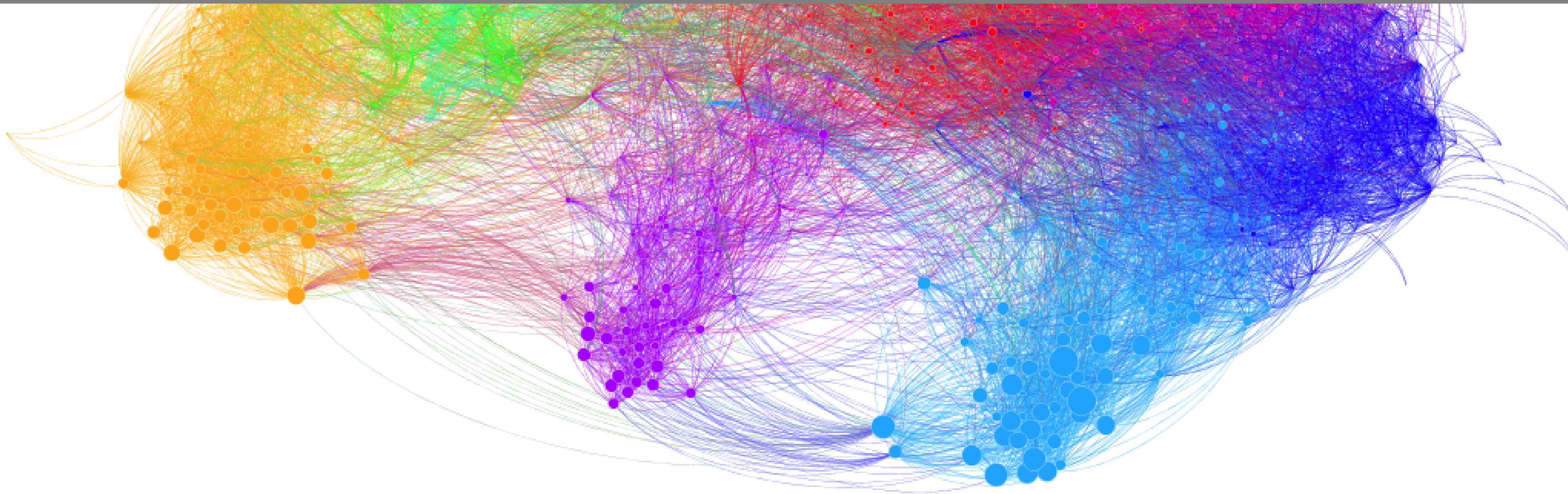
Karlsruhe Institute of Technology

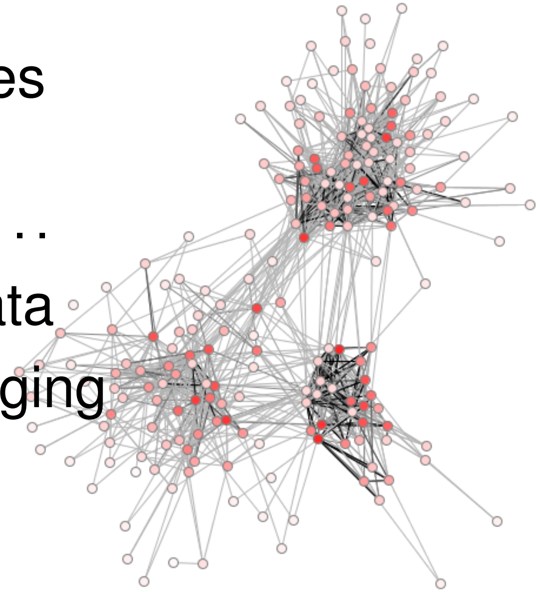
Engineering High-Performance Community Detection Heuristics for Massive Graphs

Christian L. Staudt and Henning Meyerhenke

INSTITUTE OF THEORETICAL INFORMATICS · PARALLEL COMPUTING GROUP



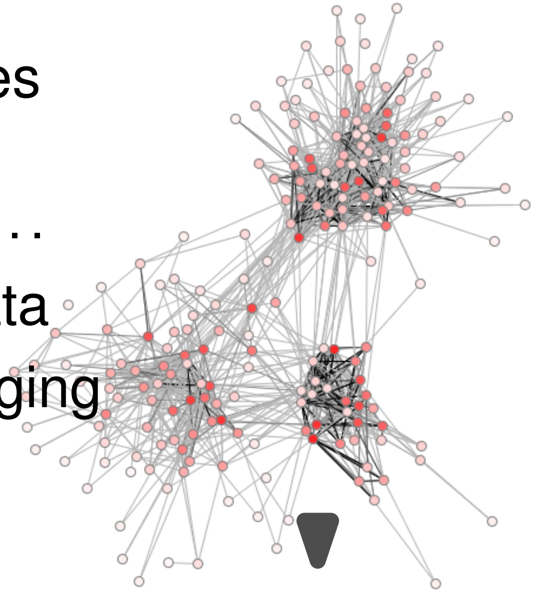
Big Network Data



- proliferation of large networks and high data rates
  - e.g. WWW ($>$ 30 billion pages), online social networks ($>$ 600 million active users),...
- analysts need information from these piles of data
- **complex networks** are computationally challenging
  - **scale-free** topology $\rightarrow$ load balancing issues
  - **small-world** network $\rightarrow$ cache performance issues

Big Network Data

- proliferation of large networks and high data rates
  - e.g. WWW ($>$ 30 billion pages),
    online social networks ($>$ 600 million active users),...
- analysts need information from these piles of data
- **complex networks** are computationally challenging
  - **scale-free** topology $\rightarrow$ load balancing issues
  - **small-world** network $\rightarrow$ cache performance issues



Community Detection

- find **internally dense, externally sparse subgraphs** (formalized: e.g. **modularity**)
- goals: uncover community structure, prepartition network

[survey: Schaeffer 07, Fortunato 10]

