Towards Real-World Graph Algorithmics

Maximilien Danisch

PostDoc at Télécom ParisTech, DBWeb team

Institut Mines-Télécom

Real-world graphs (a.k.a. complex networks)

networks	nodes	edges
Facebook	profiles	friendship
Internet	computers	connections
Web	web pages	hyperlinks
Brain	neurons	synapses
Even more general		

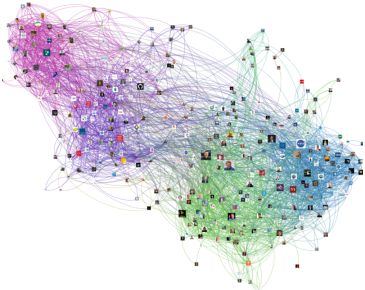
Definition

Network/Graph. A set of nodes linked by edges.

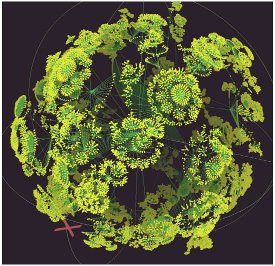
Basic properties

- Very large
- Sparse
- Many triangles
- Small diameter
- Heterogeneous degrees (hubs)

Twitter



Internet



▶ **Need efficient algorithms for real-world graphs.**

Real-world graph algorithmics ? ? ?

Special structure ⇒ special algorithmics

- Finding a maximum clique: NP-hard but "easy".
  greedy + Branch & Bound. Rossi et al. WWW2014.
- A polynomial complexity might not be "good".
  Length of all shortest paths $\Theta(n^3)$. Floyd-Warshall.
- Complexity in practice often way better than the worst case.
  Convergence of the algorithm of Louvain. Blondel et al. JSTAT2008.
- Algorithms for classes of graphs are not usable such as.
  Perfect graphs. Chudnovsky et al. Annals of mathematics 2006.

My goal

Understanding and leveraging the structure of real-world graphs in order to make better algorithms.

Working paper

Listing k-Cliques in Large Real-World Graphs

Maximilien Danisch
Télécom ParisTech University, Paris, France
danisch@telecom-paristech.fr

Oana Denisa Balalau
Télécom ParisTech University, Paris, France
balalau@telecom-paristech.fr

Mauro Sozio
Télécom ParisTech University, Paris, France
sozio@telecom-paristech.fr

ABSTRACT

The problem of listing and counting triangles has been intensively studied by the research community, in recent years. In contrast, the problem of listing and counting k-cliques has received much less attention. Motivated by recent studies in the data mining community which call for efficient algorithms for such a problem, we develop the most efficient parallel algorithm for counting and listing all k-cliques in a graph. Our theoretical analysis shows that our algorithm outperforms state-of-the-art algorithms for the same problem, while leveraging the sparsity of real-world graphs. Our experimental analysis on large real-world graphs shows that our algorithm is able to list all k-cliques in graphs containing tens of millions of edges as well as all 10-cliques in graphs containing billions of edges, within a few minutes and a few hours, respectively, while achieving excellent degree of parallelism. Armed with such a powerful tool, we define and study a natural generalization of the core decomposition (which we call k-clique core decomposition) and develop an efficient algorithm for computing such a decomposition. Our algorithm for counting k-cliques can be employed as an effective subroutine for finding approximate k-clique densest subgraphs. Finally, we show that our algorithms can effec-

each pair of which being connected with an edge. Such a problem is a natural generalization of the problem of counting triangles in a graph, which has been intensively studied by the research community. Surprisingly, the problem of listing or counting k-cliques has not received much attention in recent years. This is due perhaps to its computational challenges and from the fact that materializing or even storing all k-cliques might not be feasible if the input graph is both large and dense.

Recent works in the data mining and database community call for efficient algorithms for listing or counting all k-cliques in the input graph. In particular, in [40] the author develops an algorithm for finding subgraphs with maximum average number of k-cliques, with counting k-cliques being an important building block. In [35] an algorithm for organizing cliques into hierarchical structures is presented, which requires to list all k-cliques. In [3], algorithms for finding cliques and quasi-cliques (i.e. cliques where a few edges might be missing) with at most k nodes are used for story-identification in social media. Efficient algorithms for counting k-cliques would allow for the computation of a natural generalization of the well-known core decomposition, which we formally define in this paper and we call the *k-clique core decomposition*.

Motivation. Why listing k-cliques

Recent work in data mining are calling for it

- Community detection: percolated k-cliques.
  Uncovering the overlapping community structure of complex networks in nature and society.
  Palla et al. Nature2005.
- Dense subgraph: k-clique densest subgraph.
  The K-clique Densest Subgraph Problem.
  Charalampos WWW2015.
- K-cliques can be seen as building-blocks of real-world graphs.
  Higher-order organization of complex networks.
  Benson, Gleich & Leskovec Science, 2016

A challenging problem

Many k-cliques

- Number of 10-cliques in Friendster (1.8G edges, 65M nodes) = 487 090 833 092 739
- Number of 5-cliques in Twitter09 (1.6G edges, 53M nodes) = 3 388 795 307 518 264

So many k-cliques

- It can be very hard to store all k-cliques.
- We rather suggest to compute quantities on the fly.

Our main contribution

- We show that listing k-cliques is more tractable and useful than what people think *Jain et Seshadhri WWW2017.*

Outline

- **Algorithm**
- Theoretical analysis
- Comparison to other methods
- Application to data mining

A naive algorithm

Algorithm 1 A naive algorithm for finding 3, 4, 5-cliques

```
for each edge (u, v) ∈ E(G) do
    Δ(u, v) ← Δ(u) ∩ Δ(v)                    ▷ ∀ u ∈ V, Δ(u) = neighbors of u
    for each w in Δ(u, v) do
        output triangle {u, v, w}
        Δ(u, v, w) ← Δ(u, v) ∩ Δ(w)
        for each x in Δ(u, v, w) do
            output 4-clique {u, v, w, x}.
            Δ(u, v, w, x) ← Δ(u, v, w) ∩ Δ(x)
            for each y in Δ(u, v, w, x) do
                output 5-clique {u, v, w, x, y}
```

Problem

- k-cliques outputted several times: $\frac{k!}{2}$.
- Bad worst case running time and do not scale to large graphs.

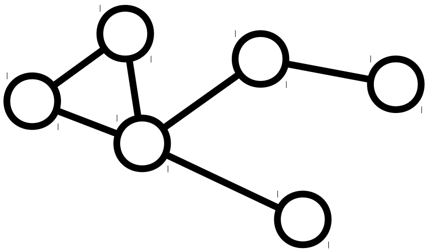
Core decomposition

Definition. Core value

The maximum number $c(G)$ such that there exists an induced subgraph of G with all nodes having degree at least $c(G)$.

Algorithm Core decomposition

1: $i \leftarrow 1$, $c \leftarrow 0$
2: **while** $V(G) \neq \emptyset$ **do**
3: Let v be a node with minimum degree in G
4: $c \leftarrow \max(c, d_G(v))$
5: $V(G) \leftarrow V(G) \setminus \{v\}$
6: $E(G) \leftarrow E(G) \setminus \Delta(v)$
7: $\eta(v) = i$
8: $i \leftarrow i + 1$



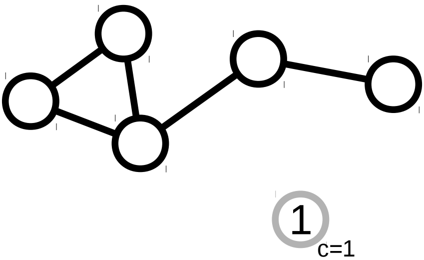
Core decomposition

Definition. Core value

The maximum number $c(G)$ such that there exists an induced subgraph of G with all nodes having degree at least $c(G)$.

Algorithm Core decomposition

$i \leftarrow 1$, $c \leftarrow 0$
while $V(G) \neq \emptyset$ **do**
  Let v be a node with minimum degree in G
  $c \leftarrow \max(c, d_G(v))$
  $V(G) \leftarrow V(G) \setminus \{v\}$
  $E(G) \leftarrow E(G) \setminus \Delta(v)$
  $\eta(v) = i$
  $i \leftarrow i + 1$



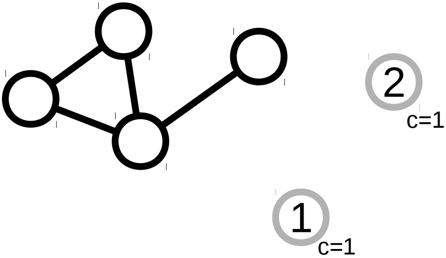
Core decomposition

Definition. Core value

The maximum number $c(G)$ such that there exists an induced subgraph of G with all nodes having degree at least $c(G)$.

