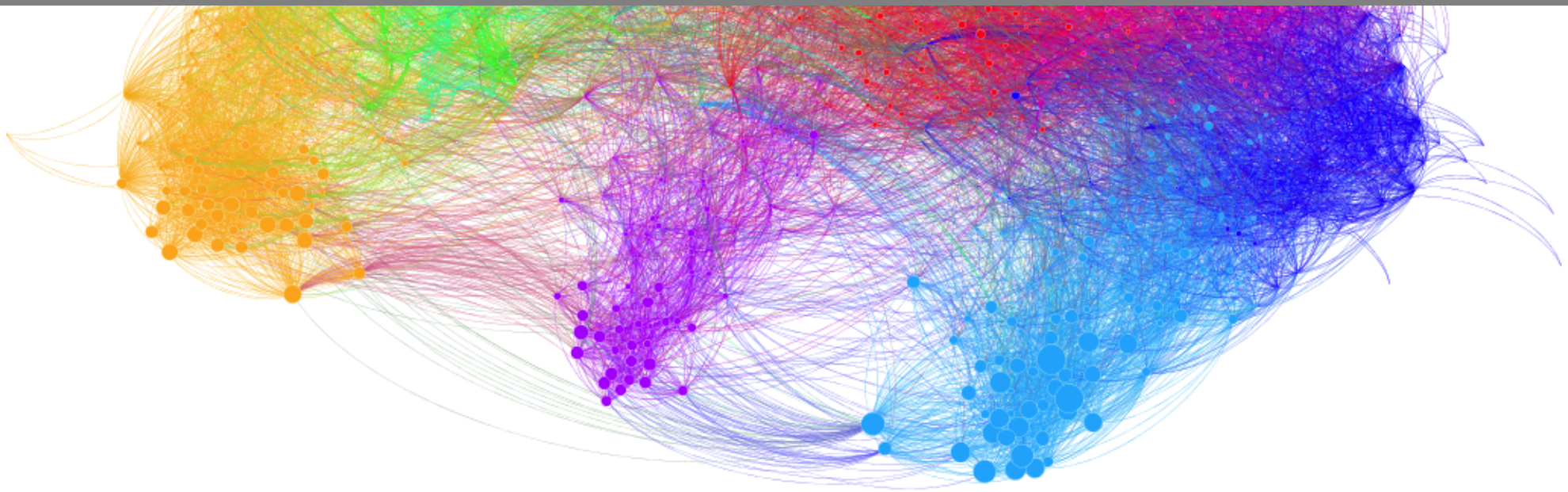
Karlsruhe Institute of Technology

Engineering High-Performance Community Detection Heuristics for Massive Graphs

International Conference on Parallel Processing 2013, Lyon · October 2, 2013

Christian L. Staudt and Henning Meyerhenke

INSTITUTE OF THEORETICAL INFORMATICS · PARALLEL COMPUTING GROUP

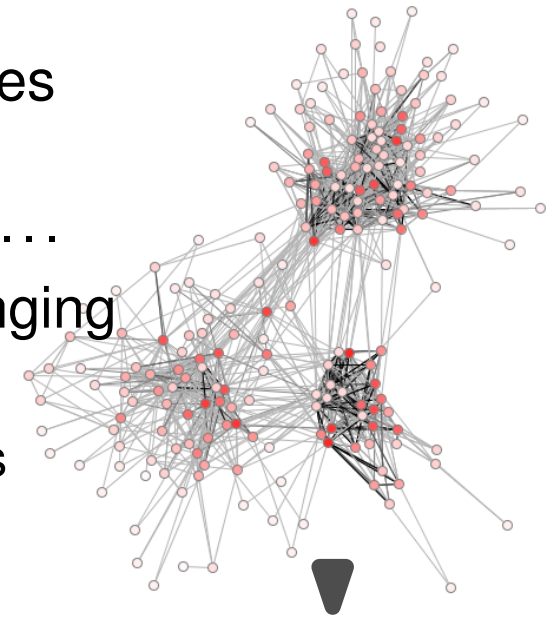


1. Introduction
2. Algorithms
3. Experiments
4. Conclusions

Introduction | Motivation

Big Network Data

- proliferation of large networks and high data rates
  - e.g. WWW ($>$ 30 billion pages),
    online social networks ($>$ 600 million active users),...
- **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]