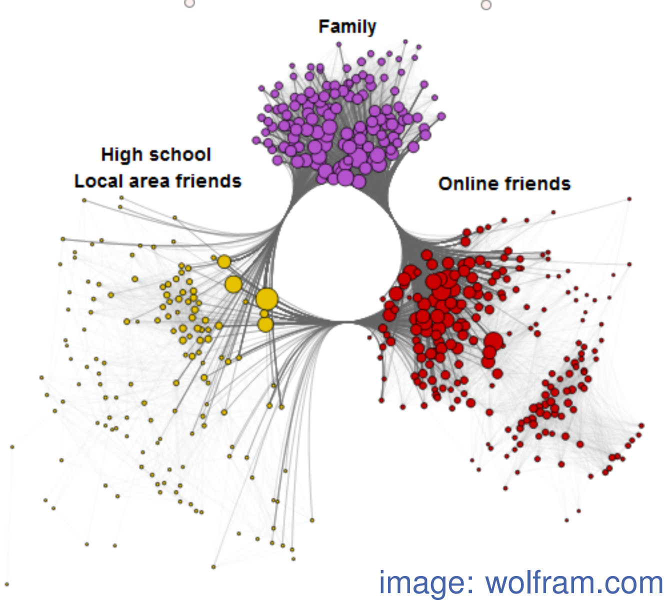


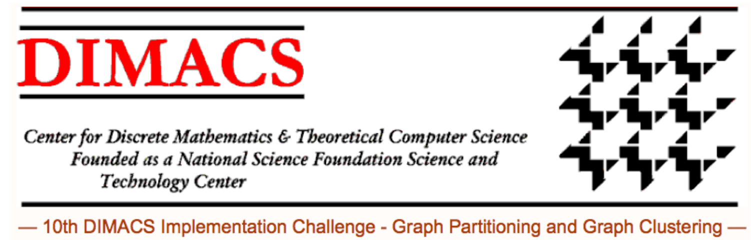
image: wolfram.com

Related Work | State of the Art

Karlsruhe Institute of Technology

Challenge

10th DIMACS Implementation Challenge
- Graph Partitioning and Clustering



- criteria: time and quality (**modularity**)
- high-quality solutions
  - **RG**, a randomized greedy agglomerative algorithm
  - **CGGC**, an ensemble using **RG** [Ovelgönne & Geyer-Schulz 13]
- large variance in running time among contestants
- few relied on parallelism
  - **CLU_TBB**, a parallel agglomerative algorithm [Fagginger Auer, Bisseling 13]
- few could handle largest graphs (billions of edges)

Challenge

10th DIMACS Implementation Challenge
- Graph Partitioning and Clustering

- criteria: time and quality (**modularity**)
- high-quality solutions
  - **RG**, a randomized greedy agglomerative algorithm
  - **CGGC**, an ensemble using **RG** [Ovelgönne & Geyer-Schulz 13]
- large variance in running time among contestants
- few relied on parallelism
  - **CLU_TBB**, a parallel agglomerative algorithm [Fagginger Auer, Bisseling 13]
- few could handle largest graphs (billions of edges)

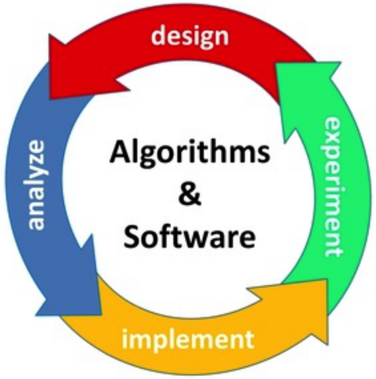
Others

- original label propagation algorithm [Raghavan et al. 07]
- distributed parallel label propagation on Hadoop [Ovelgönne 12]

Contribution | Methods & Capabilities

Requirements

- only nearly **linear time** algorithms are practical
- we need to take advantage of **parallelism**



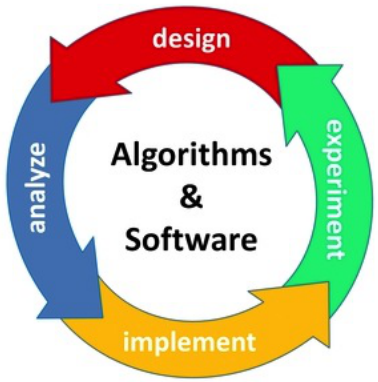
Our Approach

- algorithm engineering
- a framework of **shared-memory parallel heuristics**
  - **PLP**: a **label propagation** algorithm
  - **PLM**: a parallelization of a locally greedy **modularity maximizer**
  - **PLMR**: **PLM** with additional refinement
  - **EPP**: an **ensemble technique**

Contribution | Methods & Capabilities

Requirements

- only nearly **linear time** algorithms are practical
- we need to take advantage of **parallelism**



Our Approach

- algorithm engineering
- a framework of **shared-memory parallel heuristics**
  - **PLP**: a **label propagation** algorithm
  - **PLM**: a parallelization of a locally greedy **modularity maximizer**
  - **PLMR**: **PLM** with additional refinement
  - **EPP**: an **ensemble technique**

Capabilities

- **PLM** first parallel variant of **Louvain method**, optional refinement phase
- data rates approach 50M edges/sec, depending on algorithm
- **NetworKit**: a framework for high-performance network analysis

Basics | Modularity

- Objective function **modularity**:

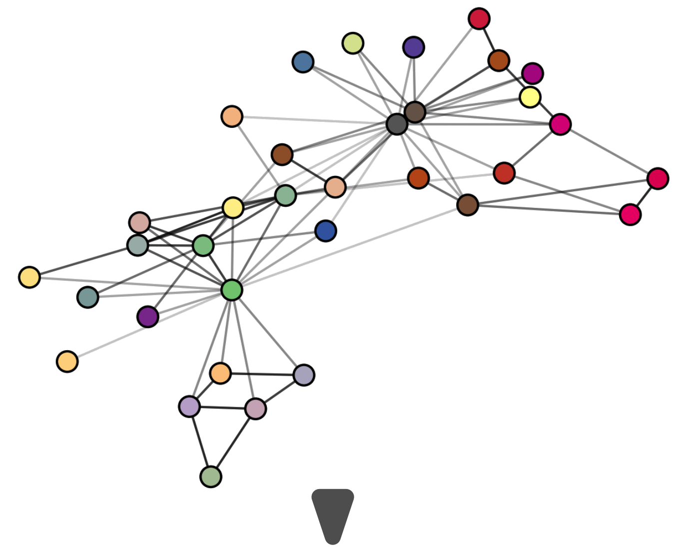
$$q(\mathcal{C}) = \sum_{C \in \mathcal{C}} \left(\frac{|E(C)|}{m} - \left(\frac{\sum_{v \in C} deg(v)}{2m} \right)^2 \right)$$

- Expected deviation from random graph with the same degree sequence
- **NP-hard** to optimize for modularity [Brandes et al., IEEE TKDE 2008] and most other (interesting) objective functions
- Modularity has some **known issues** (resolution limit, ...), some can be circumvented
- Still the **most popular clustering metric** in network analysis