Algorithm Core decomposition

$i \leftarrow 1$, $c \leftarrow 0$
while $V(G) \neq \emptyset$ **do**
  Let v be a node with minimum degree in G
  $c \leftarrow \max(c, d_G(v))$
  $V(G) \leftarrow V(G) \setminus \{v\}$
  $E(G) \leftarrow E(G) \setminus \Delta(v)$
  $\eta(v) = i$
  $i \leftarrow i + 1$



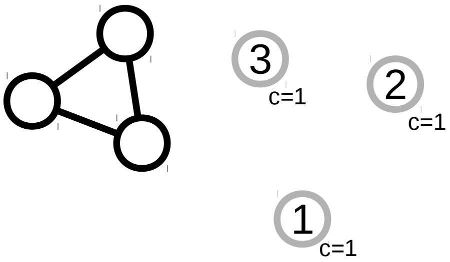
Core decomposition

Definition. Core value

The maximum number $c(G)$ such that there exists an induced subgraph of G with all nodes having degree at least $c(G)$.

Algorithm Core decomposition

1: $i \leftarrow 1$, $c \leftarrow 0$
2: **while** $V(G) \neq \emptyset$ **do**
3: Let v be a node with minimum degree in G
4: $c \leftarrow \max(c, d_G(v))$
5: $V(G) \leftarrow V(G) \setminus \{v\}$
6: $E(G) \leftarrow E(G) \setminus \Delta(v)$
7: $\eta(v) = i$
8: $i \leftarrow i + 1$



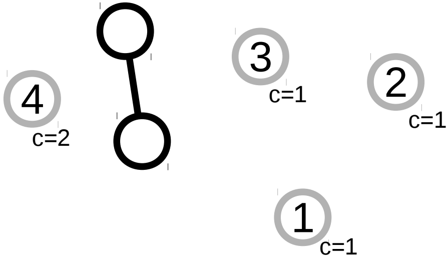
Core decomposition

Definition. Core value

The maximum number $c(G)$ such that there exists an induced subgraph of G with all nodes having degree at least $c(G)$.

Algorithm Core decomposition

$i \leftarrow 1$, $c \leftarrow 0$
while $V(G) \neq \emptyset$ **do**
    Let v be a node with minimum degree in G
    $c \leftarrow \max(c, d_G(v))$
    $V(G) \leftarrow V(G) \setminus \{v\}$
    $E(G) \leftarrow E(G) \setminus \Delta(v)$
    $\eta(v) = i$
    $i \leftarrow i + 1$



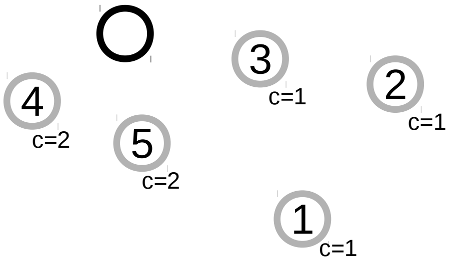
Core decomposition

Definition. Core value

The maximum number $c(G)$ such that there exists an induced subgraph of G with all nodes having degree at least $c(G)$.

Algorithm Core decomposition

$i \leftarrow 1$, $c \leftarrow 0$
while $V(G) \neq \emptyset$ **do**
  Let v be a node with minimum degree in G
  $c \leftarrow \max(c, d_G(v))$
  $V(G) \leftarrow V(G) \setminus \{v\}$
  $E(G) \leftarrow E(G) \setminus \Delta(v)$
  $\eta(v) = i$
  $i \leftarrow i + 1$



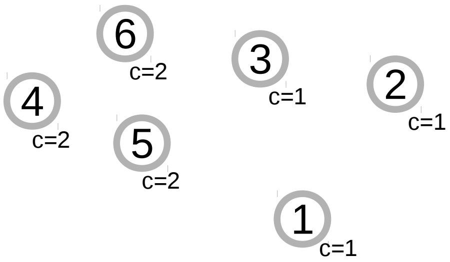
Core decomposition

Definition. Core value

The maximum number $c(G)$ such that there exists an induced subgraph of G with all nodes having degree at least $c(G)$.

Algorithm Core decomposition

$i \leftarrow 1$, $c \leftarrow 0$
while $V(G) \neq \emptyset$ **do**
  Let v be a node with minimum degree in G
  $c \leftarrow \max(c, d_G(v))$
  $V(G) \leftarrow V(G) \setminus \{v\}$
  $E(G) \leftarrow E(G) \setminus \Delta(v)$
  $\eta(v) = i$
  $i \leftarrow i + 1$



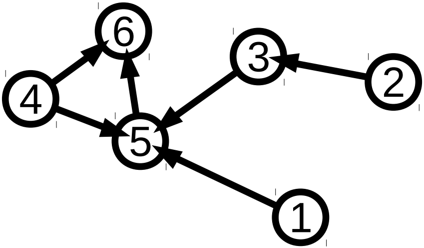
Core decomposition

Definition. Core value

The maximum number $c(G)$ such that there exists an induced subgraph of G with all nodes having degree at least $c(G)$.

Algorithm Core decomposition

$i \leftarrow 1,\ c \leftarrow 0$
while $V(G) \neq \emptyset$ **do**
    Let v be a node with minimum degree in G
    $c \leftarrow \max(c, d_G(v))$
    $V(G) \leftarrow V(G) \setminus \{v\}$
    $E(G) \leftarrow E(G) \setminus \Delta(v)$
    $\eta(v) = i$
    $i \leftarrow i + 1$



Core decomposition

Definition. Core value

The maximum number $c(G)$ such that there exists an induced subgraph of G with all nodes having degree at least $c(G)$.

Algorithm Core decomposition

1: $i \leftarrow 1$, $c \leftarrow 0$
2: **while** $V(G) \neq \emptyset$ **do**
3: Let v be a node with minimum degree in G
4: $c \leftarrow \max(c, d_G(v))$
5: $V(G) \leftarrow V(G) \setminus \{v\}$
6: $E(G) \leftarrow E(G) \setminus \Delta(v)$
7: $\eta(v) = i$
8: $i \leftarrow i + 1$

Properties

$\Delta_\eta(u) \leftarrow$ sorted list of neighbors v of u such that $\eta(v) > \eta(u)$.
We have

- $\forall\ u$, $|\Delta_\eta(u)| \leq c$,
- $\forall\ (u, v)$, $\Delta_\eta(u) \cap \Delta_\eta(v)$ can be computed in time $O(c)$ and
- c is small in front of n.

Our algorithm

Algorithm 2 Parallel algorithm for finding $3, 4, 5$-cliques

$\eta \leftarrow$ Core ordering of G
for each node u in G **do**
    $\Delta_\eta(u) \leftarrow$ sorted list of neighbors v of u, $\eta(v) > \eta(u)$
for each edge $(u, v) \in E(G)$ **do** ▷ **Parallel** loop
    $\Delta_\eta(u, v) \leftarrow \Delta_\eta(u) \cap \Delta_\eta(v)$
    for each w in $\Delta_\eta(u, v)$ **do**
        output triangle $\{u, v, w\}$
        $\Delta_\eta(u, v, w) \leftarrow \Delta_\eta(u, v) \cap \Delta_\eta(w)$
        for each x in $\Delta_\eta(u, v, w)$ **do**
            output 4-clique $\{u, v, w, x\}$.
            $\Delta_\eta(u, v, w, x) \leftarrow \Delta_\eta(u, v, w) \cap \Delta_\eta(x)$
            for each y in $\Delta_\eta(u, v, w, x)$ **do**
                output 5-clique $\{u, v, w, x, y\}$

Outline

- Algorithm
- **Theoretical analysis**
- Comparison to other methods
- Application to data mining

Worst case running time

Algorithm 3 Parallel algorithm for finding $3, 4, 5$-cliques

$\eta \leftarrow$ Core ordering of G $\quad \triangleright\ O(m)$

for each node u in G **do**

  $\Delta_\eta(u) \leftarrow$ sorted list of neighbors v of u, $\eta(v) > \eta(u)$ $\quad \triangleright\ O(m)$

for each edge $(u, v) \in E(G)$ **do** $\quad \triangleright\ O(m)$

  $\Delta_\eta(u, v) \leftarrow \Delta_\eta(u) \cap \Delta_\eta(v)$ $\quad \triangleright\ O(c)$

  for each w in $\Delta_\eta(u, v)$ **do** $\quad \triangleright\ O(N_3)$

    output triangle $\{u, v, w\}$

    $\Delta_\eta(u, v, w) \leftarrow \Delta_\eta(u, v) \cap \Delta_\eta(w)$ $\quad \triangleright\ O(c)$

    for each x in $\Delta_\eta(u, v, w)$ **do** $\quad \triangleright\ O(N_4)$

      output 4-clique $\{u, v, w, x\}$.

