Beginner's Practical: Hypergraph Conductance-based Clustering

MARIIA MESHCHANINOVA, Heidelberg University, Germany

Hypergraph clustering is a problem of partitioning the nodes of a hypergraph into "true clusters" which are densely connected internally and sparsely connected externally. Conductance is a metric measuring the quality of a cut which is defined as the ratio of the weight of the cut to the minimal volume of one side of the cut.

In this project, we implement a hypergraph clustering algorithm that heuristically tries to minimize the maximum value of the conductance over all cuts between clusters to find a high-quality hypergraph clustering. The algorithm is based on the existing shared-memory parallel algorithm for hypergraph partitioning `Mt-KaHyPar`.

1 INTRODUCTION

Hypergraph clustering is a problem of partitioning the nodes of a hypergraph into "true clusters" which are densely connected internally and sparsely connected externally. It is useful in many applications, such as semi-supervised learning [12], gene expression analysis [11] and image segmentation [6]. Conductance metric, which is defined for a cut as the ratio of the cut weight to the minimal volume of one side of the cut, is widely used in graph clustering [3, 8] but to the best of our knowledge, there are no partitioners directly optimizing conductance-based metric of a hypergraph. However, conductnace has been applied as a metric for evaluating the quality of a hypergraph clustering [1]. Therefore in this project, we implement a heuristic hypergraph clustering algorithm with the objective of minimizing the maximum value of the conductance over all cuts between the clusters while partitioning the hypargraph in at most k clusters.
Our algorithm is based on the existing shared-memory parallel hypergraph k-way partitioner `Mt-KaHyPar` and uses the same multilevel approach as the original algorithm. First, the hypergraph is coarsened in stages by detecting and merging local node clusters, then the coarsest hypergraph is partitioned into k clusters, and finally the initial partitioning is refined by uncoarsening the hypergraph nodes and applying a local search algorithm to improve the given objective. We implement two objective functions for the local search algorithm - *conductance_local* and *conductance_global* - and compare their performance on the circuit-based benchmark set *ISPD98* [2] against a state of the art hypergraph clustering algorithm `HyperModularity`. We also conduct a scaling experiment to evaluate the performance of our algorithm one multiple cores.
In following, we define the used concepts in Section 2, then we shortly describe related work on hypergraph and conductance-based clustering in Section 3. In Section 4, we describe the implementation of the new objectives for optimizing the conductance and their integration in `Mt-KaHyPar`. Afterwards, we present the experimental results of our implementation in Section 5. Finally, we conclude the report and discuss future work in Section 6.

2 PRELIMINARIES

In this section, we give an overview of the used concepts and notations and at the end define conductance of a hypergraph.

Hypergraph is a generalization of a graph, where a hyperedge connects one, two or more nodes. Formally, an edge-weighted hypergraph $H = (V, E, \omega)$ consists of a non-empty set of nodes V, a set of hyperedges (also called *nets* or simply *edges*) $E \subseteq 2^V$ and a weight function for hyperedges $\omega : E \longrightarrow \mathbb{N}$. In case of unweighted hyperedges, we set $\omega(e) = 1$ for all $e \in E$.

Author's address: Mariia Meshchaninova, mariia.meshchaninova@stud.uni-heidelberg.de,InformatikB.Sc.100%,4740180, Heidelberg University, Im Neuenheimer Feld 205, Heidelberg, Baden-Württemberg, Germany, 69120.

k**-way clustering** of a hypergraph $H = (V, E, \omega)$ is a partitioning of the set of nodes V into k disjoint subsets $V_1, \dots, V_k$ called *clusters*. Empty clusters are allowed, so k is only an upper bound on the final number of clusters. Our algorithm is allowed to eventially move all the nodes of a cluster to other clusters if it would be beneficial for the objective function.

Cut ∂S of a hypergraph $H = (V, E, \omega)$ is partitioning of the set of nodes V into two disjoint non-empty subsets S and $V \backslash S$. A hyperedge $e \in E$ is called a **cutting edge** of the cut ∂S if it has at least one pin in S and at least one pin in $V \backslash S$. The weight of the cut is defined as the sum of the weights of the cutting edges of the cut ∂S:

$$\omega(\partial S) = \sum_{e \in \partial S} \omega(e)$$

Therefore in case of unweighted hyperedges, the weight of the cut is equal to the number of cutting edges of the cut.

Pin of a hyperedge $e \in E$ is a node $v \in V$ such that $v \in e$. A hyperedge with only one pin is called a **sinle-pin net**. In the implementation, we use number of pins $\text{numPins}_i(e)$ of a hyperedge $e \in E$ in a cluster V_i to decide whether to move a node v from one cluster V_i to another cluster V_j.