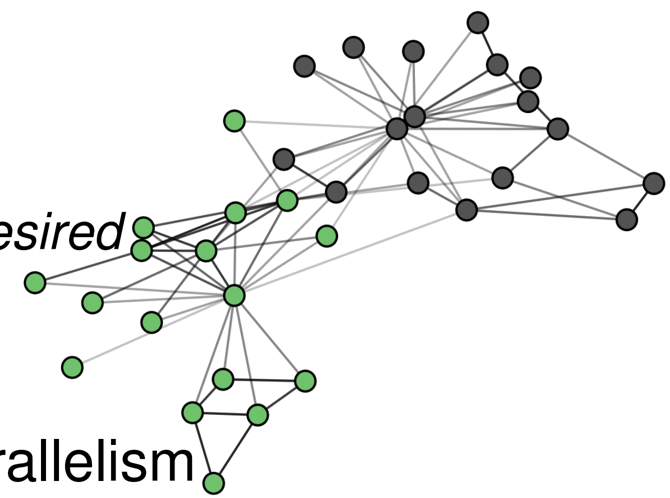
1. Introduction
2. Algorithms
3. Experiments
4. **NetworKit**
5. Conclusions

Parallel Label Propagation **PLP**

initialize nodes with unique labels
while *labels not stable* **do**
  parallel for $v \in V$
    adopt dominant label in $N(v)$
  endfor
end
return *communities from labels*



- communities from labelling of node set
- dense subgraphs agree on common label
  $\rightarrow$ stable distribution emerges
- a local coverage maximizer
  - getting stuck in local optima of coverage is *desired*
    $\rightarrow$ modularity implicitly maximized
- $O(m)$ time per iteration, few iterations
- purely local updates $\rightarrow$ high degree of parallelism



[original, sequential algorithm: Raghavan et al. 07]

- adapted to weighted graphs
- **optimizations**
  - active nodes: evaluate *v* only if labels in *N*(*v*) change
  - truncated iterations: stop if only few nodes undecided
- **OpenMP** parallelization
  - better load balancing with `parallel for schedule(guided)` (high-degree nodes)

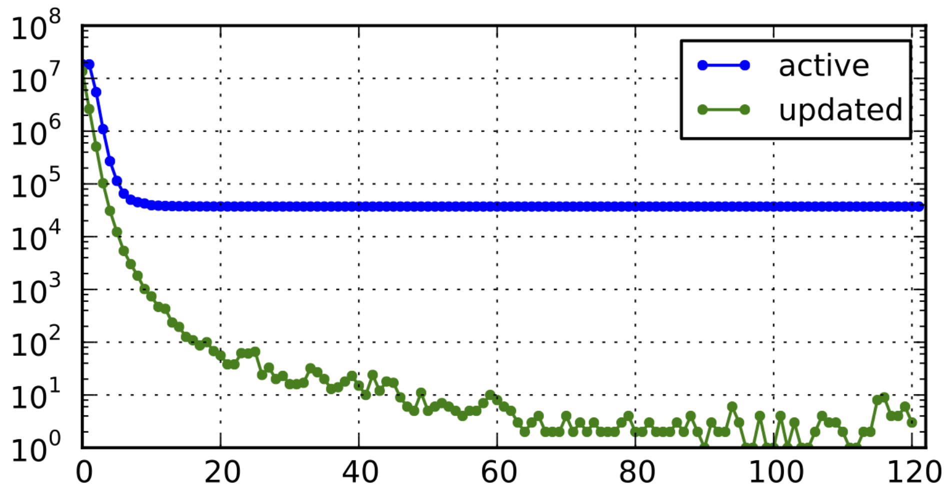


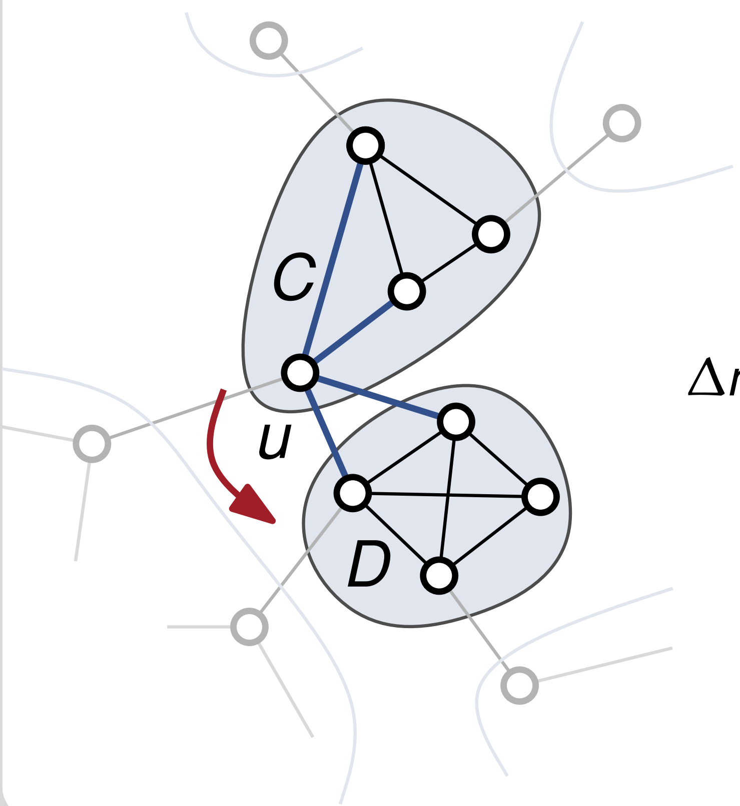
Figure: Number of active and updated nodes per iteration of **PLP** on a large web graph

Parallel Louvain Method **PLM**

- a locally greedy modularity maximizer
  - repeatedly move nodes to neighbor communities
  - coarsen the graph and repeat
- sequential algorithm: well known method for efficiently achieving high modularity values [Blondel et al. 08]
- our parallel design

```
initialize to singletons
move nodes for modularity gain
if communities changed then
    coarsen graph
    recursively apply PLM
    prolong communities
end
return communities
```

- challenge: evaluate and perform node moves in parallel
  - store and update some interim values for Δmod
  - updates need to be protected by locks
- parallel moves may be based on stale values, but self-correction possible



$$\Delta mod(u,\ C \to D) = \overbrace{\frac{\omega(u, D \setminus \{u\}) - \omega(u, C \setminus \{u\})}{\omega(E)}}^{\text{coverage}} + \underbrace{\frac{(vol(C \setminus \{u\}) - vol(D \setminus \{u\})) \cdot vol(u)}{2 \cdot \omega(E)^2}}_{\text{expected coverage}}$$

- add a refinement phase on every level
  - $\longrightarrow$ additional opportunities for modularity improvement at the cost of more iterations

```
initialize to singletons
move nodes for modularity gain
if communities changed then
    coarsen graph
    recursively apply PLMR
    prolong communities
    move nodes for modularity gain
end
return communities
```