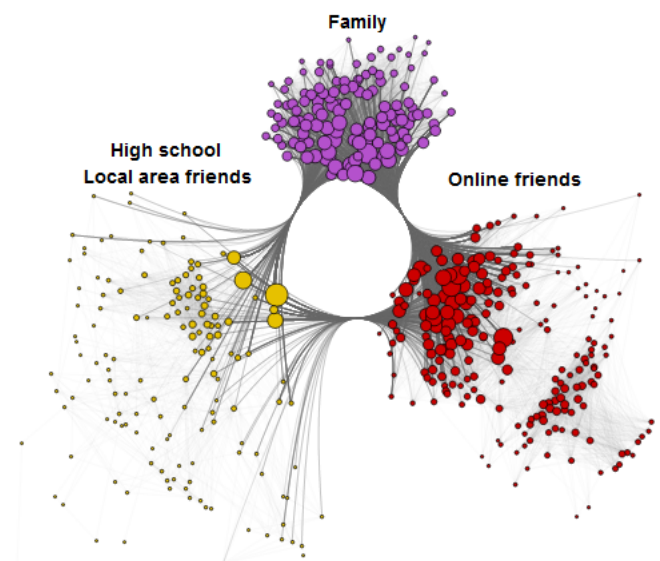


image: wolfram.com

Related Work | State of the Art

Challenge

10th DIMACS Implementation Challenge
- Graph Partitioning and Clustering

DIMACS
Center for Discrete Mathematics & Theoretical Computer Science
Founded as a National Science Foundation Science and Technology Center
— 10th DIMACS Implementation Challenge - Graph Partitioning and Graph Clustering —

- criteria: time and quality (**modularity**)
- high-quality solutions
  - **RG**, a randomized greedy agglomerative algorithm
  - **RG+**, 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

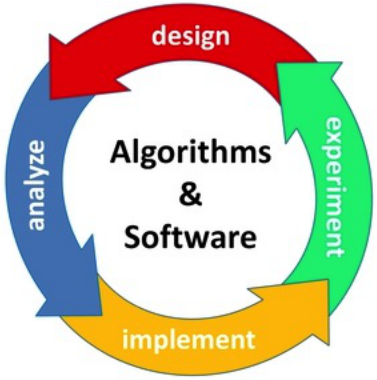
- original label propagation algorithm [Raghavan et al. 07]
- distributed parallel label propagation on Hadoop [Ovelgönne 12]

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

Capabilities

- **PLM** first parallel variant of **Louvain method**
- **ensemble technique EPP** yields best speed-quality tradeoff to date
- **NetworKit**: a framework for high-performance network analysis

1. Introduction
2. Algorithms
3. Experiments
4. Conclusions

Ensemble Preprocessing EPP

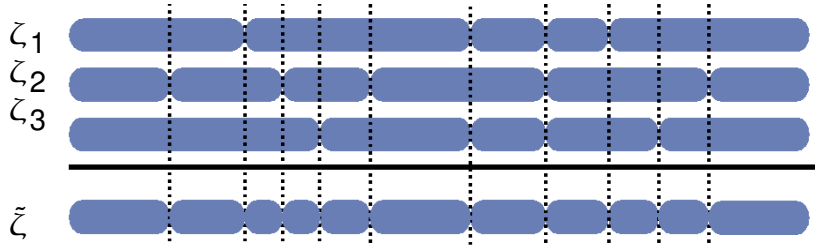
- **ensemble learning**: combine multiple weak classifiers to form a strong one
- generic scheme with exchangeable base and final algorithm [e.g. Ovelgönne & Geyer-Schulz 13]

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

EPP Implementation

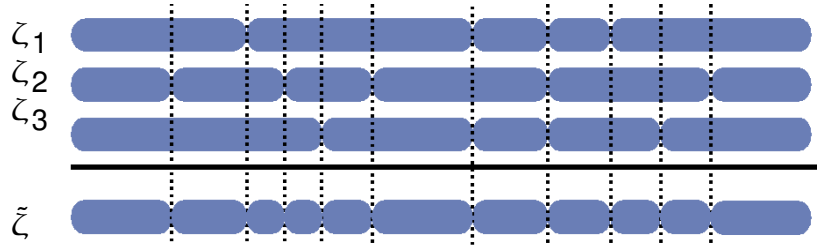
- nested parallelism in the ensemble
- efficiently calculate consensus communities through *k*-**way hashing** of community IDs
- base algorithm: focus on speed
- final algorithm: focus on quality optimization



