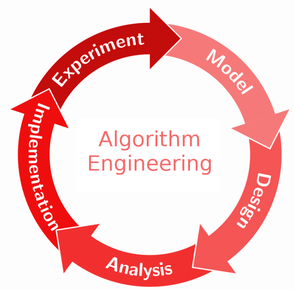
# Local Motif Clustering via (Hyper)Graph Partitioning

Adil Chhabra[1], Marcelo Fonseca Faraj[1], Christian Schulz[1]

[1]Heidelberg University, Germany

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386




## Introduction

Given a graph and a seed node, the local clustering problem consists of identifying a *well-characterized* cluster which contains the seed node.

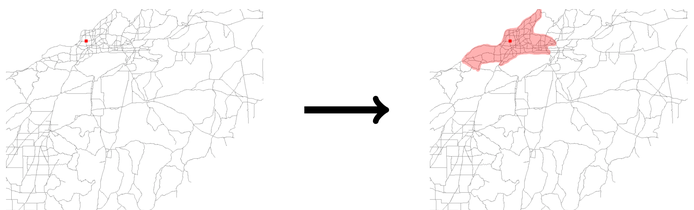


**Figure 1:** Local Graph Clustering.

The quality of a cluster C can be measured by its *conductance* [1] $\phi(C)$, which is NP-hard [2] to optimize.

$$\phi(C) = \frac{cut(C, \overline{C})}{Min\{Volume(C), Volume(\overline{C})\}} \quad (1)$$

In traditional clustering, a cluster is evaluated by its edge distribution. A promising novel approach named *local motif clustering* evaluates clusters based on the distribution of a certain motif, or subgraph $\mu$. In this approach, quality of a cluster is measured by its *motif conductance* $\phi_\mu(C)$.

$$\phi_\mu(C) = \frac{cut_\mu(C, \overline{C})}{Min\{Volume_\mu(C), Volume_\mu(\overline{C})\}} \quad (2)$$

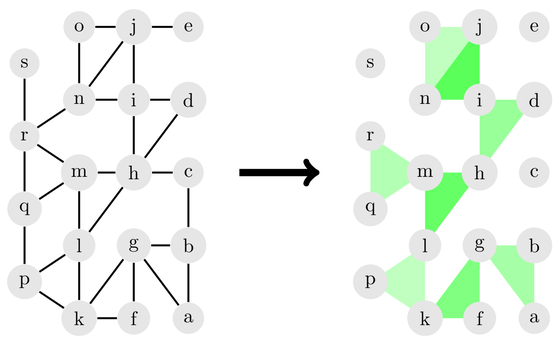


**Figure 2:** Occurrences of triangle motif.

In this work, we propose a combinatorial algorithm to solve the local motif clustering problem based on sophisticated (hyper)graph partitioning algorithms.

**Related Work.** Yin et al. [3] propose MAPPR, an algorithm based on PageRank to minimize $\phi_\mu$. It enumerates motifs in the whole graph and builds a model $W$ with edges between nodes contained in a motif such that edge weights equal the amount of motifs containing both their endpoints. It clusters on $W$ with a modified APPR.

## Local Motif Graph Clustering

**Overall Strategy.**

1: **for** $i = 1, \ldots, \alpha$ **do**
2: $S \leftarrow$ ball around $u$; $M \leftarrow$ Enumerate motifs in $S$
3: Build (hyper)graph model $H_\mu$ based on $S$ and $M$
4: **for** $j = 1, \ldots, \beta$ **do**
5: Partition model $H_\mu$ into $(C, \overline{C})$, where $u \in C$
6: **if** $C^* = \emptyset \vee \phi_\mu(C) < \phi_\mu(C^*)$ **then** $C^* \leftarrow C$
7: Convert $C^*$ into a local motif cluster in $G$

**Ball around Seed Node.** We select $S$ via a fixed-depth breadth-first search (BFS) rooted on $u$. In particular, we compute the first $\ell$ layers of the BFS tree rooted on $u$, then we include all its nodes in $S$. For the $\alpha$ repetitions of this phase, we use different amounts $\ell$ of layers for a better exploration.

**(Hyper)Graph Model.** Our algorithm has two configurations: one using a hypergraph model for partitioning, another using a graph model for partitioning.