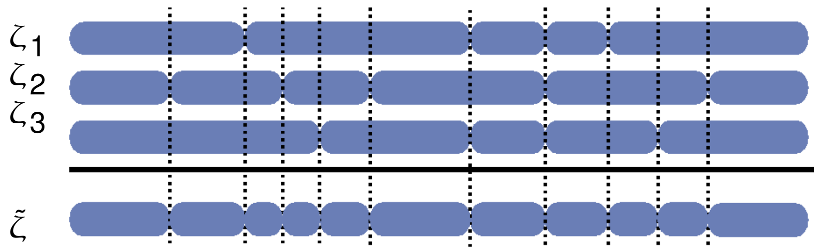
- **ensemble learning**: combine multiple weak classifiers to form a strong one
- generic scheme with exchangeable base and final algorithm [e.g. Ovelgönne & Geyer-Schulz 13]

1. ensemble of base algorithms operate on input graph independently
2. consensus solution is formed and graph coarsened accordingly
3. final algorithm operates on coarsened graph

parallel for Base *in ensemble*
  | $\zeta_i \leftarrow$ **Base**$_i(G)$
endfor
$\bar{\zeta} \leftarrow$ **consensus**$(\zeta_1, \ldots, \zeta_b)$
$G^1 \leftarrow$ **coarsen**$(G, \bar{\zeta})$
$\zeta^1 \leftarrow$ **Final**(G^1)
$\zeta \leftarrow$ **prolong**(ζ^1, G)
return ζ

EPP Implementation

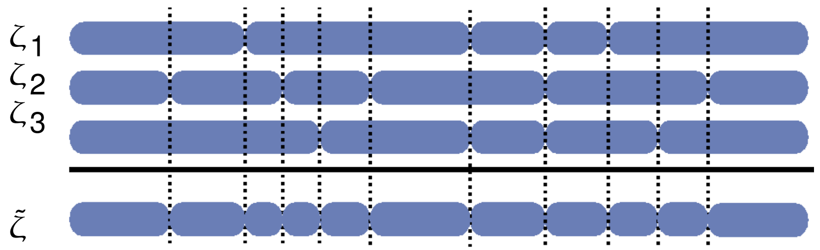
- nested parallelism in the ensemble
- efficiently calculate consensus communities through *k*-**way hashing** of community IDs
- base algorithm: focus on speed
- final algorithm: focus on quality optimization



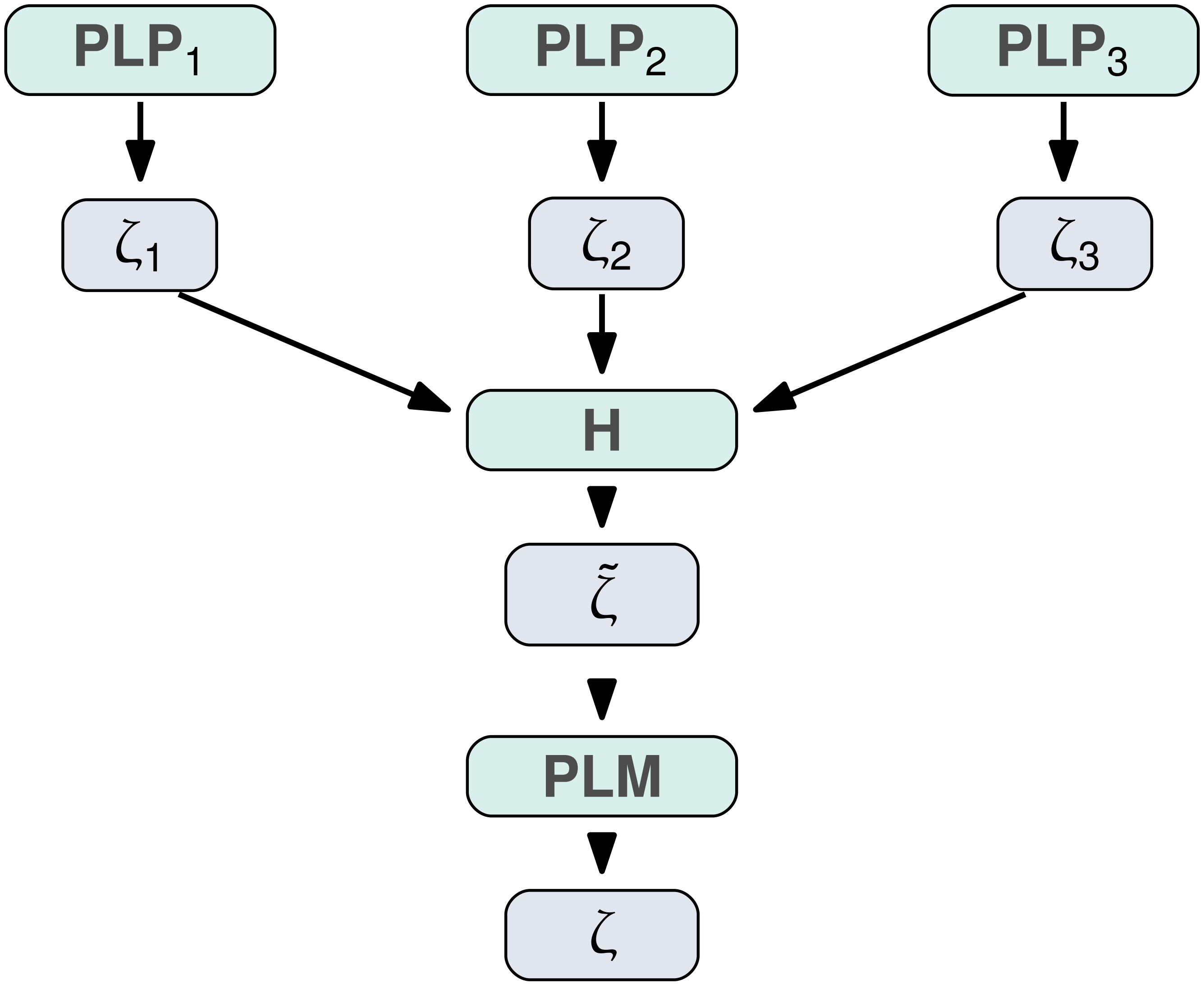
consensus communities image: Ovelgönne & Geyer-Schulz 13

EPP Implementation

- nested parallelism in the ensemble
- efficiently calculate consensus communities through *k*-**way hashing** of community IDs
- base algorithm: focus on speed $\rightarrow$ **PLP**
- final algorithm: focus on quality optimization $\rightarrow$ **PLM**



consensus communities image: Ovelgönne & Geyer-Schulz 13


PLP$_1$
PLP$_2$
PLP$_3$
ζ_1
ζ_2
ζ_3
H
$\tilde{\zeta}$
PLM
ζ

1. Introduction
2. Algorithms
3. Experiments
4. **NetworKit**
5. Conclusions

- variety of real-world and synthetic data sets
- **complex networks**: web graphs, internet topology, online social network, scientific collaboration, ...
- **Stochastic Kronecker Graphs (SKG)** for scaling experiments

