantee

Theorem

Let O be a densest subgraph in G. Our algorithm finds a subgraph H s.t.

$$\rho(H) \geq \frac{\rho(O)}{2}.$$

Approximation Guarantee

Lemma

Let O be a densest subgraph in G, then:

$$\forall v \in V_O \quad \delta_O(v) \geq \rho(O).$$

Proof.

We show that if there is v in O with $\delta_O(v) < \rho(O)$, then O is not densest.

$$\rho(O \setminus \{v\}) = \frac{|E_O| - \delta_O(v)}{|V_O| - 1}$$

□

Approximation Guarantee

Lemma

Let O be a densest subgraph in G, then:

$$\forall v \in V_O \quad \delta_O(v) \geq \rho(O).$$

Proof.

We show that if there is v in O with $\delta_O(v) < \rho(O)$, then O is not densest.

$$\begin{aligned}\rho(O \setminus \{v\}) &= \frac{|E_O| - \delta_O(v)}{|V_O| - 1} \\ &> \frac{|E_O| - \rho(O)}{|V_O| - 1}\end{aligned}$$

□

Approximation Guarantee

Lemma

Let O be a densest subgraph in G, then:

$$\forall v \in V_O \quad \delta_O(v) \geq \rho(O).$$

Proof.

We show that if there is v in O with $\delta_O(v) < \rho(O)$, then O is not densest.

$$\begin{aligned}
\rho(O \setminus \{v\}) &= \frac{|E_O| - \delta_O(v)}{|V_O| - 1} \\
&> \frac{|E_O| - \rho(O)}{|V_O| - 1} \\
&= \frac{|V_O|\rho(O) - \rho(O)}{|V_O| - 1} = \rho(O)\frac{|V_O| - 1}{|V_O| - 1} = \rho(O).
\end{aligned}$$

□

Approximation Guarantee

Theorem

Let G be any undirected graph. Let O be a densest subgraph of G, while let H be the subgraph computed by our algorithm with input G. Then,

$$\rho(H) \geq \frac{\rho(O)}{2}.$$

Proof of the approx. guarantee

Proof.

Consider the first step t when we remove $v \in V_O$ in the graph $G_t = (V_t, E_t)$. From the previous lemma $\delta_{G_t}(v) \geq \delta_O(v) \geq \rho(O)$. From this and the fact that v has minimum degree in G_t, it follows that:

$$\rho(H) \geq \rho(G_t)$$

Proof of the approx. guarantee

Proof.

Consider the first step t when we remove $v \in V_O$ in the graph $G_t = (V_t, E_t)$. From the previous lemma $\delta_{G_t}(v) \geq \delta_O(v) \geq \rho(O)$. From this and the fact that v has minimum degree in G_t, it follows that:

$$\begin{aligned} \rho(H) &\geq \rho(G_t) \\ &= \frac{|E_t|}{|V_t|} \end{aligned}$$

Proof of the approx. guarantee

Proof.

Consider the first step t when we remove $v \in V_O$ in the graph $G_t = (V_t, E_t)$. From the previous lemma $\delta_{G_t}(v) \geq \delta_O(v) \geq \rho(O)$. From this and the fact that v has minimum degree in G_t, it follows that:

$$\begin{aligned}\rho(H) &\geq \rho(G_t) \\ &= \frac{|E_t|}{|V_t|} \\ &= \frac{\frac{1}{2} \cdot \sum_{v \in V_t} \delta_{G_t}(v)}{|V_t|}\end{aligned}$$

Proof of the approx. guarantee

Proof.

Consider the first step t when we remove $v \in V_O$ in the graph $G_t = (V_t, E_t)$. From the previous lemma $\delta_{G_t}(v) \geq \delta_O(v) \geq \rho(O)$. From this and the fact that v has minimum degree in G_t, it follows that:

$$\begin{aligned}
\rho(H) &\geq \rho(G_t) \\
&= \frac{|E_t|}{|V_t|} \\
&= \frac{\frac{1}{2} \cdot \sum_{v \in V_t} \delta_{G_t}(v)}{|V_t|} \\
&\geq \frac{1}{2} \cdot \frac{|V_t| \rho(O)}{|V_t|}
\end{aligned}$$

Proof of the approx. guarantee

Proof.

Consider the first step t when we remove $v \in V_O$ in the graph $G_t = (V_t, E_t)$. From the previous lemma $\delta_{G_t}(v) \geq \delta_O(v) \geq \rho(O)$. From this and the fact that v has minimum degree in G_t, it follows that:

$$\begin{aligned}
\rho(H) &\geq \rho(G_t) \\
&= \frac{|E_t|}{|V_t|} \\
&= \frac{\frac{1}{2} \cdot \sum_{v \in V_t} \delta_{G_t}(v)}{|V_t|} \\
&\geq \frac{1}{2} \cdot \frac{|V_t| \rho(O)}{|V_t|} \\
&= \frac{\rho(O)}{2}.
\end{aligned}$$

Running time

The algo. can be implemented in linear time in the size of the input graph (i.e. the total number of edges and nodes in the graph).