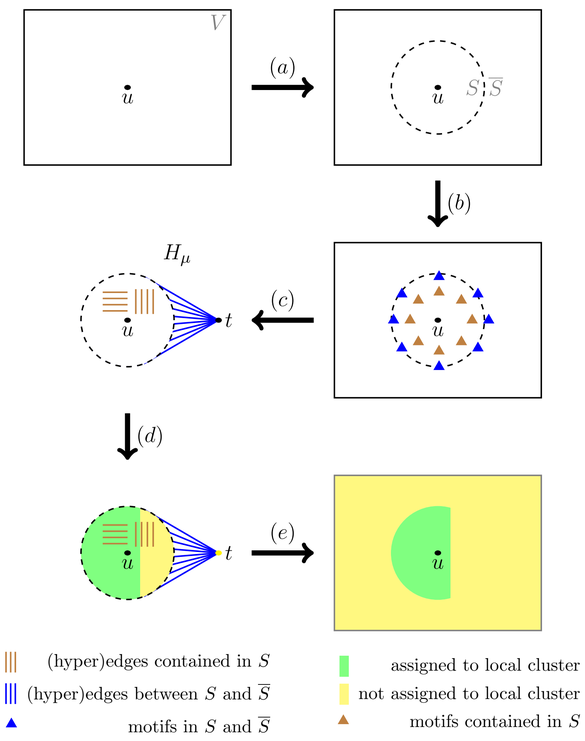



**Figure 3:** Overall strategy: (a) Compute ball $S$ of nodes around seed node $u$. (b) Enumerate motifs with at least one node in $S$. (c) Build (hyper)graph model $H_\mu$. (d) Partition $H_\mu$. (e) Convert partition to cluster around $u$.

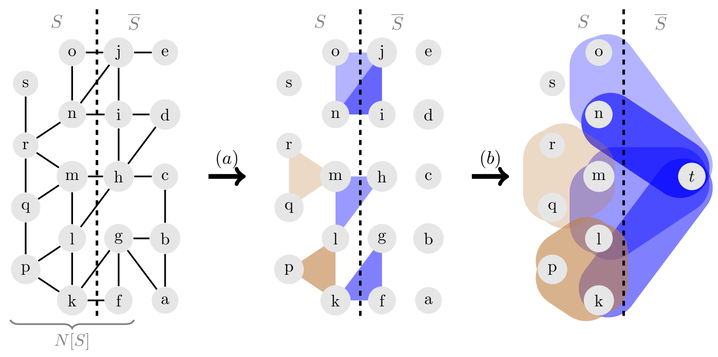



**Figure 4:** Example construction of hypergraph model $H_\mu$. (a) Enumerate motifs touching $S$. (b) Build model $H_\mu$.

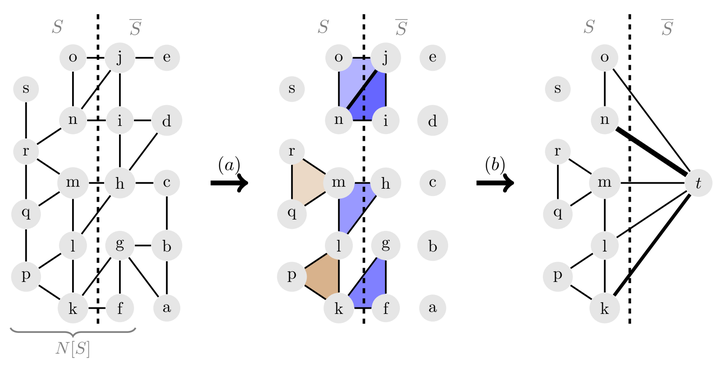



**Figure 5:** Example construction of graph model $H_\mu$. (a) Enumerate motifs touching $S$. (b) Build model $H_\mu$.

***Theorem 1.*** *A cluster $C$ in $H_\mu$, $t \notin C$, implies a cluster in $G$ whose $\phi_\mu(C)$ is the ratio of its cut-net (edge-cut) to its volume in $H_\mu$, assumed a motif with 3 nodes and $Volume_\mu(S) \leq Volume_\mu(\overline{S})$.*

**Partitioning.** For partitioning our hypergraph and graph models, we respectively use KaHyPar [4] and KaHIP [5]. For both algorithm versions, we obtain partitions where $u$ and $t$ are in different blocks to ensure consistency. We force it by assigning $u$ to the block that does not contain $t$ right after the partition is computed. We repeat the partitioning $\beta$ times with different imbalances for each ball $S$ for a better exploration.

## Experimental Evaluation

**Methodology and Instances.** We implemented our algorithm on the KaHIP [5] framework and use the KaHyPar [4] library. We used a Linux machine with a 64-core AMD EPYC 7702P and 1 TB of RAM. The graphs in our test set are exactly the ones used in the MAPPR paper [3].