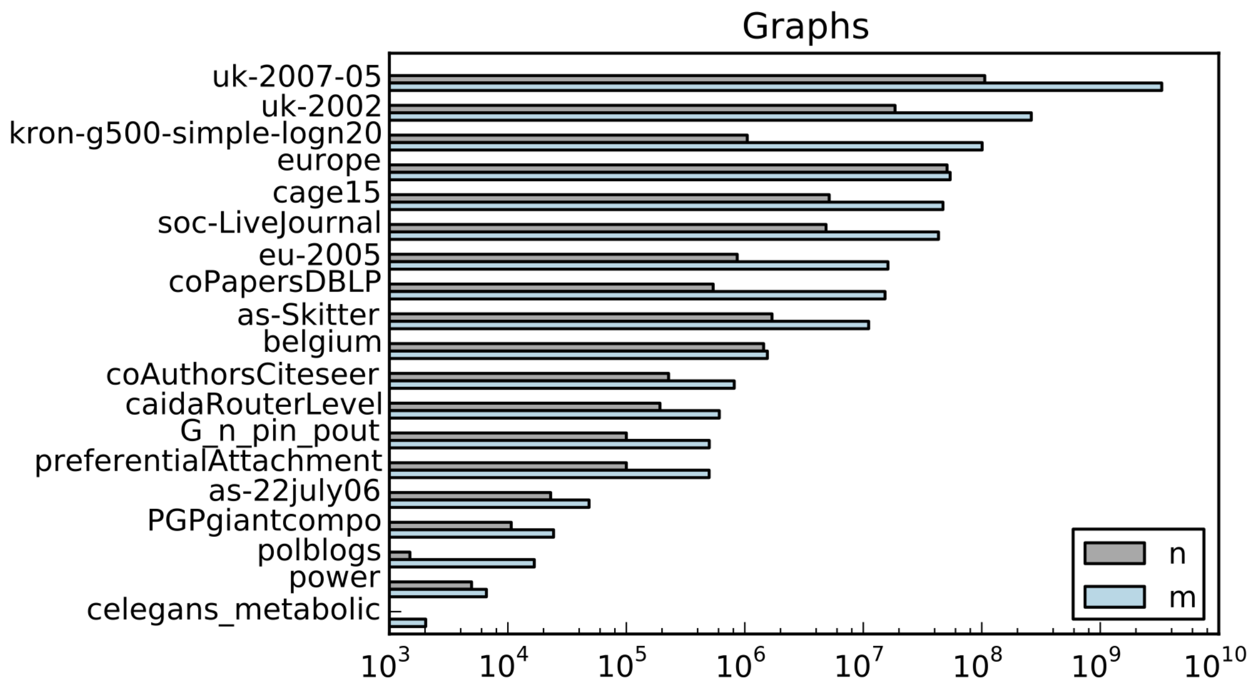


Figure: size comparison of test graphs

	phipute1.iti.kit.edu
compiler	gcc 4.7.1
CPU	2 x 8 Cores: Intel(R) Xeon(R) E5-2680 0 @ 2.70GHz, 32 threads
RAM	256 GB
OS	SUSE 12.2-64

- handles large graphs easily
  - 3.3 billion edge web graph in 60 s with 32 threads
- reasonable modularity values (but room for improvement)

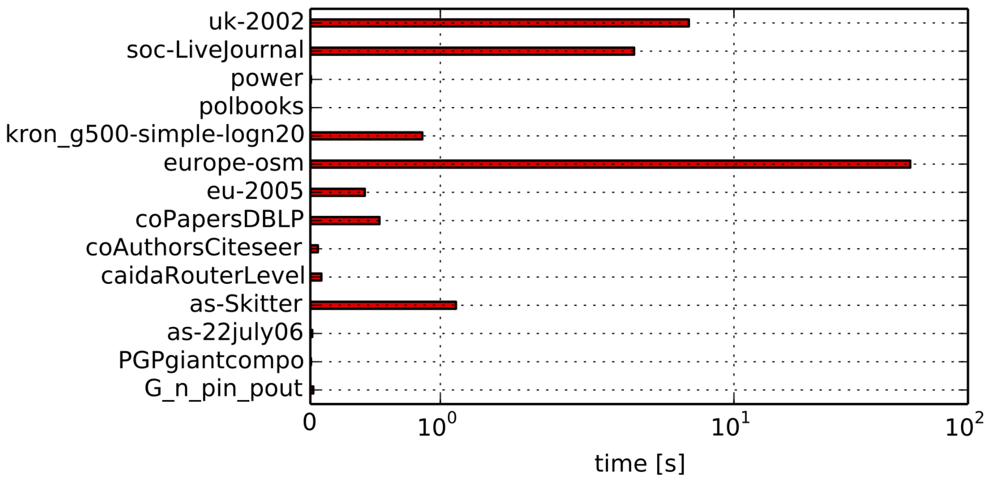


Figure: running time [s] for various networks

- handles large graphs easily
  - 3.3 billion edge web graph in 60 s with 32 threads
- reasonable modularity values (but room for improvement)