Weighted degree of a node $v \in V$ is defined the same way in hypergraphs as in graphs. It is the sum of the weights of all hyperedges $e \in E$ that contain the node v:

$$\deg_\omega(v) = \sum_{e \in E : v \in E} \omega(e)$$

As during the coarsening phase of the algorithm we merge nodes, we also use **original weighted degree** of a node $v' \in V'$ of the coarsened hypergraph $\text{origDeg}_\omega(v')$ which is defined as the sum of the weighted degrees of all nodes of the initial hypergraph $v \in V$ that were merged into v' during the coarsening phase of the algorithm.

Volume of a hypergraph $H = (V, E, c, \omega)$ is defined as the sum of the weighted degrees of all nodes $v \in V$. Analogously, we define the **volume of a cluster** V_i as the sum of the weighted degrees of all nodes $v \in V_i$:

$$\text{vol}(H) = \sum_{v \in V} \deg_\omega(v) \qquad \text{vol}(V_i) = \sum_{v \in V_i} \deg_\omega(v)$$

And for the same reason as for the original weighted degree, we also define the **original volume** of a hypargraph and a cluster:

$$\text{origVol}(H) = \sum_{v \in V} \text{origDeg}_\omega(v) \qquad \text{origVol}(V_i) = \sum_{v \in V_i} \text{origDeg}_\omega(v)$$

This way, the original volume of a cluster or of the whole hypergraph in the coarsened hypergraph is equal to the corresponding volume in the initial - *original* - hypergraph.

Conductance of a cut ∂S of a hypergraph $H = (V, E, \omega)$ is defined as the ratio of the weight of the cut to the minimum volume of one side of the cut. The maximal conductance of a cut In a hypergraph is referred to as the **conductance of the hypergraph**:

$$\varphi(S) = \frac{\omega(\partial S)}{\min\{vol(S), vol(V \backslash S)\}} \qquad \varphi(V) = \max_{\emptyset \subsetneq S \subsetneq V} \varphi(S)$$

To use conductance as a quality metric for a hypergraph clustering with possibly more than two clusters, we define the **conductance of a hypergraph clustering** $V_1, \dots, V_k$ as the maximum conductance of all cuts between the clusters. Conveniently, this definition can be simplified to the maximal conductance of a cut ∂V_i over all clusters V_i:

$$\varphi(V_1, \dots, V_k) := \max_{I \subsetneq \{1, \dots, k\} : \cup_{i \in I} V_i \neq \emptyset, V} \varphi(\cup_{i \in I} V_i) = \max_{i = 1 \dots k : V_i \neq \emptyset, V} \varphi(V_i)$$

A proof of this nice statement can be found in Appendix A.

3 RELATED WORK

For graph clustering, there are several state of the art algorithms that aim to minimize the resulting conductance. Some of them are heuristic like the diffusion based `NIBBLE` [10] and `hk-relax` [7]. Other, such as degree ratio-based `PC_de` [8] have a theoretical guaranatee of the resulting conductance.

As for the hypergraph-clustering, to the best of our knowledge, there are no existing algorithms that directly optimize a conductance-based objective. Instead, state of the art hypergraph clustering algorithms, e. g., `HyperModularity` which we use for comparison with our algorithm, deliver high quality hypergraph clustering by optimizing the modularity of the hypergraph [4, 9].

The algorithm `Mt-KaHyPar` that we use as a base for our implementation also optimizes the modularity of the hypergraph in its coarsening stage. But as whole, it could be seen as a framework for hypergraph partitioning with a user-implemented objective, which is optimized during the initial partitioning of the coarsest hypergraph and later in the uncoarsening stage consisting of label propagation and FR-refinement x[5].

4 IMPLEMENTATION

To implement a new objective function for `Mt-KaHyPar`, we needed to implement efficient calculation of the conductance of the clustering, formulate an initial partitioning algorithm to apply to the coarsest hypergraph and a define a gain function for the local search algorithm. In this section, we describe our approaches to these tasks.

4.1 Efficient Conductance Calculation

Conductance calculation is needed to evaluate the quality of the clustering and to stop the local search algorithm when no more moves are worsening the conductance of the clustering.

Firstly, to be able to efficiently calculate the conductance of a clustering, we keep all original weighted degrees, original volumes and cut weights up to date. This is done by adjusting their values during the changes of the clustering or (un)coarsening of the hypergraph. Here are both these changes:

- **(Un)contraction** of a group of nodes $v_1 \dots v_c$ into (from) one node v':
  - $\text{origDeg}_\omega(v') = \text{origDeg}_\omega(v_1) + \dots + \text{origDeg}_\omega(v_c)$. Hence, atomical substraction (addition) is enough to update the original weighted degrees; It is important to note that this operation doesn't increase the run-time complexity of the algorithm, as to contract the nodes algorithm needs to go through them.
  - original volumes and cut weights should't be updated, as uncontracted nodes $v_1 \dots v_c$ are placed in the cluster of their representative node v'.
- **Node move** of v from one cluster V_i to another cluster V_j:
  - for original volumes we simpty need one atomical addition and one atomical substraction:

$$\text{origVol}(V_i) -= \text{origDeg}_\omega(v), \qquad \text{origVol}(V_i) += \text{origDeg}_\omega(v)$$