Our experiments are based on the triangle motif. For each graph, we use 50 random seed nodes.

| Graph | GL;3;80 | | | MAPPR | | |
|---|---|---|---|---|---|---|
| | $\phi_\mu$ | \|C\| | t(s) | $\phi_\mu$ | \|C\| | t(s) |
| com-amazon | **0.037** | 64 | 0.22 | 0.153 | 58 | 2.68 |
| com-dblp | **0.115** | 56 | 0.38 | 0.289 | 35 | 3.04 |
| com-youtube | **0.172** | 1443 | 7.93 | 0.910 | 2 | 10.44 |
| com-livejournal | **0.244** | 387 | 8.17 | 0.507 | 61 | 173.80 |
| com-orkut | **0.150** | 13168 | 496.94 | 0.407 | 511 | 923.26 |
| com-friendster | **0.368** | 10610 | 1339.99 | 0.741 | 121 | 16565.99 |
| Overall | **0.181** | 823 | 12.67 | 0.500 | 50 | 79.34 |

**Table 1:** Average comparison against state-of-the-art.

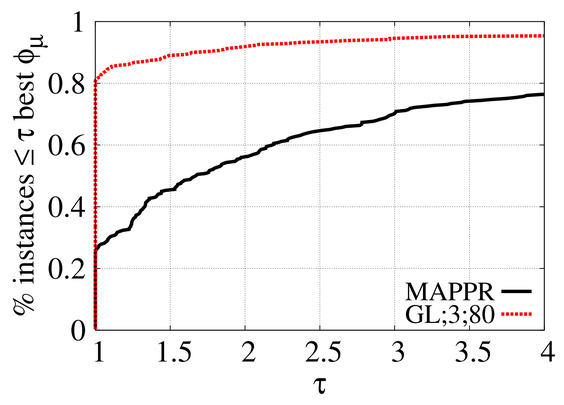



**Figure 6:** Motif conductance perf. profile.

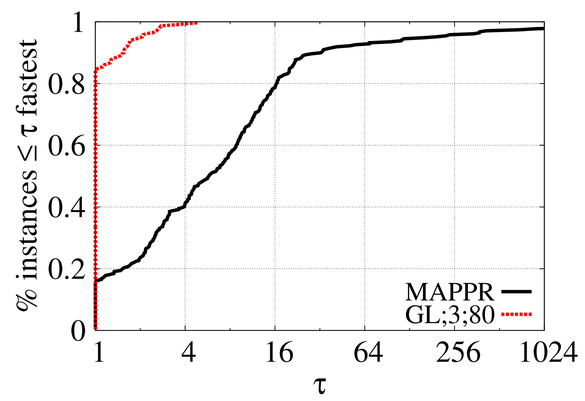



**Figure 7:** Running time perf. profile.

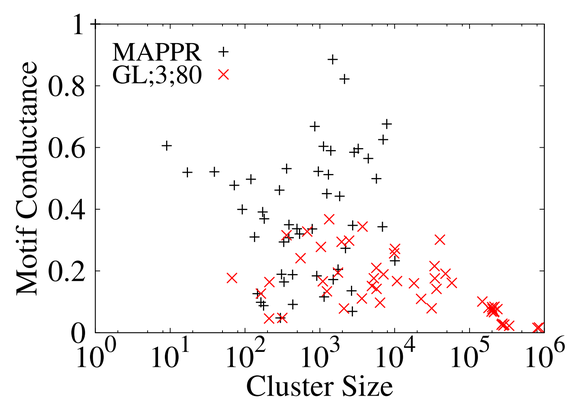



**Figure 8:** Found clusters for com-orkut.

**Results.** Our algorithm gets the best or equal $\phi_\mu$ for 80% of the instances. This is due to the fact that by performing multiple cluster constructions and refinements our algorithm can explore the solution space better than MAPPR which simply uses the APPR algorithm. Our algorithm is faster than MAPPR for 84% of the instances, besides being on average a factor 6.3 faster.

## References

[1] R. Kannan et al., On clusterings: Good, bad and spectral, *JACM, 2004*

[2] S. Dutt, Between Min Cut and Graph Bisection, *IEEE/ACM Intl. Conf. on Computer-Aided Design, 1993*

[3] H. Yin et al., Local higher-order graph clustering, *ACM SIGKDD, 2017*

[4] S. Schlag et al., KaHyPar, https://github.com/kahypar/kahypar

[5] P. Sanders et al., KaHIP, https://github.com/KaHIP/KaHIP

**15th International Symposium on Combinatorial Search (SoCS-22), 21 - 23 July 2022, Vienna, Austria**