consensus communities image: Ovelgönne & Geyer-Schulz 13

EPP Implementation

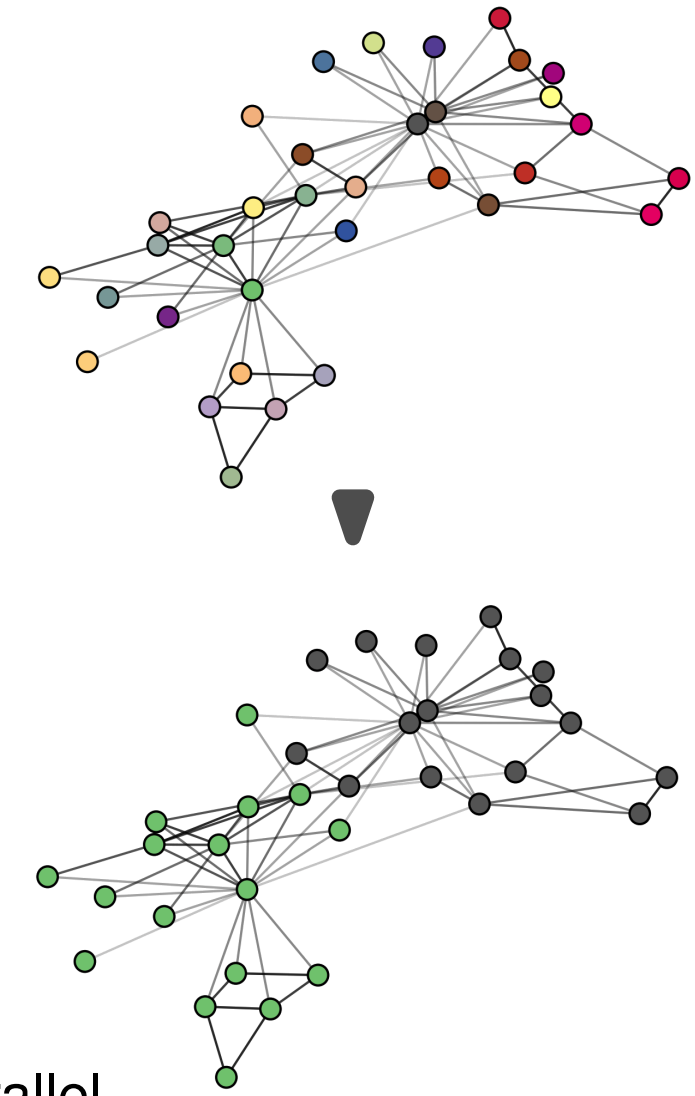
- nested parallelism in the ensemble
- efficiently calculate consensus communities through *k*-**way hashing** of community IDs
- base algorithm: focus on speed → **PLP**
- final algorithm: focus on quality optimization → **PLM**