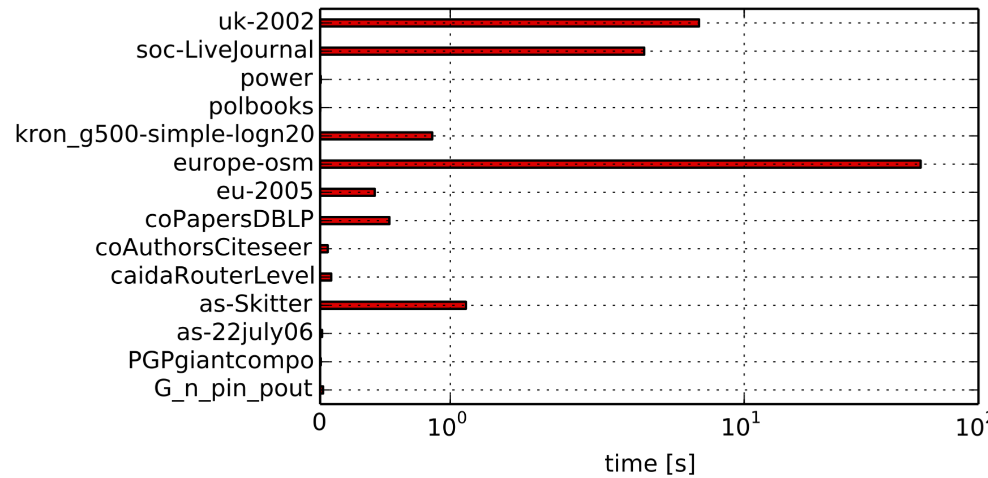


Figure: running time [s] for various networks

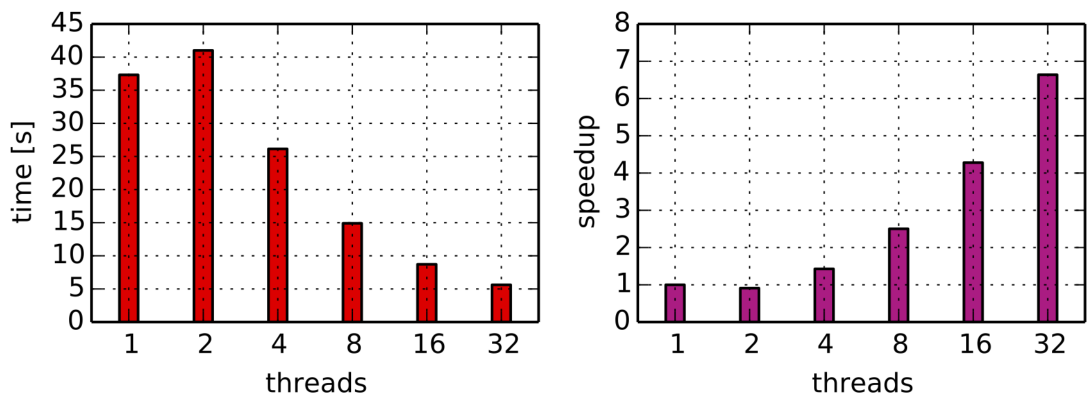


Figure: strong scaling of **PLP** on 250M edge web graph

- Considering the complex input, **PLP** scales well from 2 to 32 threads

Results | PLM

- only minor differences in solution quality between sequential and parallel versions
  - **PLM** able to correct undesirable decisions due to stale data
- better modularity than **PLP** (ca. 0.1)
- but slower (ca. factor 10)
- scaling: worse than **PLP** mainly because of sequential coarsening

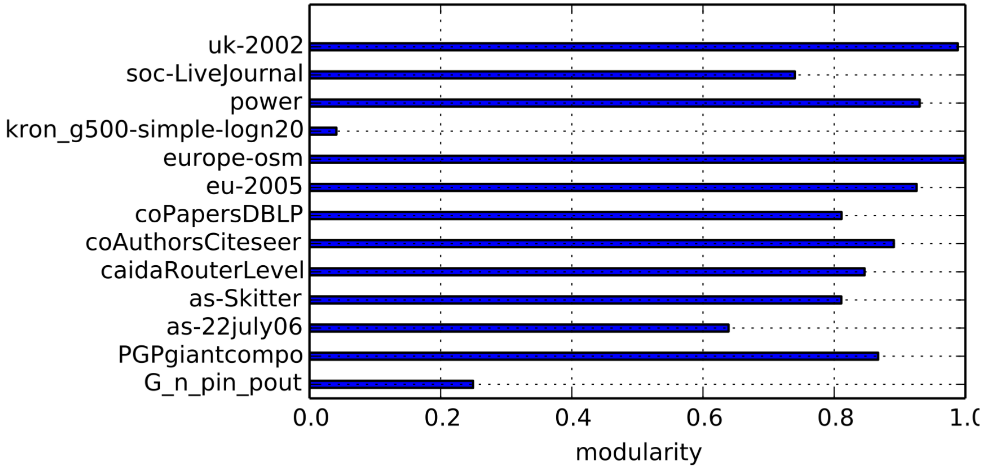


Figure: modularity for **PLM**

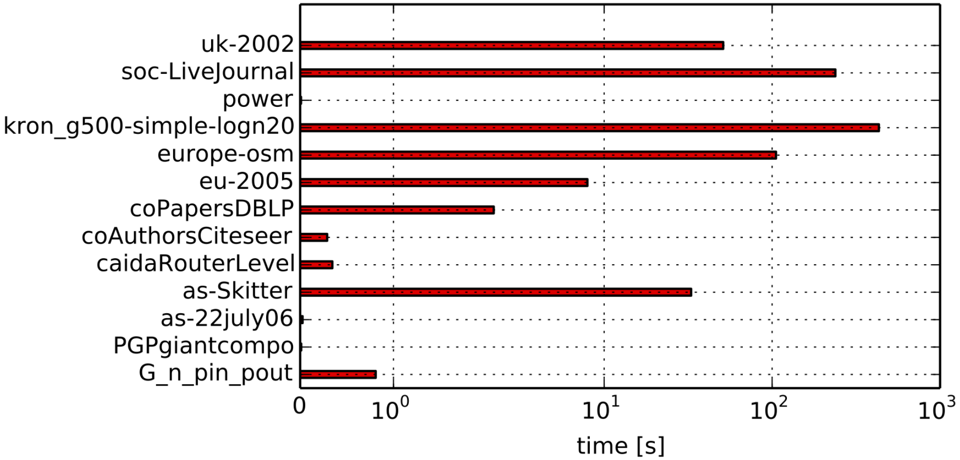


Figure: time for **PLM**

PLMR

- refinement phase gives small quality boost at the cost of a few more iterations

