# *Efficient and Effective Community Search*

**Nicola Barbieri**
Yahoo Labs – London
barbieri@yahoo-inc.com

**Francesco Bonchi**
ISI Foundation & Eurecat
francesco.bonchi@acm.org

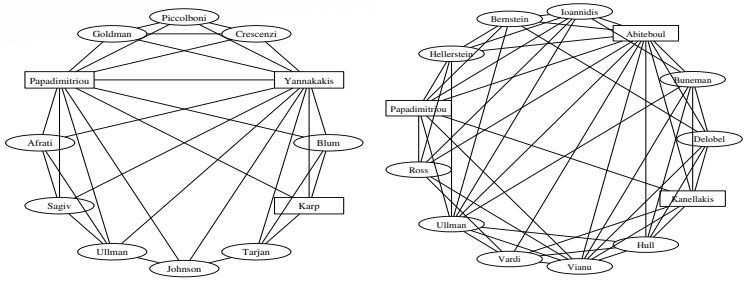
**Edoardo Galimberti**
Deloitte
edoardogalimberti1@gmail.com

**Francesco Gullo**
UniCredit R&D
francesco.gullo@unicredit.eu

YAHOO!

# Community search & applications

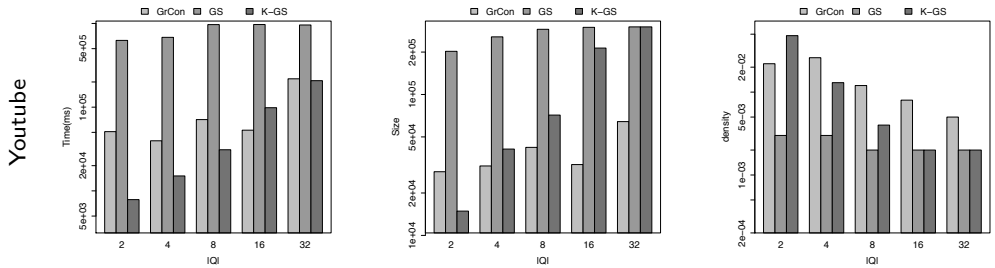
- Given a set of *query vertices* from a large graph, the ***community-search*** problem requires to find a cohesive (high density), connected sub-graph that contains the given vertices;
- It is a *query-dependent* variant of the *community detection* problem!!




Example of Community search on DBLP dataset (co authorship) [1]. The query is specified by rectangular nodes.

- ***Efficient and effective community search***

| | *efficiency* | *quality* | *multiple query vertices* | *optimality* | *parameter-free* |
|---|---|---|---|---|---|
| Global Search | + | + | yes | yes | yes |
| Constrained Global Search | + | ++ | yes | no | no |
| Local Search | ++ | ++ | no | yes | yes |
| **Our method** | +++ | +++ | yes | yes | yes |




[1] M.Sozio and A. Gionis: *The community search problem and how to plan a successful cocktail party*. In KDD 2010.

**Some applications:**

**Tag recommendation**

The graph encodes associations between tags and pictures.

a) Given a new photo and a set of initial tags, provide the user with additional tags.

b) Given some tags/keywords, recommend related pictures.

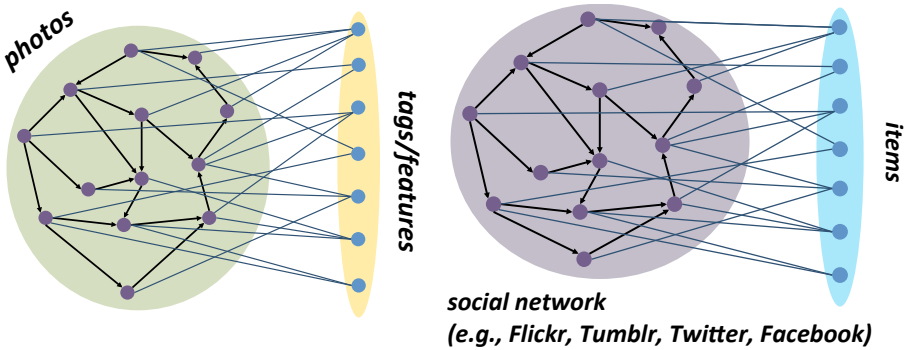
**Discovering circles in social networks**

The graph encodes friendship relationships between users. Given a user and a set of his friends, we can provide a compact visualization of their social circle.