consensus communities image: Ovelgönne & Geyer-Schulz 13

Algorithms | Parallel Label Propagation PLP

```
initialize nodes with unique labels
while labels not stable do
    parallel for v ∈ 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 is *desired*
    $\rightarrow$ modularity implicitly maximized
- $O(m)$ time per iteration, few iterations
- purely local updates $\rightarrow$ embarrassingly parallel

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

PLP Implementation

- 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)

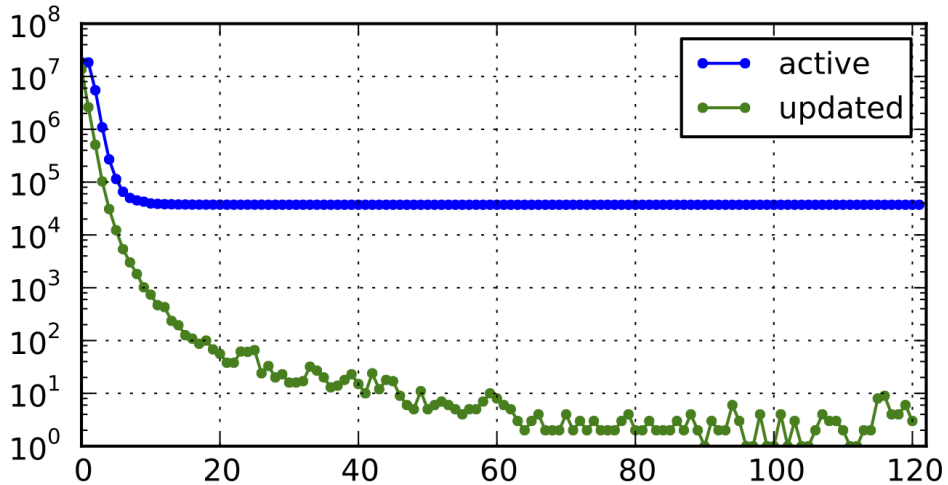


Figure: Number of active and updated nodes per iteration of **PLP** on a large web graph

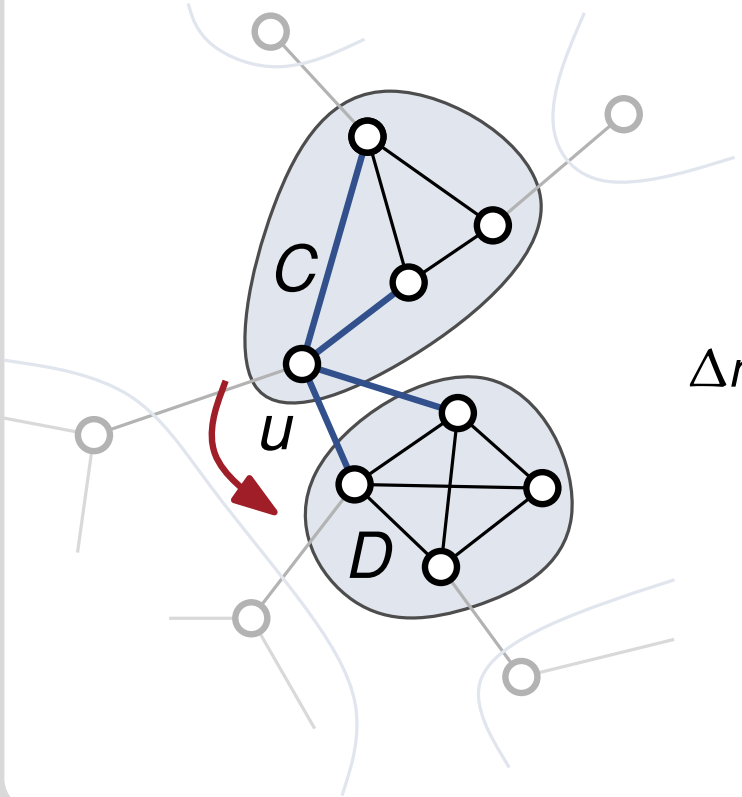
Parallel Louvain Method **PLM**

- 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
repeat
    while communities not stable do
        parallel for v ∈ V
            move v to neighbor community for max. Δmod
        endfor
    end
    coarsen graph
until no change in communities
return communities induced by coarsest graph
```