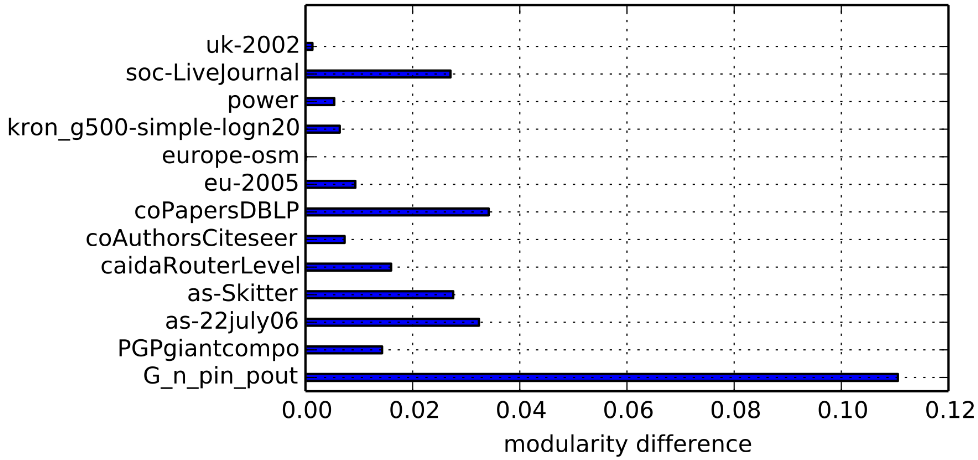


Figure: modularity improvement of **PLMR** compared to **PLM**

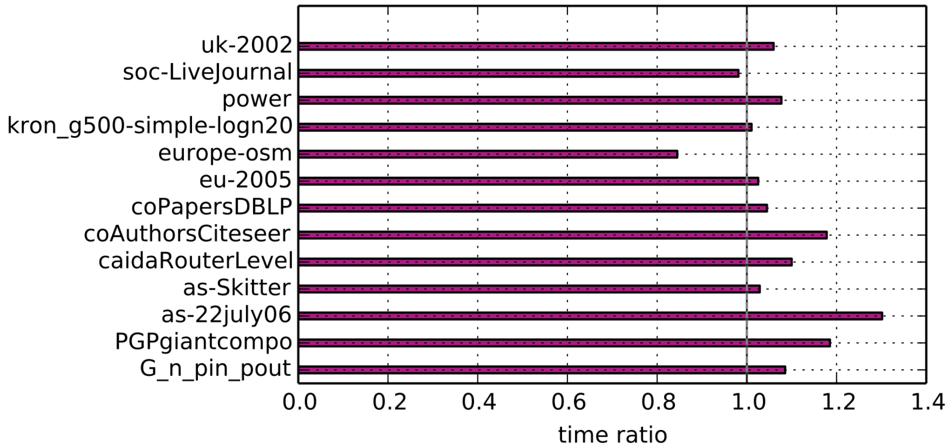


Figure: relative difference in running time of **PLMR** compared to **PLM**

Results | EPP

- improved solution quality compared to **PLP**
- small ensembles work best (here: 4-piece ensemble)
- ca. factor 10 slower than **PLP** alone
- modularity improvement ca. 0.05
- slightly faster (and with smaller memory footprint) than **PLM** and **PLMR**

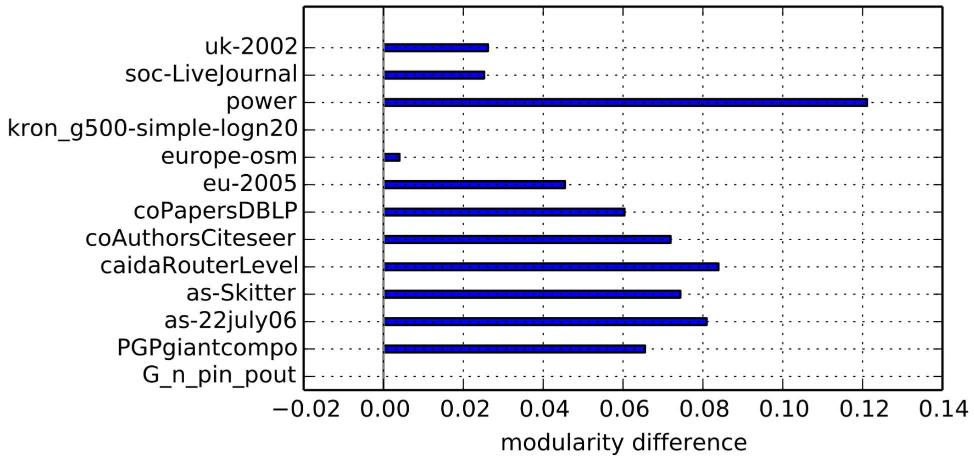


Figure: modularity improvement of **EPP** compared to single **PLP**

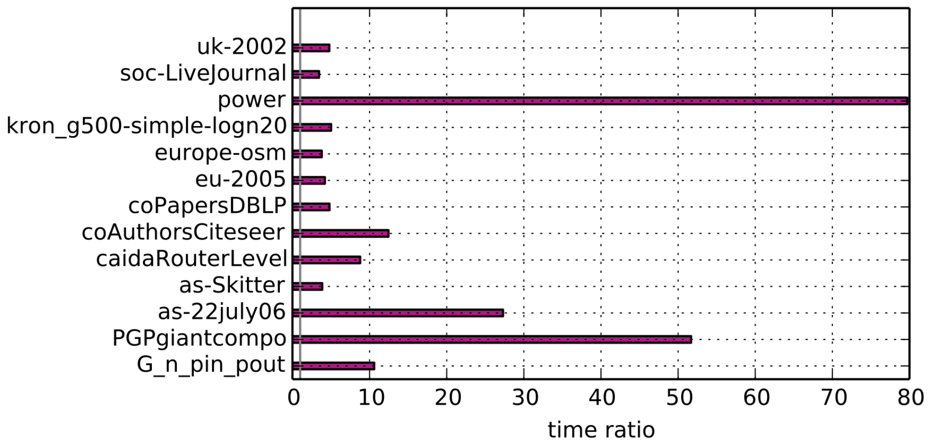
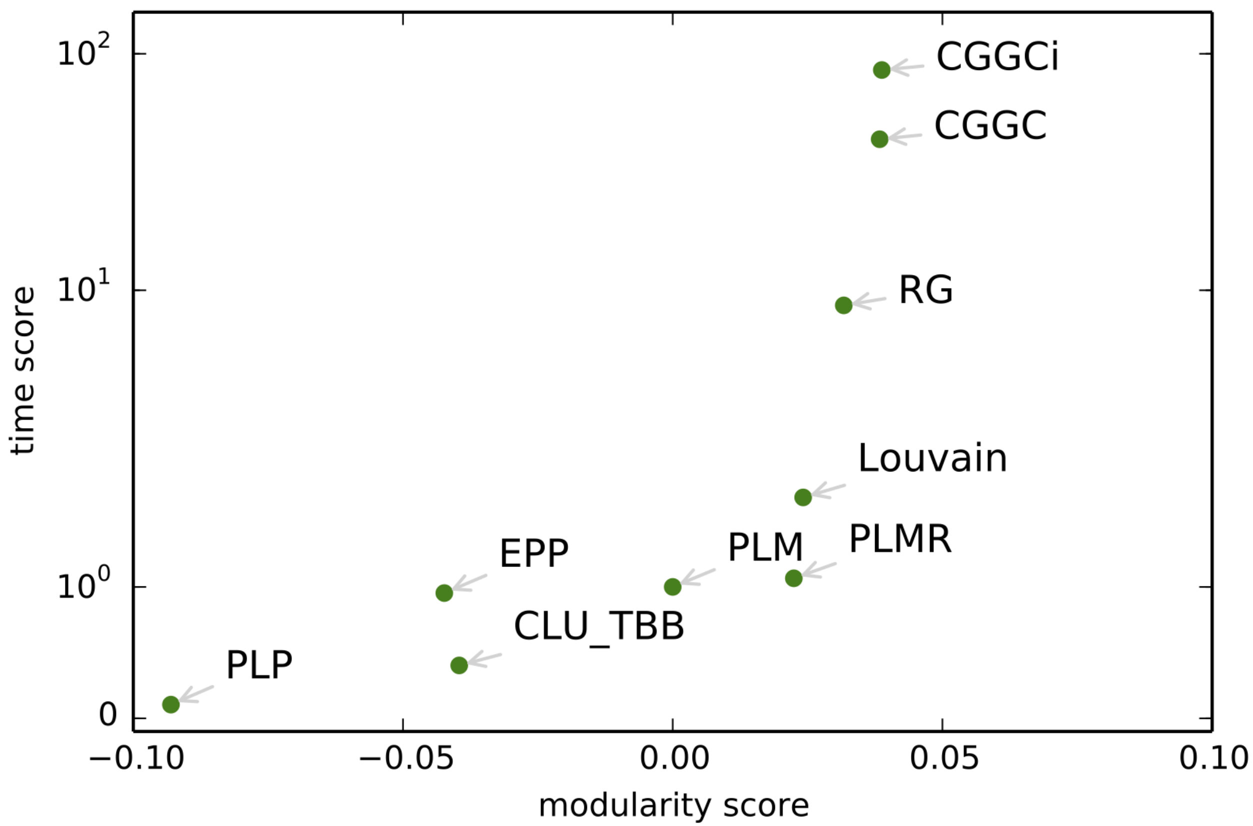