**Marketing**

The graph encodes relationships between users and items purchased/clicked.

Given an item (i), to which users should it be recommended?



# Problem definition

We aim at finding a connected subgraph that contains all the query vertices *Q* and that maximizes a quality measure of density μ(.):

$$\textsc{Problem 1} \quad (\textsc{Community Search}).\ \textit{Given a graph } G = (V, E) \textit{ and a set of query vertices } Q \subseteq V, \textit{ find}$$

$$H^* = \underset{\substack{Q \subseteq H \subseteq V, \\ G[H] \textit{is connected}}}{\arg\max} \mu(H). \quad \square$$

As measure of density we adopt the min degree of nodes in H, $\mu(H) = \min_{u \in H} \delta_H(u)$

Intuition: by maximizing the degree of the least connected node we increase the overall density of the sub-graph!!

This choice allows us to exploit an optimal greedy algorithm, *Global Search*(GS) [1]:

- Init: $H_0$= V
- $H_i$ = iteratively remove from $H_{curr}$ the vertex with the minimum degree
- Stopping criterion: when the next candidate to be removed is a query vertex.
- Output: The process generates a series of sub-graphs. Among these, select the one with max-min-degree and where query vertices are connected.

[1] M.Sozio and A. Gionis: *The community search problem and how to plan a successful cocktail party*. In KDD 2010.

# Size-bounded Community Search

The size-bounded version of community search includes an upper bound K on the size of the output community.

This version of CSP becomes NP-Hard.

*K-Global Search* (K-GS) [1] is an heuristic approach for addressing CSP with a *soft* constraint on the desired size.

Design principle:

- Consider a new constraint on the max distance between a query vertex and each other vertex in the solution.
- A tighter distance constraint implies smaller communities.

K-GS calls GS as a sub-routine, enforcing each time a tighter distance bound until the size constraint is satisfied (or the query nodes become disconnected).

Drawbacks:

- Shortest-path distance computation
- K-GS does not guarantee optimality with respect to max-min degree
- The upper-bound on the size is a soft constraint.



[1] M.Sozio and A. Gionis: *The community search problem and how to plan a successful cocktail party*. In KDD 2010.

# Local community search

*Local search algorithm*(LS) [2] to improve the efficiency of GS.

- Iteratively expands the neighborhood of the (unique) query vertex, to retrieve a sub-graph that is guaranteed to contain an optimal solution.
- This sub-graph is used as a reduced version of the input graph to retrieve the optimal solution.

LS has been shown to achieve better efficiency than GS in practice.

Main limitation: it works only when a single query vertex is provided as input.



[2] W.Cui, Y.Xiao, H.Wang, and W.Wang*: Local search of communities in large graphs*. In SIGMOD 2014.

# Our contribution

## Efficiency

We show how it is possible to *pre-compute* and *organize* information about the structure of the graph to address CSP queries in an efficient way.

## Quality

We explicitly seek *compact* solutions to CSP queries. Hence, we study and address the problem of finding the *smallest* sub-graph that is solution to CSP.

# Addressing efficiency via k-core decomposition:

- The *k-core* of a graph is defined as the maximal sub-graph in which every vertex is connected to at least k other vertices.
- All cores are nested into each other: $V \supseteq C_1 \supseteq C_2 \supseteq \ldots \supseteq C_{k^*}$
- The k-shell is defined as $S_k = C_k \setminus C_{k+1}$
- The *core index c(u)* of a vertex u is the highest order core that contains u.
- The core decomposition can be computed in linear time [3]:
  - Remove iteratively the smallest-degree vertex and sets its core number accordingly.

**Theorem 1** *Given a graph $G = (V, E)$, its core decomposition $\mathbf{C} = \{C_1, \ldots, C_{k^*}\}$, and query vertices $Q \subseteq V$, let $C_Q^*$ be the highest-order (i.e., smallest-sized) core in $\mathbf{C}$ such that every $q \in Q$ belongs to the same connected component of $C_Q^*$. It holds that:*

1. *The connected component of $C_Q^*$ that contains $Q$ is a solution to* CSP*;*
2. *The connected component of $C_Q^*$ containing $Q$ contains all solutions to* CSP.