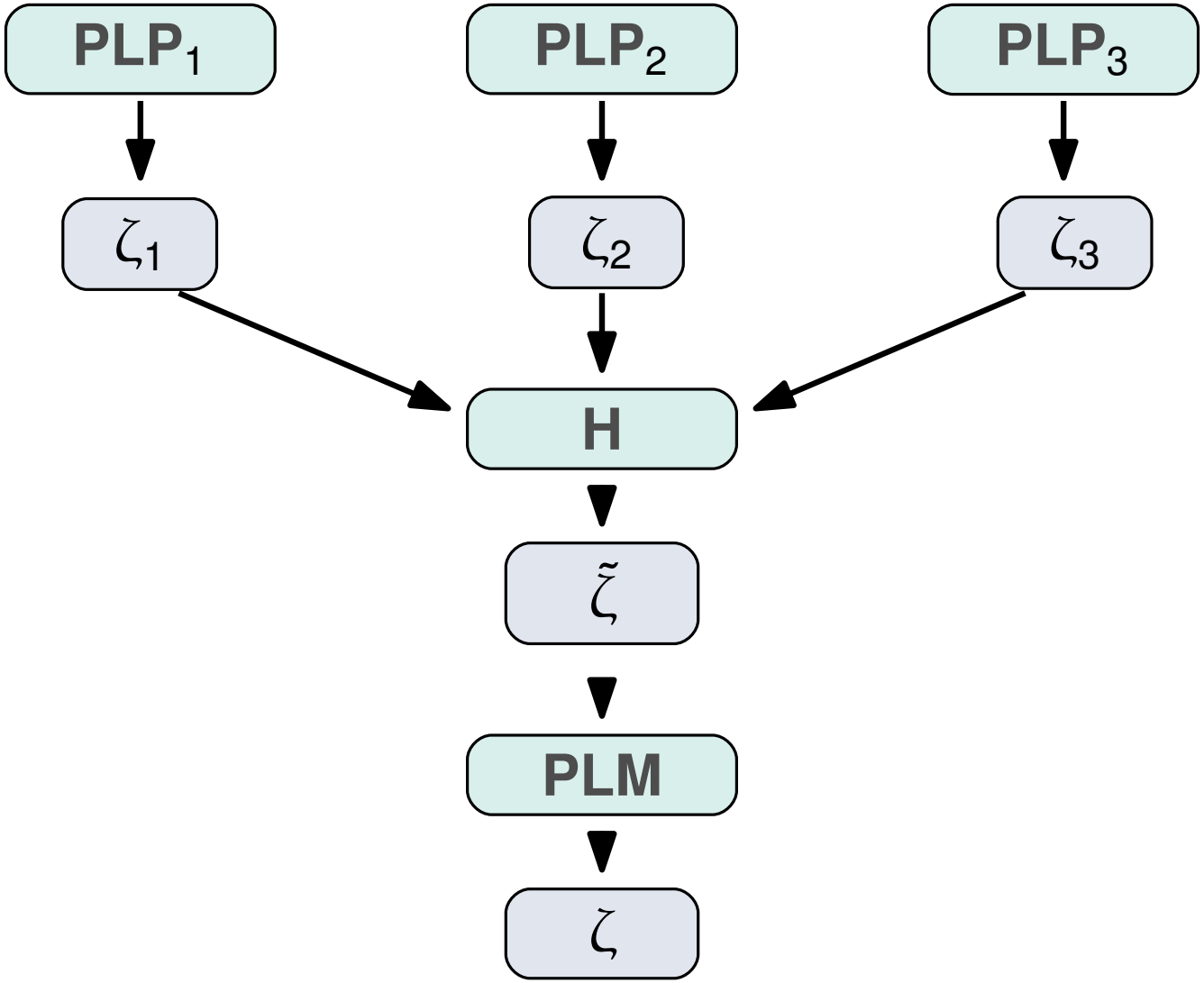
PLM Implementation

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



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

Algorithms | EPP



1. Introduction
2. Algorithms
3. Experiments
4. Conclusions

Experimental Setup | Networks

- variety of real-world and synthetic data sets
- **complex networks**: web graphs (uk2007), internet topology, online social network, scientific collaboration, . . .
- **Stochastic Kronecker Graphs (SKG)** for scaling experiments