      $\Delta_\eta(u, v, w, x) \leftarrow \Delta_\eta(u, v, w) \cap \Delta_\eta(x)$ $\quad \triangleright\ O(c)$

      for each y in $\Delta_\eta(u, v, w, x)$ **do** $\quad \triangleright\ O(N_5)$

        output 5-clique $\{u, v, w, x, y\}$

Algorithm 3 requires $O(c \cdot \sum_{l=2}^{k-1} N_l)$ total number of operations.

Theoretical analysis

Theorem 1

Algorithm 3 requires $O(c \cdot \sum_{l=2}^{k-1} N_l)$ total number of operations.
It requires linear amount of memory in the size of the graph.

Lemma

Let $k > 1$ be an integer, it holds that $N_k \leq m \cdot \binom{c-1}{k-2} \leq 2 \cdot m \cdot (\frac{c-1}{2})^{k-2}$.

Theorem 2

Algorithm 3 requires $O(m \cdot (\frac{c-1}{2})^{k-2})$ total number of operations for counting k-cliques.
It requires linear amount of memory in the size of the graph.

N_l denotes the number of l-cliques in G.
c denotes the core value of G.

Outline

- Algorithm
- Theoretical analysis
- **Comparison to other methods**
- Application to data mining

Comparison to other methods, in theory

- $O(c \cdot \sum_{l=2}^{k-1} N_l)$ is the best output sensitive bound ever reported.
- $O(m \cdot (\frac{c-1}{2})^{k-2})$ is the best bound ever reported for sparse graphs.

Competitors

- *Finding and counting given length cycles.*
  Alon et al. (Algorithmica 1997).
  - For triangle counting only. $O(m^{1.41})$ and $O(m \cdot c)$.
- *Main-memory Triangle Computations for Very Large Sparse Power-Law Graphs. Latapy (TCS 2008).*
  - For triangles only. $O(m^{\frac{3}{2}})$ and $O(m \cdot n^{\frac{1}{\alpha}})$ for power-law graphs.
- *Arboricity and Subgraph Listing Algorithms.*
  Chiba and Nishizeki (SIAM 1986).
  - Not parallel. $O(m \cdot a^{k-2})$.
    a is the arboricity. Note that $a \leq c \leq 2 \cdot a - 1$.

Comparison to other methods, in practice

TABLE : Our set of large graphs (for which we are able to count all cliques)

networks	n	m	c	k_{max}	$N_{k_{max}}$
soc-pocket	1 632 803	22 301 964	47	29	6
loc-gowalla	196 591	950 327	51	29	2
Youtube	1 134 890	2 987 624	51	17	2
cit-patents	3 774 768	16 518 947	64	11	2
zhishi-baidu	2 140 198	17 014 946	78	31	4
WikiTalk	2 394 385	4 659 565	131	26	141

.

TABLE : Our set of very large graphs (can count k-cliques of limited size).

networks	n	m	c
DBLP	425 957	1 049 866	113
Wikipedia	2 080 370	42 336 692	208
Orkut	3 072 627	117 185 083	253
Friendster	124 836 180	1 806 067 135	304
LiveJournal	4 036 538	34 681 189	360
Twitter	52 579 683	1 614 106 500	2647

Comparison to other methods, in practice

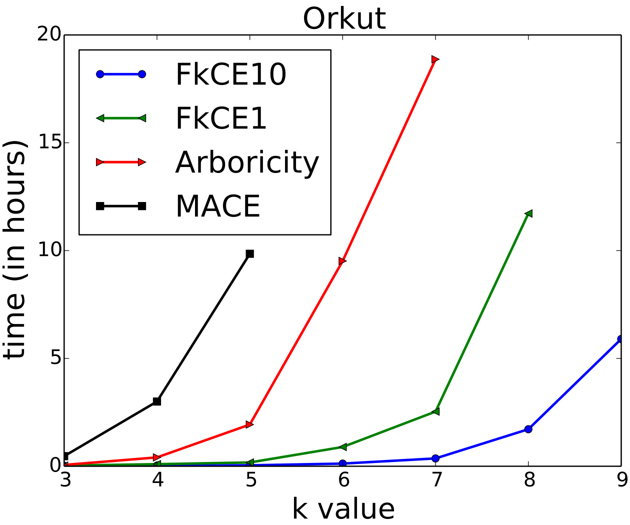
TABLE : Running time for counting triangles on our very large graphs.

		Algorithms			
networks	**# triangles**	**CF**	**MACE**	**Arboricity**	**FkCE1**
DBLP	2 224 385	0.8s	1.3s	0.6s	**0.4s**
Wikipedia	145 707 846	1m07s	22m22s	1m04s	**40s**
Orkut	627 584 181	4m06s	28m02s	3m41s	**2m14s**
Friendster	4 173 724 142	1h50m41s	5h29m40s	2h57m21s	**1h05m31s**
LiveJournal	177 820 130	44s	6m13s	37s	**27s**
Twitter	55 428 265 841	1h57m31s	$>$24h	3h55m38s	**1h24m13s**

TABLE : Time for counting all cliques on our large graphs.

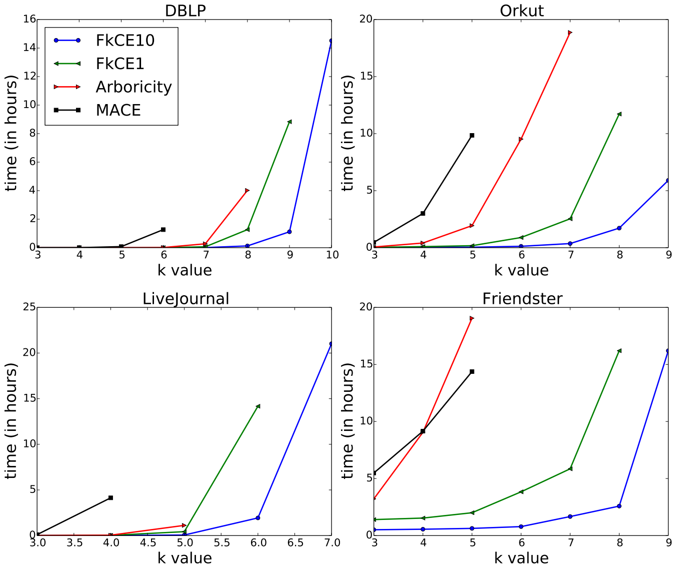
	Algorithms		
networks	**MACE**	**Arboricity**	**FkCE1**
soc-pocket	14m27s	11m23s	**1m15s**
loc-gowalla	8m46s	7m52s	**34s**
Youtube	1m05s	1m12s	**3.9s**
cit-patents	22s	24s	**15s**
zhishi-baidu	1h00m44s	32m23s	**3m58s**
WikiTalk	$>$24h	$>$24h	**5h53m36s**

Comparison to other methods, in practice

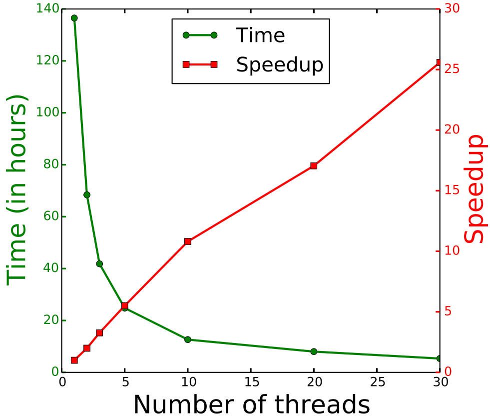


TELECOM ParisTech

Comparison to other methods, in practice



Parallelism



No need to share locks among threads: we achieve an almost optimal degree of parallelism.

TELECOM ParisTech

Outline

- Algorithm
- Theoretical analysis
- Comparison to other methods
- **Application to data mining**

Solving new problems

- **Problem definition (k-clique core decomposition).** Given an undirected graph $G = (V(G), E(G))$ and an integer $k > 1$, compute a k-clique core decomposition of G.

Algorithm

- Same algorithm as the core decomposition removing a node of minimum k-clique degree.
- We use our k-clique algorithm on the whole graph to initialize the k-clique degree of each node.
- Given a node to remove, we use our k-clique algorithm on the subgraph induced by its neighbors to update their k-clique degree.

We can solve this problem on very large real-world graphs without storing all k-cliques.

Solving new problems

- **Problem definition (k-clique densest problem).** Given an undirected graph $G = (V(G), E(G))$, find a subgraph H of G such that the k-clique density is maximized.