[3] V. Batagelj and M. Zaversnik. Fast algorithms for determining (generalized) core groups in social networks. Advances in Data Analysis and Classification.

# Retrieving solutions to CSP queries

How to organize the k-core decomposition to efficiently retrieve solutions to CSP queries?

## CoreStruct

- Store k-cores and connected components within a k-core.
- Replication of information.

## ShellStruct

- Store k-shells in a hierarchical way.
  - Each node at the level (k) represents a connected component within the k-shell.
  - Links between nodes in different level represent connected component that are merged.

| | Building-time | Building: Space | Retrieval |
|---|---|---|---|
| **CoreStruct** | O(h*n+m) | O(h*n) | O(\|Q\|*log h) |
| **ShellStruct** | O(h*n+m) | O(h*M) | O(\|Q\|*h+n) |

h #distinct k-cores

M Max #connected components in a core

# The Minimum Community Search problem

- Our goal is to further refine the solution H* provided by the k-core structure to extract a sub-graph that is still optimal and as small as possible.

**Problem 2** (**MINIMUM COMMUNITY SEARCH (MIN-CSP)**) Given a graph $G = (V, E)$, a set of query vertices $Q \subseteq V$, let $H^* \subseteq V$ be the subgraph of $G$ containing all the solutions to CSP for $Q$. Find

$$H^*_{min} = \underset{\substack{Q \subseteq H \subseteq H^*: \\ G[H] \text{is connected}, \\ \mu(H) \geq \mu(H^*)}}{\arg\min} |H|. \qquad \square$$

- MIN-CSP is NP-hard

# Heuristic approaches to Min-SCP

Given a CSP query Q, retrieve the solution H* from the k-core structure.
Start with $H^*_{min}$= Q and add vertices until $H^*_{min}$ is optimal.

## Greedy

- Start from query vertices and add nodes by applying greedy selection on the score: $p(u) = \langle p'(u), p''(u) \rangle$
- p'(u) is the number of distinct connected components that will be merged by adding (u) to the current solution.
- p''(u) is focuses on satisfying the optimality of min-degree:
  - #neighbors of (u) in $H^*_{min}$ that will satisfy the optimal degree.
  - How far is (u) from satisfying the optimal degree?

## Connection

- Build a Steiner tree on Q by exploring H*;
- Iteratively add more nodes according to score p''(u) until all vertices satisfy the condition on the degree.

## GreedyConnection:

- Run Greedy and Connection *sequentially.*

# Experimental evaluation

- We compare *GS*, *K-GS*, *LS* and *GrCon* on real-world datasets.
- Evaluation focuses on 3 dimensions: time, size, density.
- Sample 100 sets of query vertices at random, avg the results.

| | $n$ | $m$ | $h$ | type |
|---|---|---|---|---|
| Email | $33,696$ | $180,811$ | $43$ | Communication network |
| Web-NotreDame | $325,729$ | $1,090,108$ | $41$ | Web graph |
| Web-Google | $791,822$ | $3,815,994$ | $40$ | Web graph |
| Youtube | $1,134,889$ | $2,987,623$ | $51$ | Social network |
| Flickr | $1,624,992$ | $15,476,835$ | $567$ | Social network |