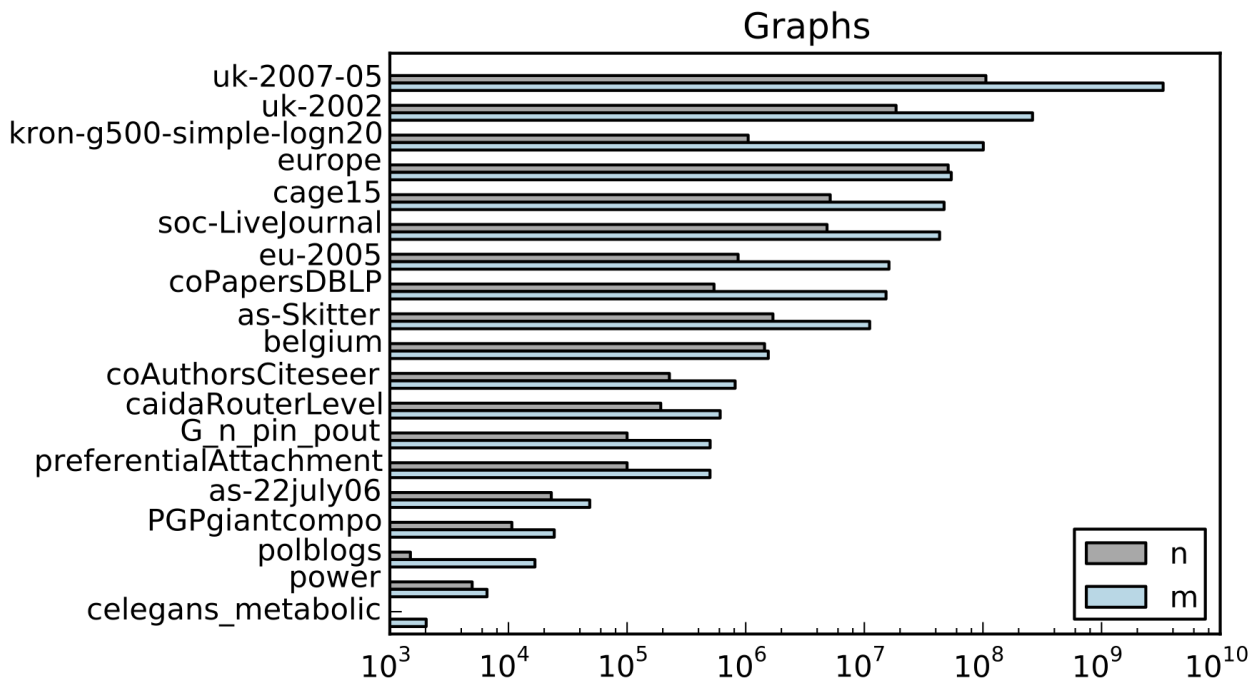


Figure: size comparison of test graphs

Experimental Setup | Settings

	Platform 1 (compute11.iti.kit.edu)	Platform 2 (ic2.scc.kit.edu)
compiler	gcc 4.7.1	gcc 4.7.2
OS	SUSE 12.2-64	SUSE ES 11 SP2
CPU	2x 8-Core Xeon E5-2670, 2.6 GHz	4x 8-Core Xeon E7-8837, 2.67 GHz
RAM	64 GB	512 GB

main machine for experiments

required for largest web graph

Results | PLP

- handles large graphs easily
  - 3.3 billion edge web graph in 120 s with 32 threads
- reasonable modularity values (but not optimal)

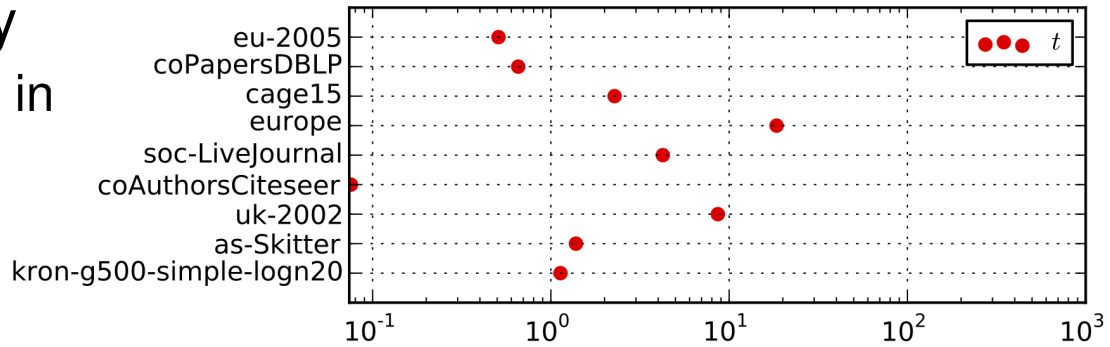


Figure: running time [s] for various networks

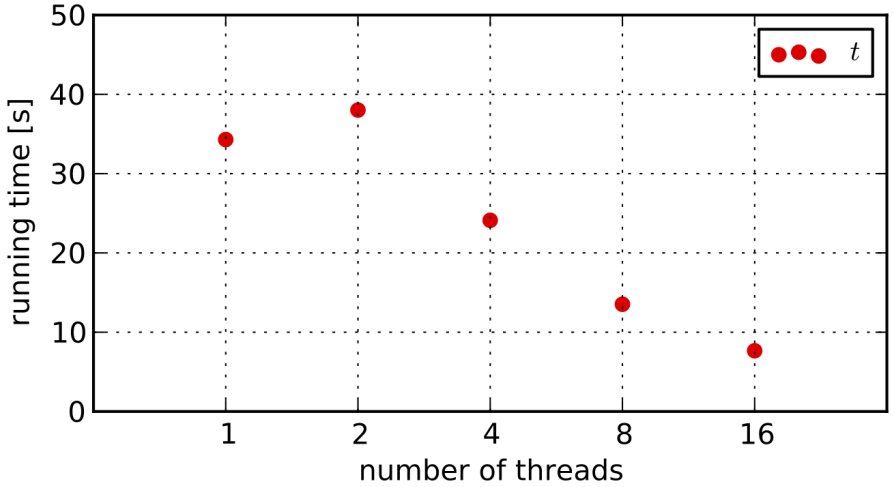


Figure: strong scaling of **PLP**

- linear strong scaling for 2-16 threads
  - 1-2 thread transition: presumably due to Intel dynamic scaling technology

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**
- but slower
  - rough estimate: factor 30
- scaling: worse than **PLP** because of locking