Theorem

The k-clique densest prefix of a k-clique core decomposition is a $\frac{1}{k}$-approximation of the k-clique densest subgraph.

We can give an approximated solution to this problem on very large real-world graphs without storing all k-cliques.

Conclusion and future work

- Conclusion
  - Best known asymptotic running time for sparse graphs
  - Linear memory
  - An order of magnitude faster than state-of-the-art algorithms
  - Almost optimal degree of parallelism
  - We generate a stream of k-cliques to compute the k-clique core decomposition
- Future Work.
  - Our algorithm could be employed in graph compression, community and event detection
  - Stream of k-cliques to compute other quantities
  - Our k-clique core decomposition seems to be a promising tool for data mining

WWW2017 paper

Large Scale Density-friendly Graph Decomposition via Convex Programming

Maximilien Danisch
LTCI, Télécom ParisTech,
Université Paris-Saclay,
75013, Paris, France
danisch@telecom-paristech.fr

T-H. Hubert Chan
Department of Computer Science, The University of Hong Kong, Hong Kong, China
hubert@cs.hku.hk

Mauro Sozio
LTCI, Télécom ParisTech,
Université Paris-Saclay,
75013, Paris, France
sozio@telecom-paristech.fr

ABSTRACT

Algorithms for finding dense regions in an input graph have proved to be effective tools in graph mining and data analysis. Recently, Tatti and Gionis [WWW 2015] presented a novel graph decomposition (known as the locally-dense decomposition) that is similar to the well-known k-core decomposition, with the additional property that its components are arranged in order of their densities. Such a decomposition provides a valuable tool in graph mining. Unfortunately, their algorithm for computing the exact decomposition is based on a maximum-flow algorithm which cannot scale to massive graphs, while the approximate decomposition defined by the same authors misses several interesting properties. This calls for scalable algorithms for computing such a decomposition. In our work, we devise an efficient algorithm which is able to compute exact locally-dense decompositions in real-world graphs containing up to billions of edges. Moreover, we provide a new definition of approximate locally-dense decomposition which retains most of the properties of an exact decomposition, for which we devise an algorithm that can scale to real-world graphs containing up to tens of billions of edges. Our algorithm is based on the

the well-known k-core decomposition stands out for its simplicity and its ability to unravel the structural organization of a graph. It has been successfully applied in many contexts such as speeding up algorithms [19, 33], finding best spreaders [27], drawing large graphs [2], bioinformatics [5], analyzing human brains [23] and team formation [10].

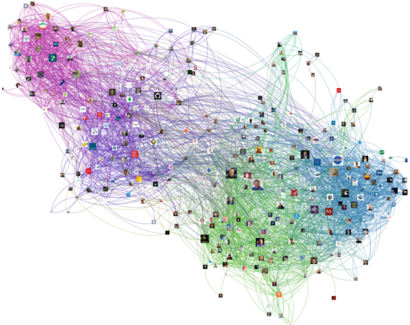
Recently, Tatti and Gionis [38] proposed a novel graph decomposition, known as the *locally-dense graph decomposition*. Such a decomposition boasts similar properties to the k-core decomposition with the additional property that its components are nested into one another, with inner components having larger density than outer ones. Moreover, the locally-dense decomposition contains all the so-called *locally-dense* subgraphs of the input graph. The locally-dense graph decomposition provides a valuable tool in graph mining.

Unfortunately, their algorithm for computing the exact decomposition does not scale to massive graphs, while the approximate decomposition defined by the same authors may not contain any non-trivial locally-dense subgraph.

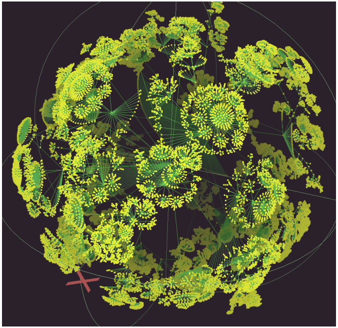
In our work, we devise an efficient algorithm for computing exact locally-dense decompositions in massive graphs. Our main algorithm is based on a variant of the classic Frank-Wolfe algorithm that is similar to gradient descent

Motivation. Mining large graphs

Twitter



Internet



The density-friendly decomposition (Tatti and Gionis WWW2015) is interesting as it merges two classic graph mining concepts:

1. k-core decomposition
2. dense subgraph

Outline

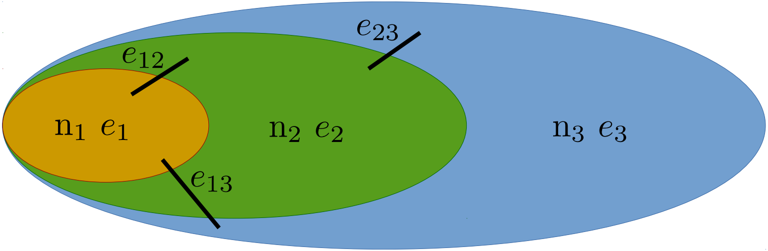
- **A new and intuitive definition of density-friendly**
- A very simple, yet powerful, algorithm
- Theoretical analysis via convex programing
- Experiments

A definition of density-friendly

Definition (density-friendly)

Collection of non-overlapping sets of nodes $\{B_i\}$, such that

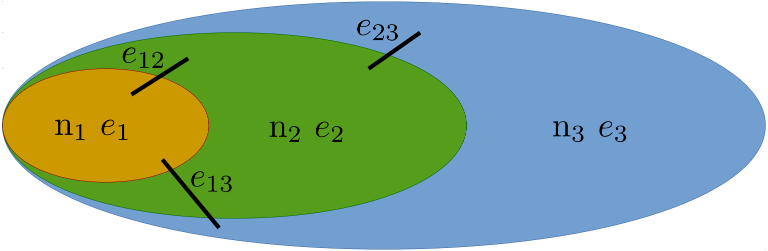
- B_1 maximizes $\frac{e_1}{n_1}$ and has maximum size,
- B_2 maximizes $\frac{e_2+e_{12}}{n_2}$ and has maximum size,
- B_3 maximizes $\frac{e_3+e_{13}+e_{23}}{n_3}$ and has maximum size, ...



A definition of density-friendly

Theorem

Our definition is equivalent to the one of Tatti and Gionis WWW2015.



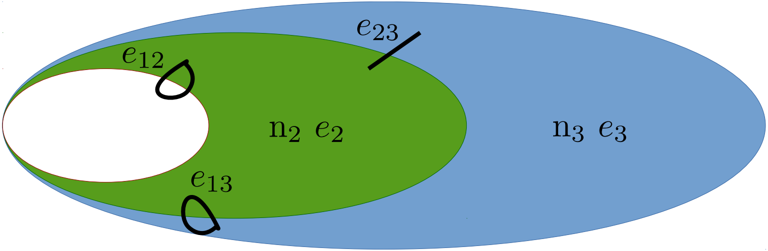
Algorithm (do not scale to huge graphs)

1. find the densest subgraph (Goldberg's maxflow algorithm)
2. remove it and form self-loops with outgoing edges
3. go to 1. taking into account self-loops

A definition of density-friendly

Theorem

Our definition is equivalent to the one of Tatti and Gionis WWW2015.



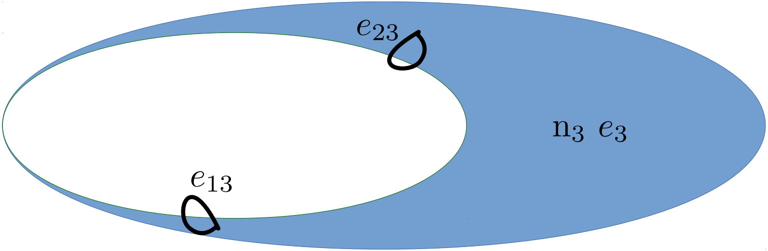
Algorithm (do not scale to huge graphs)

1. find the densest subgraph (Goldberg's maxflow algorithm)
2. remove it and form self-loops with outgoing edges
3. go to 1. taking into account self-loops

A definition of density-friendly

Theorem

Our definition is equivalent to the one of Tatti and Gionis WWW2015.