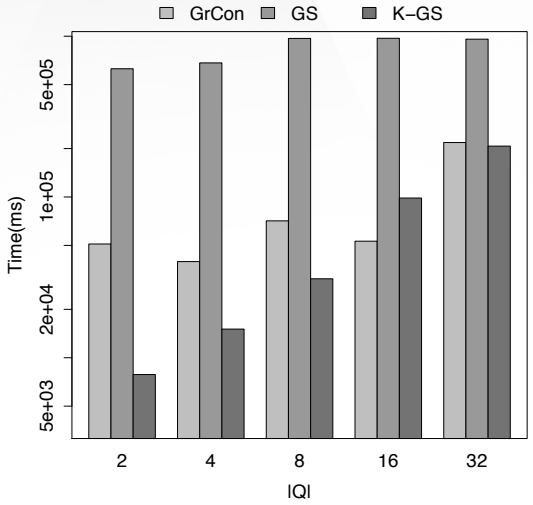
# Preprocessing-phase & Retrieval

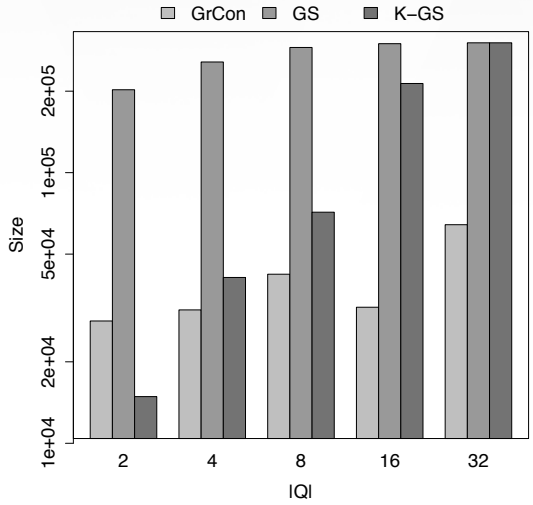
| | *building time* (s) | | *space* (MB) | |
|---|---|---|---|---|
| | CoreStruct | ShellStruct | CoreStruct | ShellStruct |
| Email | 1 | 3 | 7 | 4 |
| Web-ND | 12 | 574 | 41 | 28 |
| Web-G | 43 | 3,155 | 147 | 90 |
| Youtube | 201 | 5,946 | 130 | 100 |
| Flickr | 1,765 | 15,865 | 564 | 347 |

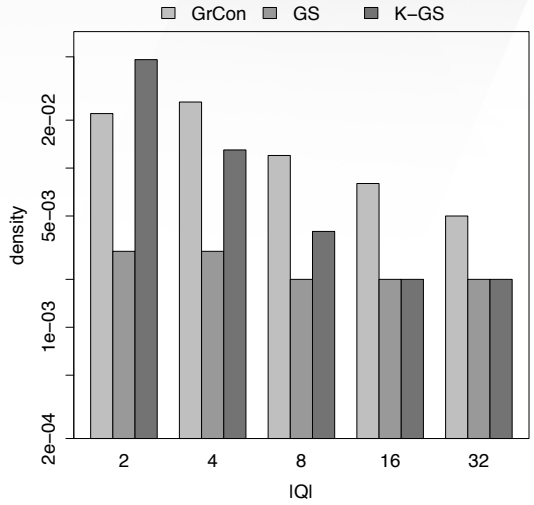
| | $\|Q\| = 1$ | | $\|Q\| = 2$ | | $\|Q\| = 4$ | | $\|Q\| = 8$ | | $\|Q\| = 16$ | | $\|Q\| = 32$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Core Struct | Shell Struct | Core Struct | Shell Struct | Core Struct | Shell Struct | Core Struct | Shell Struct | Core Struct | Shell Struct | Core Struct | Shell Struct |
| Email | 14 | 17 | 18 | 48 | 26 | 115 | 41 | 284 | 69 | 361 | 121 | 153 |
| Web-ND | 17 | 16 | 18 | 50 | 26 | 106 | 42 | 232 | 76 | 227 | 144 | 420 |
| Web-G | 36 | 19 | 20 | 69 | 30 | 169 | 49 | 242 | 115 | 311 | 197 | 350 |
| Youtube | 13 | 17 | 19 | 39 | 27 | 81 | 47 | 198 | 77 | 335 | 137 | 338 |
| Flickr | 13 | 18 | 18 | 57 | 26 | 128 | 40 | 275 | 68 | 485 | 123 | 129 |