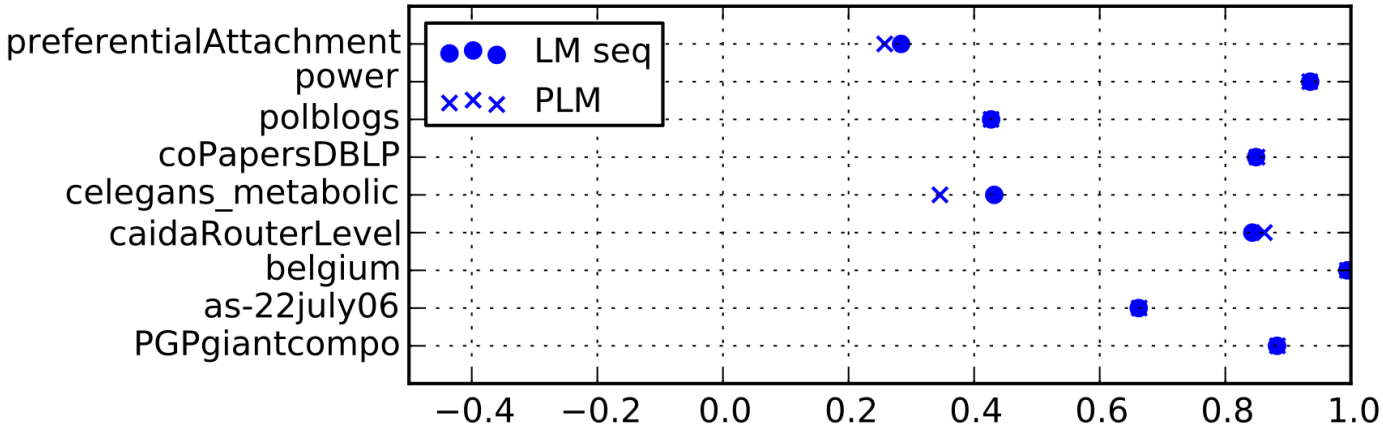


Figure: modularity values sequential vs parallel

Results | EPP

- improved solution quality compared to **PLP**
- small ensembles work best
- ca. factor 10 slower than **PLP** alone (4-piece ensemble)
- large graphs in short amount of time
  - 3 billion edge web graph in 11 minutes