Algorithm (do not scale to huge graphs)

1. find the densest subgraph (Goldberg's maxflow algorithm)
2. remove it and form self-loops with outgoing edges
3. go to 1. taking into account self-loops

Outline

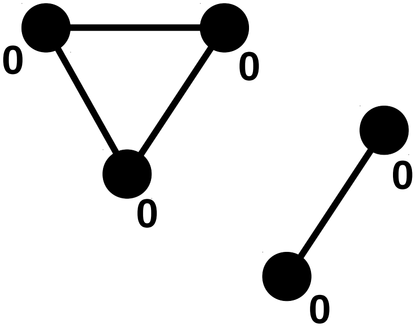
- A new and intuitive definition of density-friendly
- **A very simple, yet powerful, algorithm**
- Theoretical analysis via convex programing
- Experiments

A simple algorithm

Algorithm Compute density score

function MKSCORE(G,t)
  Set $\mathbf{r}[i] = 0$ for each node i in G
  iterate t times
    for each edge (i, j) in G **do**
      if $\mathbf{r}[i] \leq \mathbf{r}[j]$ **then**
        $\mathbf{r}[i]{+}{+}$
      else
        $\mathbf{r}[j]{+}{+}$
  return $\mathbf{r}/t$

For $t = 2$ iterations

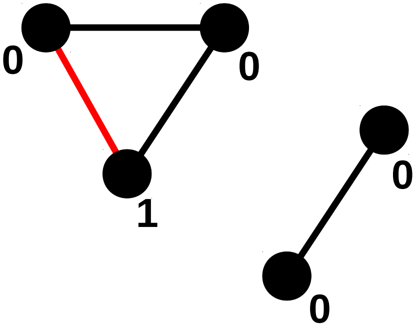


A simple algorithm

Algorithm Compute density score

function MKSCORE(G,t)
    Set $\mathbf{r}[i] = 0$ for each node i in G
    iterate t times
        for each edge (i, j) in G **do**
            if $\mathbf{r}[i] \leq \mathbf{r}[j]$ **then**
                $\mathbf{r}[i]{+}{+}$
            else
                $\mathbf{r}[j]{+}{+}$
    return $\mathbf{r}/t$

For $t = 2$ iterations



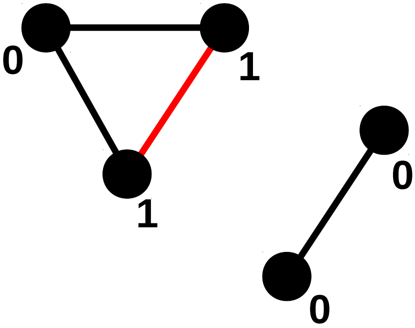
A simple algorithm

Algorithm Compute density score

---

function MKSCORE(G,t)
  Set $\mathbf{r}[i] = 0$ for each node i in G
  iterate t times
    for each edge (i, j) in G **do**
      if $\mathbf{r}[i] \leq \mathbf{r}[j]$ **then**
        $\mathbf{r}[i]{+}{+}$
      else
        $\mathbf{r}[j]{+}{+}$
  return $\mathbf{r}/t$

For $t = 2$ iterations



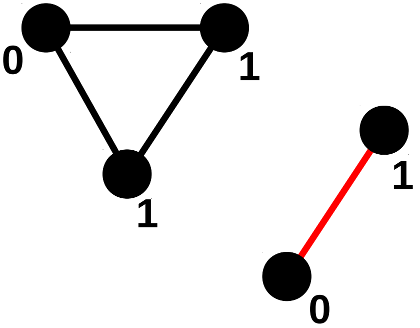
TELECOM ParisTech

A simple algorithm

Algorithm Compute density score

function MKSCORE(G,t)
  Set $\mathbf{r}[i] = 0$ for each node i in G
  iterate t times
    for each edge (i, j) in G **do**
      if $\mathbf{r}[i] \leq \mathbf{r}[j]$ **then**
        $\mathbf{r}[i]{+}{+}$
      else
        $\mathbf{r}[j]{+}{+}$
  return $\mathbf{r}/t$

For $t = 2$ iterations

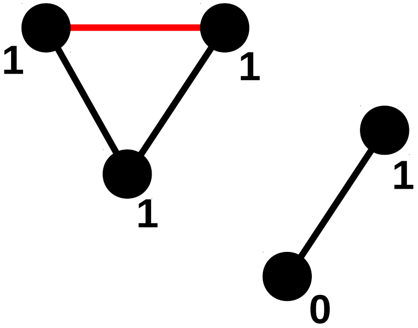


A simple algorithm

Algorithm Compute density score

function MKSCORE(G,t)
  Set $\mathbf{r}[i] = 0$ for each node i in G
  iterate t times
    for each edge (i, j) in G **do**
      if $\mathbf{r}[i] \leq \mathbf{r}[j]$ **then**
        $\mathbf{r}[i]{++}$
      else
        $\mathbf{r}[j]{++}$
  return $\mathbf{r}/t$

For $t = 2$ iterations



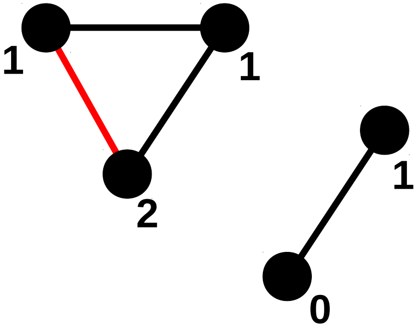
A simple algorithm

Algorithm Compute density score

---

function MKSCORE(G,t)
  Set $\mathbf{r}[i] = 0$ for each node i in G
  iterate t times
    for each edge (i, j) in G **do**
      if $\mathbf{r}[i] \leq \mathbf{r}[j]$ **then**
        $\mathbf{r}[i]{+}{+}$
      else
        $\mathbf{r}[j]{+}{+}$
  return $\mathbf{r}/t$

For $t = 2$ iterations

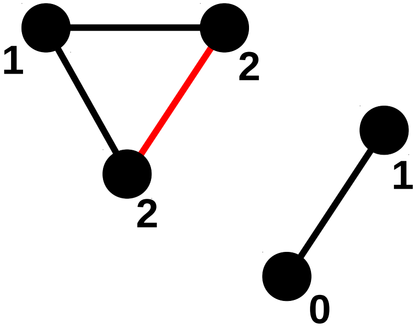


A simple algorithm

Algorithm Compute density score

```
function MKSCORE(G,t)
    Set r[i] = 0 for each node i in G
    iterate t times
        for each edge (i, j) in G do
            if r[i] ≤ r[j] then
                r[i]++
            else
                r[j]++
    return r/t
```

For $t = 2$ iterations



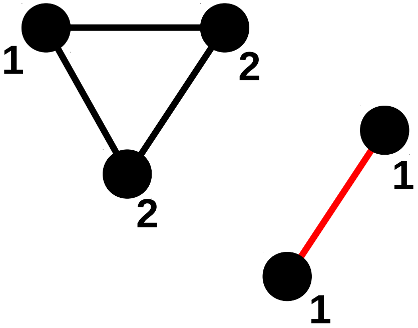
A simple algorithm

Algorithm Compute density score

---

function MKSCORE(G,t)
    Set $\mathbf{r}[i] = 0$ for each node i in G
    iterate t times
        for each edge (i, j) in G **do**
            if $\mathbf{r}[i] \leq \mathbf{r}[j]$ **then**
                $\mathbf{r}[i]{+}{+}$
            else
                $\mathbf{r}[j]{+}{+}$
    return $\mathbf{r}/t$

For $t = 2$ iterations

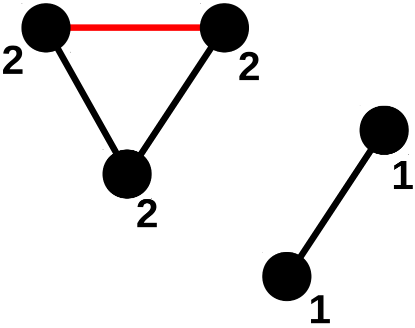


A simple algorithm

Algorithm Compute density score

function MKSCORE(G,t)
  Set $\mathbf{r}[i] = 0$ for each node i in G
  iterate t times
    for each edge (i, j) in G **do**
      if $\mathbf{r}[i] \leq \mathbf{r}[j]$ **then**
        $\mathbf{r}[i]{+}{+}$
      else
        $\mathbf{r}[j]{+}{+}$
  return $\mathbf{r}/t$

For $t = 2$ iterations

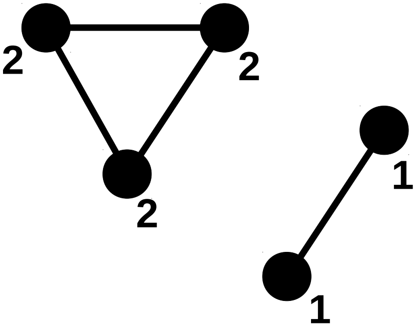


A simple algorithm

Algorithm Compute density score

```
function MKSCORE(G,t)
    Set r[i] = 0 for each node i in G
    iterate t times
        for each edge (i, j) in G do
            if r[i] ≤ r[j] then
                r[i]++
            else
                r[j]++
    return r/t
```