# Evaluation: multiple-vertex queries

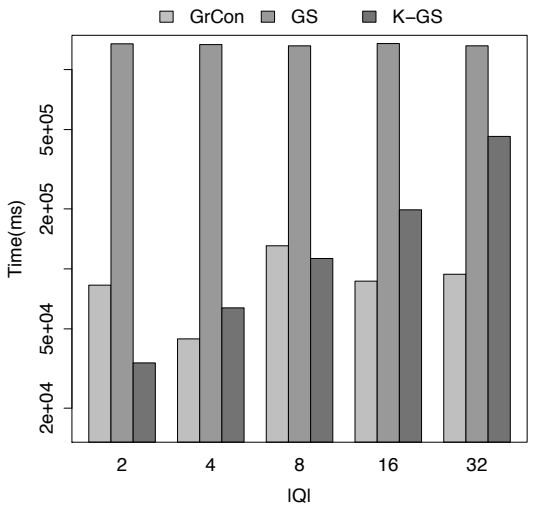
Youtube

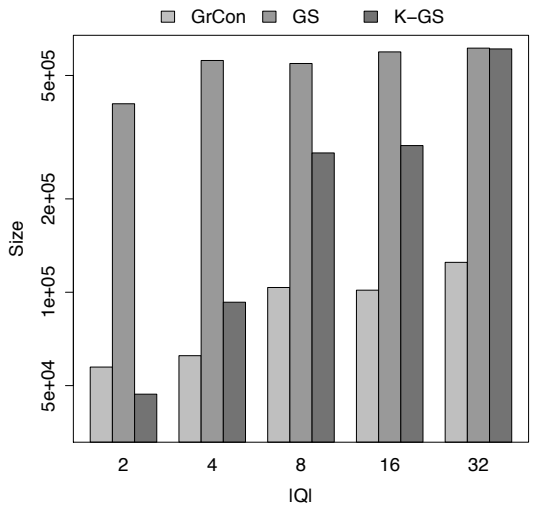


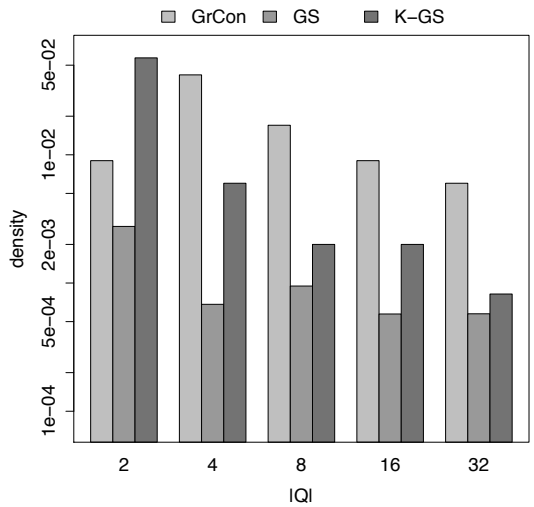




Flickr

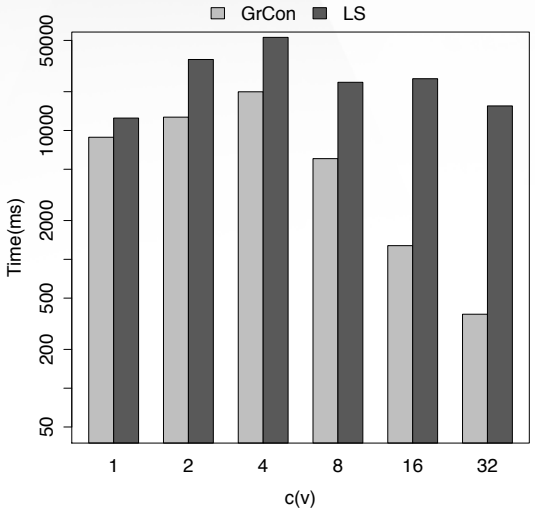


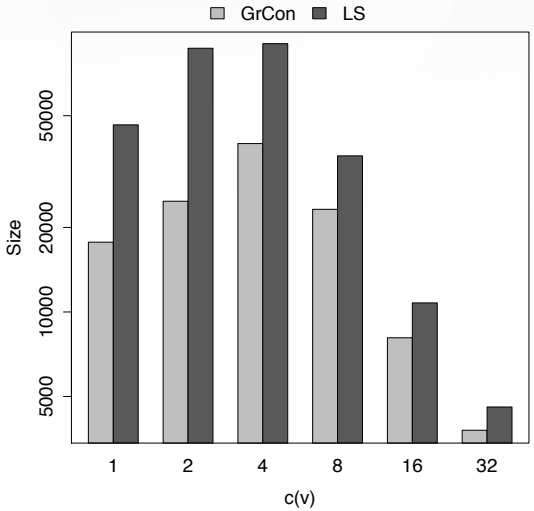



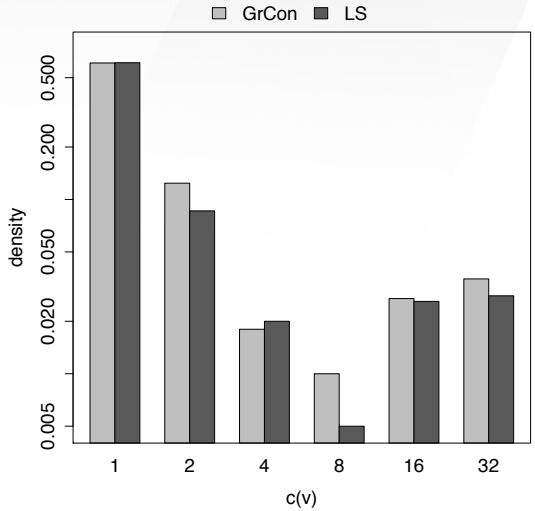



# Evaluation: one-vertex queries

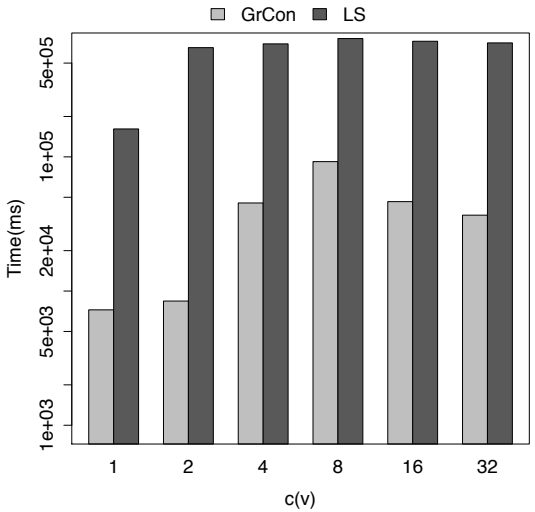
Youtube

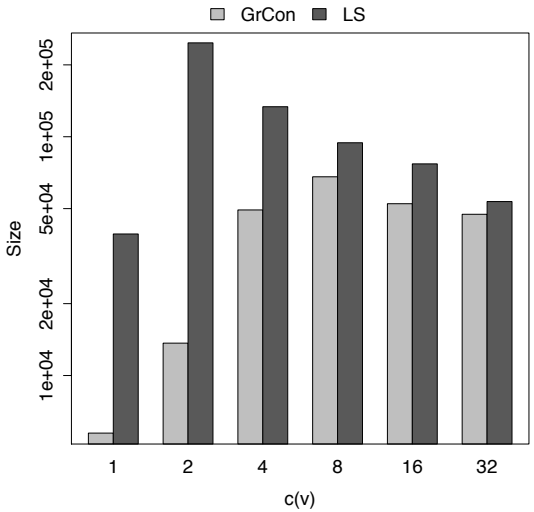


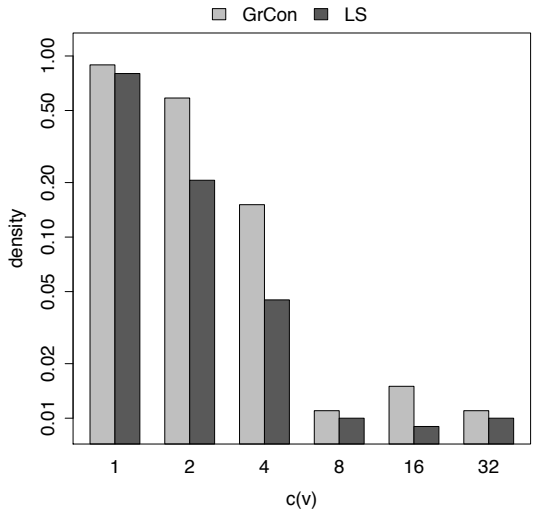




Flickr



# Summary

| | *efficiency* | *quality* | *multiple query vertices* | *optimality* | *parameter-free* |
|---|---|---|---|---|---|
| Global Search | + | + | yes | yes | yes |
| Constrained Global Search | + | ++ | yes | no | no |
| Local Search | ++ | ++ | no | yes | yes |
| **Our method** | +++ | +++ | yes | yes | yes |

Source code and datasets are available online!!

# Future work

- The minimum Weiner connector problem, Sigmod 2015.
- CSP problem on heterogeneous graphs, with constraints on admissible labels of nodes/edges.

Thank you!