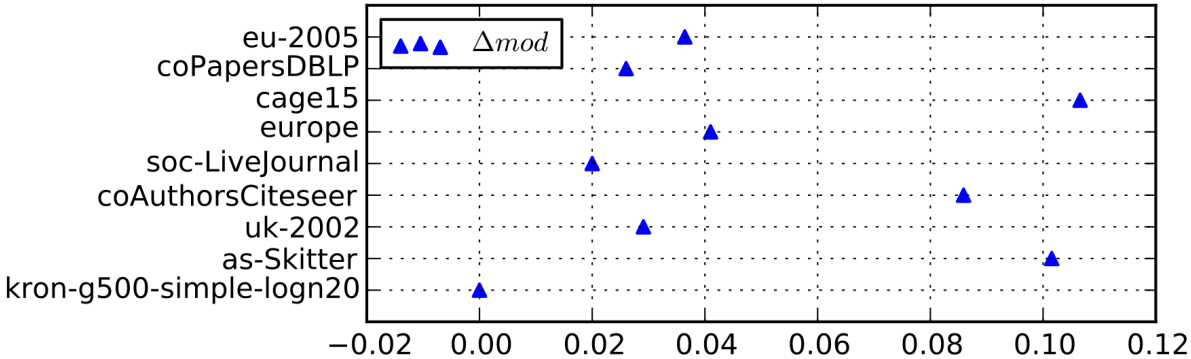


Figure: modularity improvement of **EPP** compared to single **PLP**

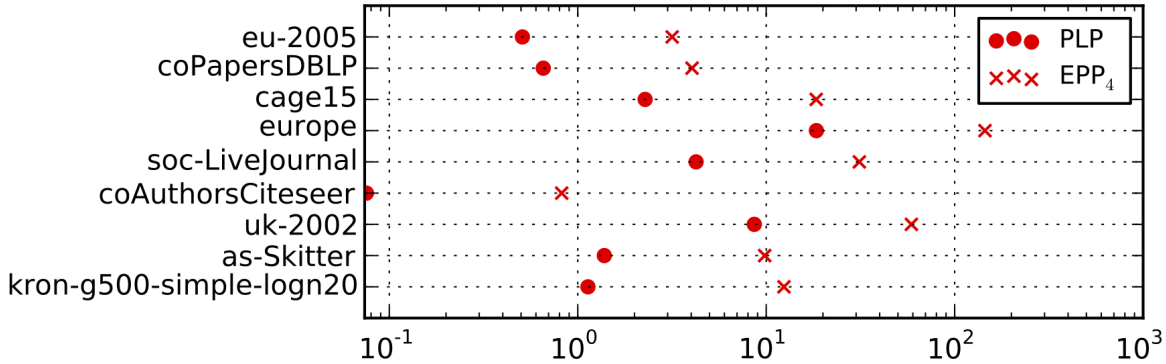


Figure: running time [s] of **EPP** compared to single **PLP**

Results | Comparison with DIMACS Competitors

- fastest algorithms: 1. **CLU_TBB** (parallel agglomerative), 2. **EPP**$_4$
  - 8 vs 5 mio edges/s (caveat: measurement on different machines) - but **EPP** qualitatively better
- **RG** (greedy agglomerative, sequential) and **RG+** (ensemble with **RG** as base and final) achieve better quality
  - but **EPP** 1-2 orders of magnitude faster
- $\rightarrow$ **EPP** not dominated
- **PLP** alone faster than any competitor

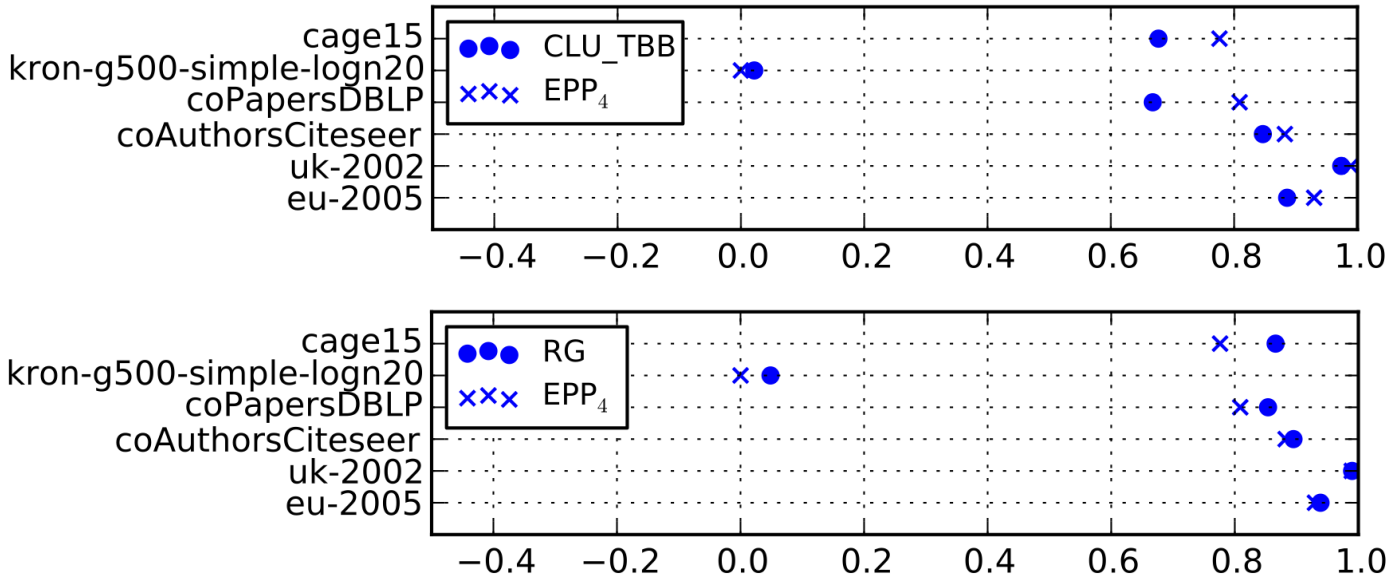


Figure: modularity values - **EPP** vs **CLU_TBB** and **RG**

1. Introduction
2. Algorithms
3. Experiments
4. 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, but is significantly slower
  - **EPP** combines their strengths and delivers best speed-quality tradeoff to date

Ongoing and Future Work

- improve global community detection methods
  - label propagation for different objectives
  - lock-free **PLM**
- algorithms for related scenarios
  - selective and dynamic community detection
    [presented at ECDA2013 Luxembourg]

Conclusion | NetworKit

- a toolkit of **high-performance network analysis algorithms**
  - **C++11** and **OpenMP**
- **free software** (*MIT License*)
  - 1.0 release in spring 2013: static global community detection
  - 2.0 release coming later in 2013

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

Conclusion | NetworKit

- a toolkit of **high-performance network analysis algorithms**
  - **C++11** and **OpenMP**
- **free software** (*MIT License*)
  - 1.0 release in spring 2013: static global community detection
  - 2.0 release coming later in 2013

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

Thank you for your attention

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*

Appendix