- for each value δ in $[1, n]$ maintain a list of nodes with degree δ in the current graph.
- As nodes are removed from the graph, update the lists so that each node is placed in the correct list (depending on its current degree).

Exercise: which data structures to have a linear-time algorithm?

Computing the Densest Subgraph

Three main techniques:

- maximum flow [3];
- based on linear programming (LP) [2].
- one recent technique based on convex programming [5].

Introduction to Linear Programming (Example 1) [4]

A chocolate shop sells two products: Choco (1€ per box) and Fancy Choco (6€ per box). It makes x_1 boxes of Choco and x_2 boxes of Fancy Choco, daily. We wish to find x_1, x_2 so as to max the profit. Constraints:

1. the daily demand of Choco is at most 300 boxes ($x_1 \leq 300$)
2. the daily demand of Fancy Choco is at most 200 ($x_2 \leq 200$)
3. the current workforce can produce at most 400 boxes daily

Introduction to Linear Programming (Example 1) [4]

A chocolate shop sells two products: Choco (1€ per box) and Fancy Choco (6€ per box). It makes x_1 boxes of Choco and x_2 boxes of Fancy Choco, daily. We wish to find x_1, x_2 so as to max the profit. Constraints:

1. the daily demand of Choco is at most 300 boxes ($x_1 \leq 300$)
2. the daily demand of Fancy Choco is at most 200 ($x_2 \leq 200$)
3. the current workforce can produce at most 400 boxes daily

Linear Program

$$\begin{array}{rl} \max & x_1 + 6x_2 \\ \text{s.t.} & x_1 \leq 300 \\ & x_2 \leq 200 \\ & x_1 + x_2 \leq 400 \\ & x_1, x_2 \geq 0. \end{array}$$

Introduction to Linear Programming (Example 2)

We need to carry 1kg of salt, 1kg of floor, 1kg of olive oil, 1kg of tomato sauce, 1kg of mozzarella (ingredients for pizza). However, our knapsack can only carry 3.5kg of goods. We can sell olive oil for 3€ per Kg, mozzarella 2€ per Kg, the rest 1€ per kg. We need to carry at least 1.5 Kg of mozzarella. We would like to fill our knapsack so as to maximize our profit. We can write the following LP:

$$
\begin{aligned}
\max \quad & x_s + x_f + 3x_o + x_t + 2x_m \\
\text{s.t.} \quad & x_s + x_f + x_o + x_t + x_m \leq 3.5 \\
& x_m \geq 1.5 \\
& x_s, x_f, x_o, x_t, x_m \geq 0.
\end{aligned}
$$

Example 3. Maximum Flow problems can also be formulated as LP's.

Matrix Vector Notation

A linear function (e.g. $x_1 + 6x_2$) can be written as the dot product of two vectors:

$$\mathbf{c} = \begin{bmatrix} 1 \\ 6 \end{bmatrix} \text{ and } \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

denoted $\mathbf{c}^T\mathbf{x}$. Similarly, linear constraints can be expressed into matrix-vector form:

$$\begin{aligned} x_1 &\leq 300 \\ x_2 &\leq 200 \\ x_1 + x_2 &\leq 400 \end{aligned} \quad \Rightarrow \quad \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\mathbf{x}} \leq \underbrace{\begin{bmatrix} 300 \\ 200 \\ 400 \end{bmatrix}}_{\mathbf{b}}$$

Matrix Vector Notation

A generic LP can be expressed in the following compact form:

$$\max \mathbf{c}^T \mathbf{x}$$
$$\mathbf{A}\mathbf{x} \leq \mathbf{b}$$
$$\mathbf{x} \geq 0,$$

where $\mathbf{A}, \mathbf{c}, \mathbf{b}, \mathbf{x}$, are rational-number matrices and vectors.

LP's can be solved in polynomial time with the *ellipsoid method* or some *interior point* methods. Simplex method is more efficient in practice although it might not run in polynomial time.

CPLEX, Gurobi are commercial solvers that can be used to solve efficiently relatively large LP's. They can be used for free for non-profit purposes.

Integer Linear Programs

Can we solve in polynomial linear programs of the form?

$$
\begin{aligned}
\max \mathbf{c}^T \mathbf{x} \\
\mathbf{A}\mathbf{x} \leq \mathbf{b} \\
\mathbf{x} \in \{0,1\}^n
\end{aligned}
$$

Integer Linear Programs

Can we solve in polynomial linear programs of the form?

$$\max \mathbf{c}^T \mathbf{x}$$
$$\mathbf{A}\mathbf{x} \leq \mathbf{b}$$
$$\mathbf{x} \in \{0, 1\}^n$$

No! Many NP-Hard problems (set cover, 0-1 knapsack, etc.) can be formulated that way.

Towards an LP Formulation for Densest Subgraph

Let $G = (V, E)$, we introduce y_u ($u \in V$), with $y_u = 1$ if u is selected, 0 otherwise and x_e ($e \in E$), with $x_e = 1$ if e is selected, 0 otherwise.