For $t = 2$ iterations

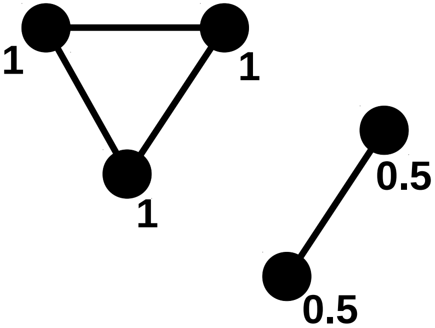


A simple algorithm

Algorithm Compute density score

```
function MKSCORE(G,t)
    Set r[i] = 0 for each node i in G
    iterate t times
        for each edge (i, j) in G do
            if r[i] ≤ r[j] then
                r[i]++
            else
                r[j]++
    return r/t
```

For $t = 2$ iterations

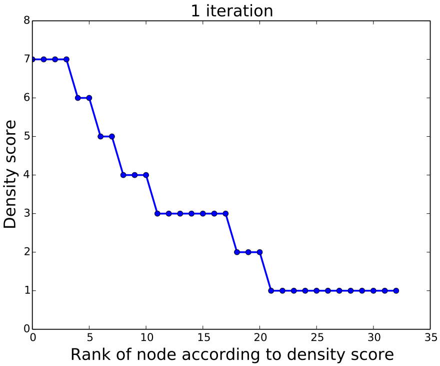


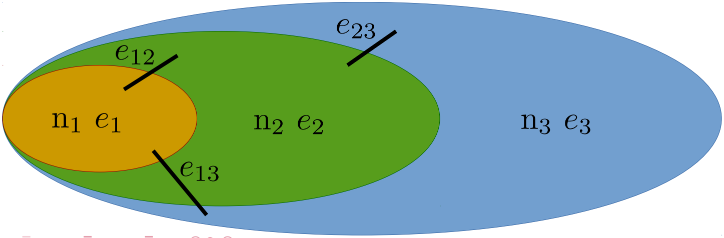
A simple algorithm

Theorems

Rank the nodes in decreasing order of density score.

- For t large enough, the nodes in the densest subgraph are ranked first.
- For t large enough, we have a density-friendly ordering.



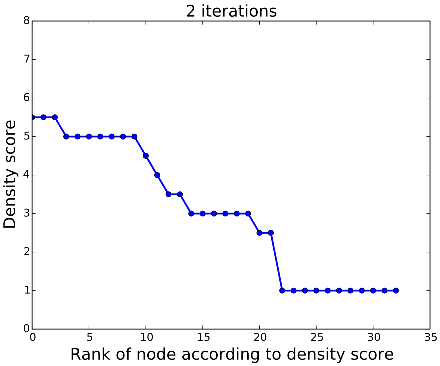


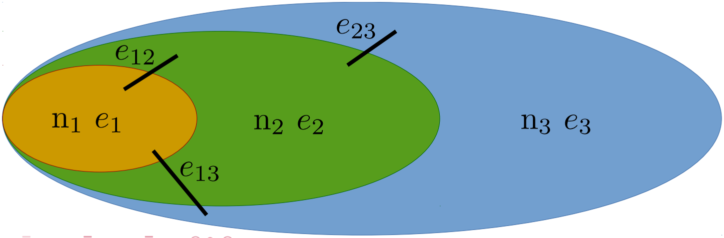
A simple algorithm

Theorems

Rank the nodes in decreasing order of density score.

- For t large enough, the nodes in the densest subgraph are ranked first.
- For t large enough, we have a density-friendly ordering.



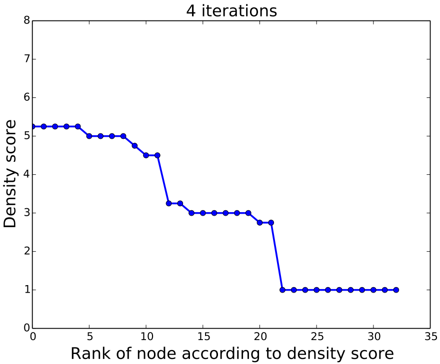


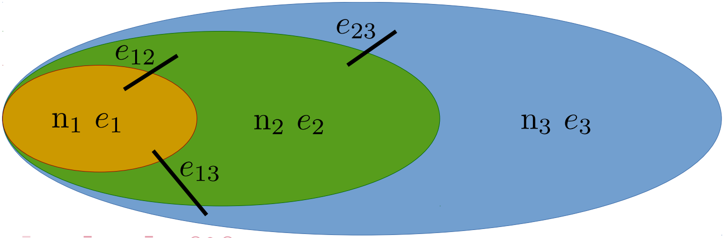
A simple algorithm

Theorems

Rank the nodes in decreasing order of density score.

- For t large enough, the nodes in the densest subgraph are ranked first.
- For t large enough, we have a density-friendly ordering.



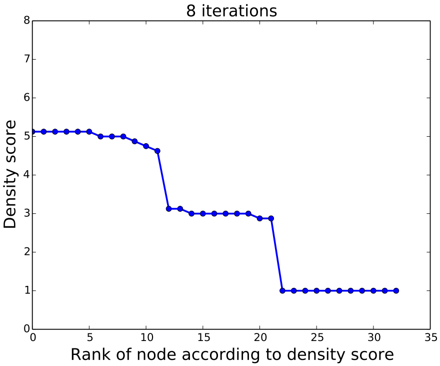


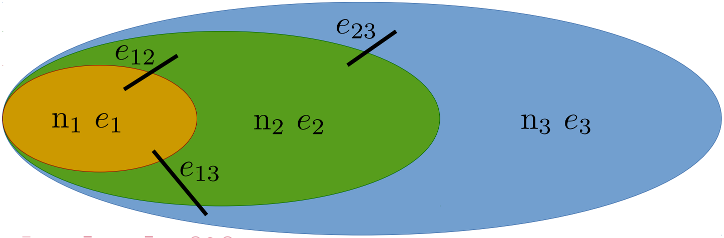
A simple algorithm

Theorems

Rank the nodes in decreasing order of density score.

- For *t* large enough, the nodes in the densest subgraph are ranked first.
- For *t* large enough, we have a density-friendly ordering.



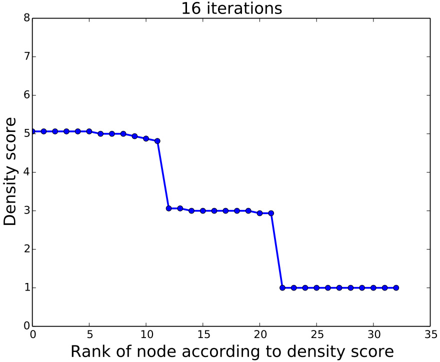


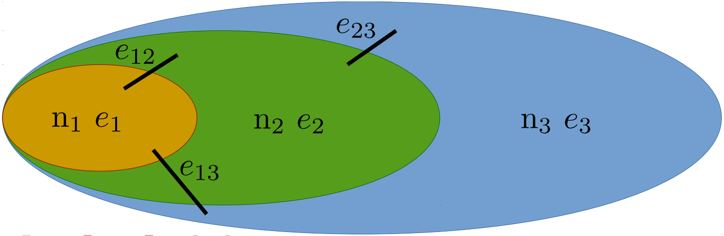
A simple algorithm

Theorems

Rank the nodes in decreasing order of density score.

- For *t* large enough, the nodes in the densest subgraph are ranked first.
- For *t* large enough, we have a density-friendly ordering.



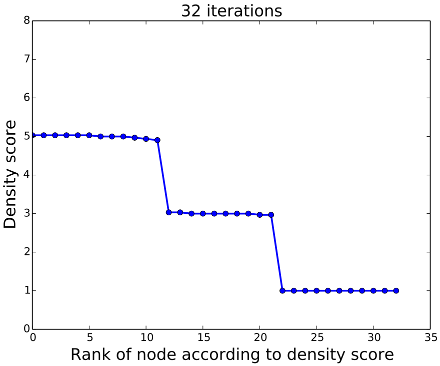


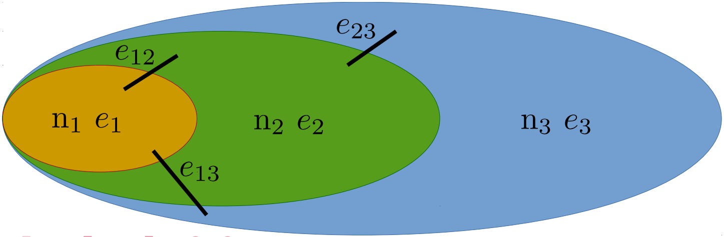
A simple algorithm

Theorems

Rank the nodes in decreasing order of density score.