Figure: relative running time of **EPP** compared to single **PLP**

Pareto Evaluation

- modularity score: arithmetic mean over all networks of modularity differences
- time score: geometric mean over all networks of relative time differences
- baseline: **PLM** at (0, 1)



1. Introduction
2. Algorithms
3. Experiments
4. **NetworKit**
5. Conclusions

Software | NetworKit

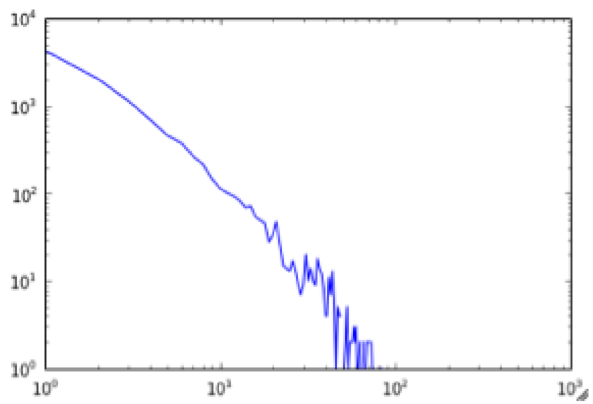
- a toolkit of **high-performance network analysis algorithms**
  - high-performance kernel in **C++11** and **OpenMP**
  - **Python** shell for interactive data analysis (via **Cython**)
- **free software** (*MIT License*)
  - 1.0 (spring 2013): community detection algorithms, data structures
  - 2.0 (November 2013): interactive Python shell
  - 2.1 (TBA): adds various network analysis kernels

NetworKit: An Example

Read Network from File

```
In [4]: G = readGraph("pgp.graph")
```

```
In [12]: xscale("log")
         yscale("log")
         plot(properties.degreeDistribution(G))
Out[12]: [<matplotlib.lines.Line2D at 0x107c5e090>]
```



Network Properties Overview

```
In [5]: properties.showProperties(G)
```

```
Network Properties
==================
Basic Properties
--------------  ------------
nodes (n)       10680
edges (m)       24316
min. degree         1
max. degree       205
avg. degree         4.55356
isolated nodes      0
self-loops          0
density             0.000426
--------------  ------------
Path Structure
-----------------------  -----
connected components         1
size of largest component  10680
diameter
avg. eccentricity
-----------------------  -----
Miscellaneous
--------------------  ------------
degree assortativity      0.238211
cliques               13814
--------------------  ------------
Community Structure
---------------------------------  -----------  --------
avg. local clustering coefficient               0.265945
PLP community detection
                                   communities  985
                                   modularity   0.795751
PLM community detection
                                   communities  108
                                   modularity   0.879849
---------------------------------  -----------  --------
```

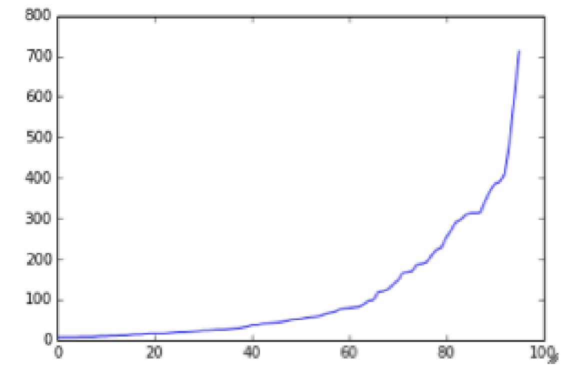
Software | NetworKit: An Example

Community Detection

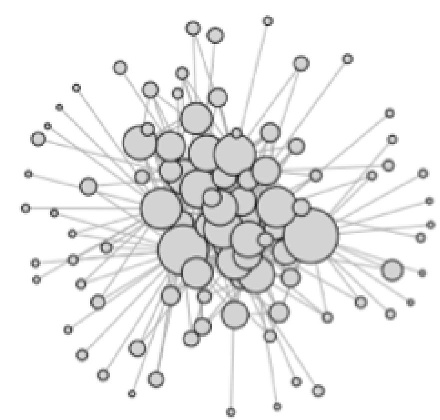
```
In [13]: communities = community.detectCommunities(G, algo=community.PLM())
         PLM(balanced) detected communities in 0.1455368995666504 [s]
         solution properties:
         -------------------  ----------
         # communities          98
         min community size      6
         max community size    683
         avg. community size   108.98
         imbalance               6.26606
         modularity              0.882828
         -------------------  ----------
```

```
In [7]: communitySizeDistribution = communities.clusterSizes()
        communitySizeDistribution.sort()
        plot(communitySizeDistribution)

Out[7]: [<matplotlib.lines.Line2D at 0x10c2bf1d0>]
```




```
In [8]: viztasks.drawCommunityGraph(G, communities)
```



- growing collection of network analysis kernels, graph generators, basic graph algorithms etc.
- integration with Python tools for data analysis and visualization
- users and contributors welcome

[http://parco.iti.kit.edu/software/networkit.shtml]

1. Introduction
2. Algorithms
3. Experiments
4. **NetworKit**
5. Conclusions

Conclusion | Summary & Future

Summary

- developed, implemented and evaluated scalabe heuristics for community detection
  - **PLP** extremely fast, but quality may not be high enough
  - **PLM** yields high quality
  - **PLMR** increases quality at the expense of more iterations
  - **EPP** combines their strengths

Ongoing and Future Work

- improve global community detection methods
  - e.g. parallel coarsening
- algorithms for related scenarios
  - selective and dynamic community detection [presented at ECDA2013 Luxembourg]

Thank you for your attention

Further Reading

C.L. Staudt, H. Meyerhenke:
Engineering High-Performance Community Detection Heuristics for Massive Graphs.
In Proc. 42nd International Conference on Parallel Processing (ICPP 2013).

Acknowledgements

This work was partially funded through the project *Parallel Analysis of Dynamic Networks - Algorithm Engineering of Efficient Combinatorial and Numerical Methods* by the *Ministry of Science, Research and Arts Baden-Württemberg*