We derive:

$$
\begin{array}{lll}
\max & \dfrac{1}{\sum_{u \in V} y_u} \displaystyle\sum_{e \in E} x_e & \\
\text{s.t.} & x_e \leq y_u & \forall e \in E, \forall u \in e, \\
& x_e, y_u \in \{0, 1\} & \forall u \in V, e \in E.
\end{array}
$$

LP Formulation

By dividing the variables by $\sum_{u \in V} y_u$, adding the constraint $\sum_{u \in V} y_u = 1$ and relaxing the integral constraints, we obtain:

$$\begin{aligned}
\max \quad & \sum_{e \in E} x_e & \\
\text{s.t.} \quad & x_e \le y_u & \forall e \in E, \forall u \in e \\
& \sum_{u \in V} y_u = 1, & \\
& x_e, y_u \ge 0 & \forall u \in V, e \in E.
\end{aligned}$$

LP-based Algorithm

It consists of the following two steps:

- solve the LP formulation for the densest subgraph problem;
- Let $\bar{x}, \bar{y}$ be an optimum solution to the LP. Let $\bar{y}_1 \geq \bar{y}_2 \geq \cdots \geq \bar{y}_n$ ($\bar{y}_i$ associated to v_i). Return a prefix $\bar{y}_1, \ldots, \bar{y}_k$ of $\bar{y}_1, \ldots, \bar{y}_n$, whose nodes $v_1, \ldots, v_k$ induce a subgraph with maximum density.

Theorem

The LP-based algorithm computes a densest subgraph in the input graph.

Proof of Theorem 11

Lemma

Let ρ_O be the density of a densest subgraph in G. Let λ be the value of an optimum solution to the LP. It holds:

$$\lambda \geq \rho_O.$$

Proof.

Let $O = (V_O, E_O)$ be a densest subgraph in G. For every $u \in V_O$, let $\bar{y}_u = \frac{1}{|V_O|}$, 0 otherwise. For every edge $e \in E_O$, let $\bar{x}_e = \frac{1}{|V_O|}$, 0 otherwise. $\bar{x}, \bar{y}$ is feasible and its value is $\frac{|E_O|}{|V_O|} = \rho_O$. $\square$

Lemma

Let ρ_O be the density of a densest subgraph in G. Let λ be the value of an optimum solution to the LP. It holds $\lambda \leq \rho_O$.

Proof.

Let $(\bar{x}, \bar{y})$ be a feasible LP solution with value $\bar{\lambda}$. W.l.g., $\forall u, v \in V$, $\bar{x}_{uv} = \min(\bar{y}_u, \bar{y}_v)$. Let

$$S(r) := \{u : \bar{y}_u \geq r\}, \quad E(r) := \{uv \in E : \bar{x}_{uv} \geq r\}.$$

Since $\bar{x}_{uv} \leq \bar{y}_v$ and $\bar{x}_{uv} \leq \bar{y}_u$, $uv \in E(R) \Rightarrow u \in S(r), v \in S(r)$. Since $\bar{x}_{uv} = \min(\bar{y}_u, \bar{y}_v)$, $u \in S(r), v \in S(r) \Rightarrow uv \in E(r)$. Therefore $E(r)$ is the set of edges induced by $S(r)$.

(Cont.)

Proof of Lemma 13

Proof.

We have:

$$\int_0^{\infty} |S(r)| dr = \sum_{u \in V} \bar{y}_u \leq 1, \quad \int_0^{\infty} |E(r)| dr = \sum_{uv \in E} \bar{x}_{uv} = \bar{\lambda}.$$

Suppose there is no r such that $\frac{|E(r)|}{|S(r)|} \geq \bar{\lambda}$. Then:

$$\int_0^{\infty} |E(r)| dr < \bar{\lambda} \int_0^{\infty} |S(r)| dr \leq \bar{\lambda},$$

which gives a contradiction. □

What about edge density?

Finding a "large" subgraph with maximum edge density is very difficult!.

However, in practice the following heuristic works very well. Modify the greedy algorithm as follows:

at each step remove the node belonging to the smallest number of k-cliques[1] until the graph becomes empty. Return the subgraph H maximizing the number of k-cliques in H divided by $|V(H)|$.

Typical values of k are 3 or 4. See algorithms for listing cliques [6] and counting cliques [7].

[1] k-clique = graph with k nodes each pair of which being connected by an edge

References I

Khuller, Samir, and Barna Saha.
On finding dense subgraphs
ICALP, 2009.

Charikar, Moses.
Greedy approximation algorithms for finding dense components in a graph
APPROX 2000.

Charikar, Moses.
Goldberg, Andrew V.
Finding a maximum density subgraph.
Berkeley, CA: University of California, 1984.

Vazirani, Vijay V.
Approximation Algorithms.
Springer 2003.

References II

Maximilien Danisch, T-H. Hubert Chan and Mauro Sozio
Large Scale Density-friendly Graph Decomposition via Convex Programming
26th International World Wide Web Conference (WWW 17)

Maximilien Danisch, Oana Balalau and Mauro Sozio
Listing k-cliques in Real World Graphs
27th International World Wide Web Conference (WWW 18)

Shweta Jain, C. Seshadhri
A Fast and Provable Method for Estimating Clique Counts Using Turán's Theorem,
26th International Conference on World Wide Web (WWW 17)