- For *t* large enough, the nodes in the densest subgraph are ranked first.
- For *t* large enough, we have a density-friendly ordering.



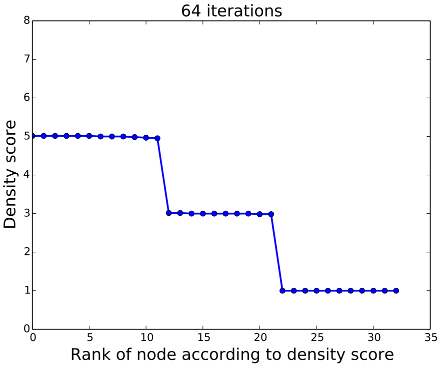


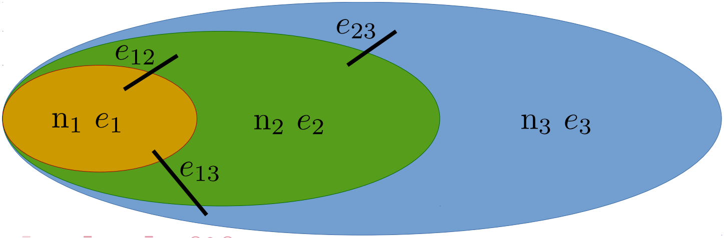
A simple algorithm

Theorems

Rank the nodes in decreasing order of density score.

- For *t* large enough, the nodes in the densest subgraph are ranked first.
- For *t* large enough, we have a density-friendly ordering.



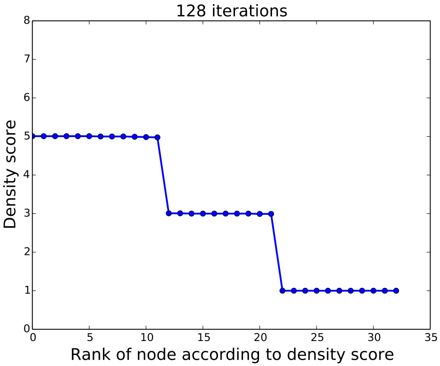


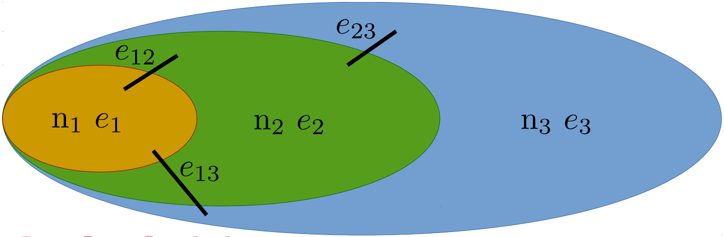
A simple algorithm

Theorems

Rank the nodes in decreasing order of density score.

- For t large enough, the nodes in the densest subgraph are ranked first.
- For t large enough, we have a density-friendly ordering.



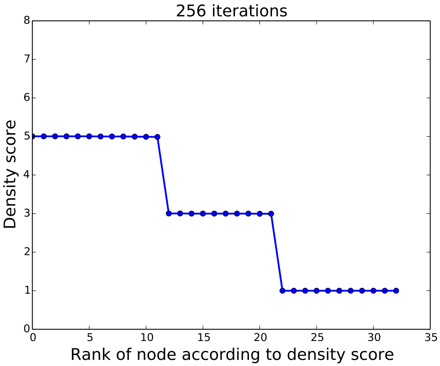


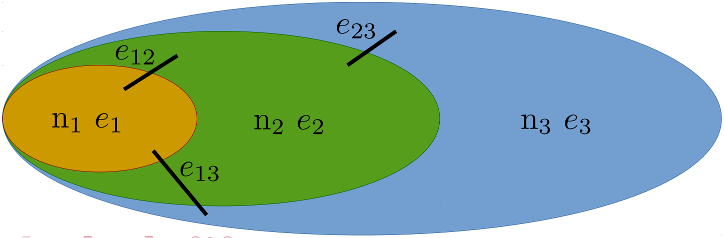
A simple algorithm

Theorems

Rank the nodes in decreasing order of density score.

- For t large enough, the nodes in the densest subgraph are ranked first.
- For t large enough, we have a density-friendly ordering.



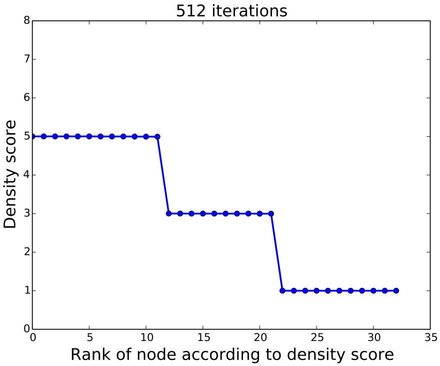


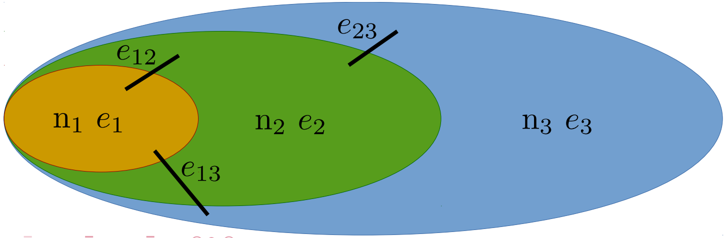
A simple algorithm

Theorems

Rank the nodes in decreasing order of density score.

- For *t* large enough, the nodes in the densest subgraph are ranked first.
- For *t* large enough, we have a density-friendly ordering.



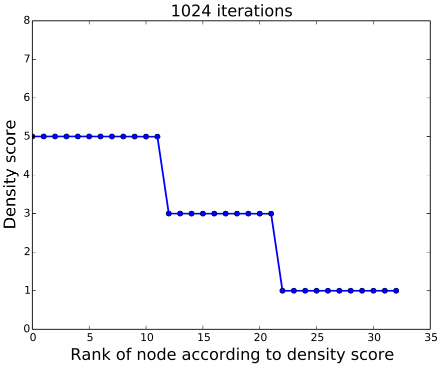


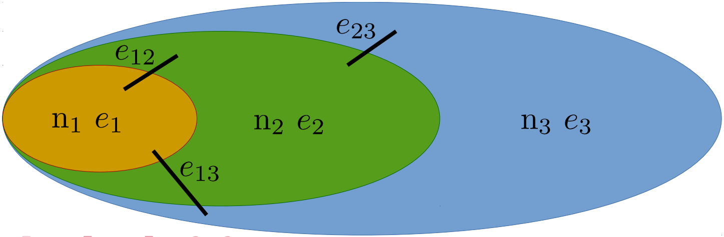
A simple algorithm

Theorems

Rank the nodes in decreasing order of density score.

- For t large enough, the nodes in the densest subgraph are ranked first.
- For t large enough, we have a density-friendly ordering.



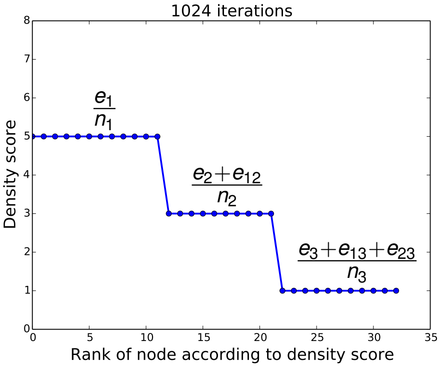


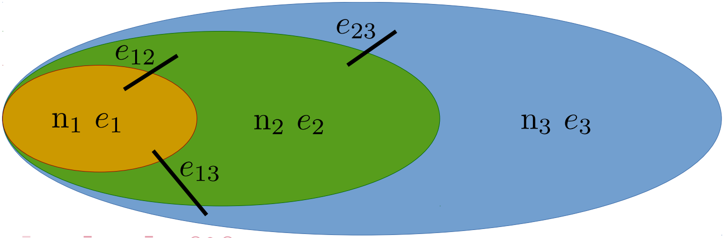
A simple algorithm

Theorems

Rank the nodes in decreasing order of density score.

- For t large enough, the nodes in the densest subgraph are ranked first.
- For t large enough, we have a density-friendly ordering.





Outline

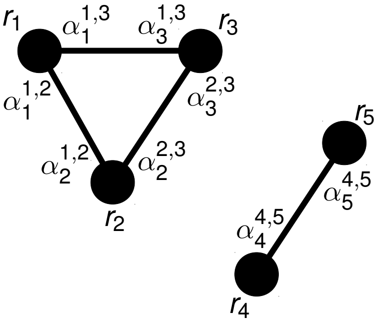
- A new and intuitive definition of density-friendly
- A very simple, yet powerful, algorithm
- **Theoretical analysis via convex programing**
- Experiments

A quadratic convex programing

Given an edge-weighted (hyper) graph $G = (V, E, w)$, we consider the following quadratic convex programing.

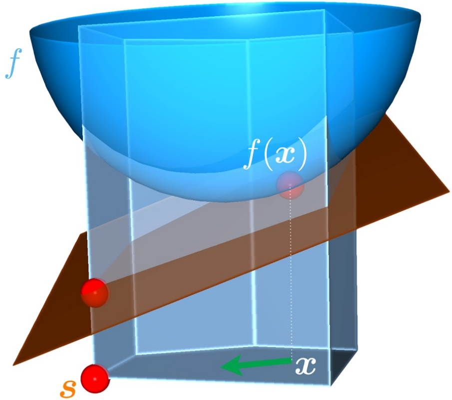
$$CP(G)$$

$$\begin{aligned}
\min \quad & \sum_{u \in V} r_u^2 \\
\text{s.t.} \quad & \forall u \in V,\ r_u = \sum_{e: u \in e} \alpha_u^e \\
& \forall e \in E,\ \sum_{u \in e} \alpha_u^e = w_e \\
& \forall u \in e \in E,\ \alpha_u^e \geq 0
\end{aligned}$$



Frank-Wolfe Algorithm

The Frank-Wolfe algorithm is a projection free gradient-descent method which has convergence guaranties for convex problems.

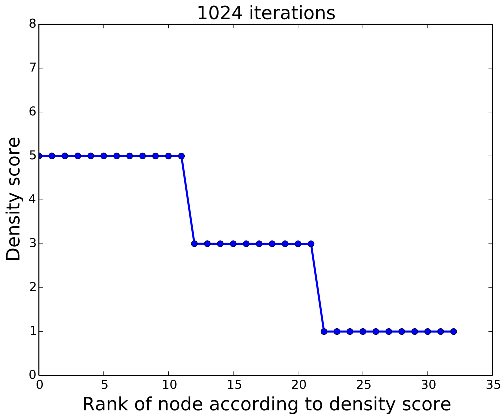


Applying Frank-Wolfe on “our quadratic convex programing” leads to an algorithm very similar to “our very simple algorithm”.

Correctness and worst case running time

Theorem (correctness)

The level sets of an optimal solution to the quadratic convex programing give the density-friendly decomposition.



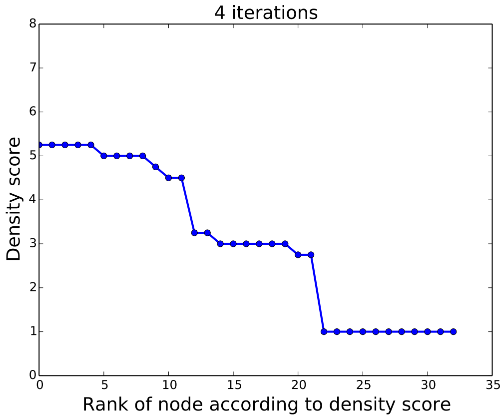
Theorem (running time)

After $t > \frac{4\Delta|E|}{\epsilon^2}$ iterations, we have $||r^{(t)} - r^G||_2 \leq \epsilon$.

Correctness and worst case running time

Theorem (correctness)

The level sets of an optimal solution to the quadratic convex programing give the density-friendly decomposition.



Theorem (running time)

After $t > \frac{4\Delta|E|}{\epsilon^2}$ iterations, we have $||r^{(t)} - r^G||_2 \leq \epsilon$.

Outline

- A new and intuitive definition of density-friendly
- A very simple, yet powerful, algorithm
- Theoretical analysis via convex programing
- **Experiments**

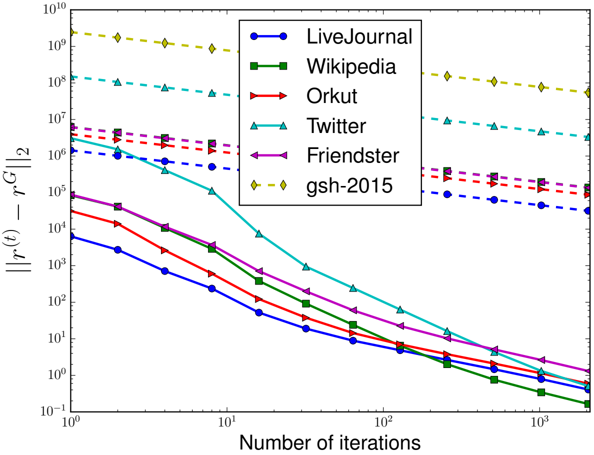
Experimental setup

TABLE : Our set of large graphs.

networks	n	m
LiveJournal	4 036 538	34 681 189
Wikipedia	2 080 370	42 336 692
Orkut	3 072 627	117 185 083
Twitter	52 579 683	1 614 106 500
Friendster	124 836 180	1 806 067 135
gsh-2015	988 490 691	25 690 705 119

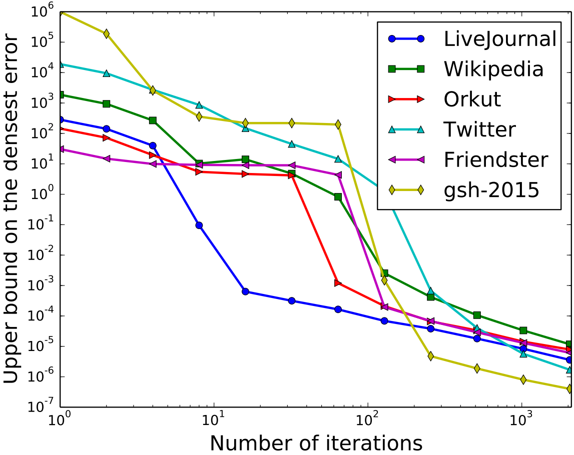
We use a machine with 64G of RAM for all networks except gsh-2015 for which we use a machine with 512G of RAM.

Convergence of the r vector



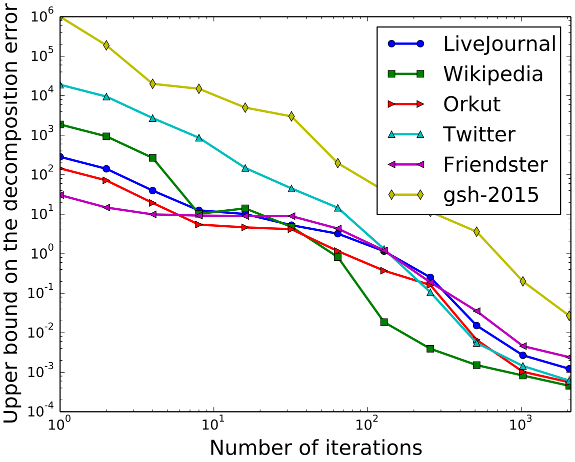
The convergence is in practice much faster than the worst case one.

Densest multiplicative error



We obtain a 10^{-3} approximation of the densest subgraph within 300 iterations on all networks.

Decomposition multiplicative error



We obtain a 10^{-2} approximation of the full decomposition within 1000 iterations on all networks except gsh-2015 (almost 10^{-1}).

Comparison to maxflow algorithm

TABLE : Running time comparison of our exact algorithm to the maxflow algorithm of Tatti and Gionis.

Networks	exact	TG15
LiveJournal	2m45s	12m02s
Wikipedia	2m14s	7m07s
Orkut	13m08s	1h02m23s
Twitter	4h57m28s	-
Friendster	5h48m27s	-

We computed the densest subgraph in gsh-2015 (25G edges) within 10 hours of computation.

Conclusion and future work

- Conclusion.
  - We gave a new and intuitive definition of the density-friendly decomposition.
  - We made an interesting link between the density-friendly decomposition and convex programing.
  - We scaled up the computation of the density-friendly decomposition using the Frank-Wolfe algorithm.
  - Code on github: `https://github.com/maxdan94`.

- Future work.
  - Density-friendly decomposition, a classic subroutine like k-core?
  - Spark/MapReduce implementation is possible.
  - Generalize our approach to other similar decompositions.
  - Investigate the potential of Frank-Wolfe for real-world graphs.

Global conclusion and future work

- We designed simple, yet powerful, algorithms leveraging the structure of real-world graphs.
- We designed algorithms to better understand the structure of real-world graphs.
- We have other work along the same lines: Branch&Bound based algorithms.
- Future work: removing "Towards" in the title of this talk.

Thank you for your attention

https://github.com/maxdan94

maximilien.danisch@telecom-paristech.fr