  - updating cut weight is a bit trickier, but conveniently, `Mt-KaHyPar` maintains the number of pins $\text{numPins}_t(e)$ of each hyperedge e in each cluster V_t, so the following adjustments for each incident to v hyperedge e are enough to update the cut weights:
    - $*$ if $1 = \text{numPins}_i(e) < size(e)$:

e is a cutting net for the cut $\partial(V_i)$, v is its last pin in V_i, therefore e won't be a cutting edge after removal of v: cut weight of $\partial(V_i)$ shuld be decreased by the weight of the hyperedge e,

  * else if $1 < \text{numPins}_i(e) < size(e)$:
    e was and still will be a cutting net for the cut $\partial(V_i)$, therefore no changes from this edge are needed for the cut weight of $\partial(V_i)$.
  * else if $1 < \text{numPins}_i(e) = size(e)$:
    e was not a cutting net for the cut $\partial(V_i)$, but it will be after the move of v, therefore the weight of the hyperedge e should be added to the cut weight of $\partial(V_i)$.

Again, this operation doesn't increase the run-time complexity of the algorithm, as to move a node the algorithm needs to go through all its incident hyperedges and update the number of pins in each cluster.

Maintaining this information makes calculating of the maximal conductance of a cut $\partial(V_i)$ possible. But to do it more efficient than going through all the partitions each time we need to calculate the conductance of the clustering, we implement an additional data structure: **ConductancePriorityQueue** - a priority queue with an entery for each cluster V_i. This entery is a fraction with $\omega(\partial(V_i))$ in the numerator and $\min(\text{origVol}(V_i), \text{origVol}(V) - \text{origVol}(V_i))$ in the denominator. These enteries are updated by changes of the cut weights and volumes of the clusters. To make sure, that our priority queue (max 2 heap) is always sorted, we use a lock by every change of its entery.

To store and compare the fractions, we also use another new class **NonnegativeFraction**, that supports comparison of two nonnegative fractions without a risk of an overflow. This is achieved by tricks with integral diviion and comparing recipocal rest fractions in the case of the same integral part of the fraction. Also, for the case of an empty cluster, we make sure, that a fraction with 0 in the denominator is always not than any other fraction without a zero in the denominator.

With all this information, we can efficiently calculate the conductance of the clustering by sinmpy looking at the top element of the ConductancePriorityQueue.

4.2 Initial Partitioning

Initial partitioning algorithm is applied after the coarsening phase of the algorithm (which works exactly the same way as in the original `Mt-KaHyPar` algorithm) to the coarsest hypergraph. Our first but less successful approach was to use a *singleton* partitioning, where each node is assigned to a separate cluster. The idea is, that during the uncoarsening phase, the algorithm will merge some of the clusters together, to minimize the conductance of the resulting partition. However, this approach turned out to be inefficient, as most of the time the resulting clustering had conductance 1 and the numebr of clusters equaled the number of nodes in the coarsest hypergraph.

The second approach was to use a *random* partitioning, where each node of the coarsest hypergraph is randomly assigned to one of the k clusters. For a better qulity of the initial partitioning, we run the random partitioning algorithm 10 times and choose the one with the lowest conductance. this approach was more successful, so it is default in our *cluster*-mode.

4.3 Gain function for Label Propagation

Gain function for the local search algorithm is needed to decide whether to move a node v from one cluster V_i to another cluster V_j of the current partition during the uncoarsening phase. We again tried two approaches.

The first one is called *conductance_local*. Here, the gain of moving a node v from one cluster V_i to another cluster V_j is defined as the difference between the maximal conductance of the cuts ∂V_i and ∂V_j after and before the move.

The second one is called *conductance_global*. The gain of a move here is defined as the difference between the conductance of the partition after and before the move, i. e., the difference between the maximal conductance of all the cuts ∂V_t after and before the move. This is done by looking at the top tree elements of the ConductancePriorityQueue: if the maximal conductance of the clustering will be changed, the new maximum will be one of the clusters V_i, V_j or the cluster with the highest cut conductance apart from these two. As this third cluster is guaranteed to be one of the top three elements of the ConductancePriorityQueue, we simply compute the conductance of the clustering after the move in a constant time and therefore the gain of one move is also computed in a constant time.

In both cases, non-negative gain means that executing the move will make the maximal conductance of the clustering higher. However, as the label propagation algorithm first computes gains for moves of all nodes to all their neighbouring partitions, and then moves active nodes in parallel in the cluster with the best gain, it can so happen that some or all of the precomputed gains are incorrect. Because of this, the algorithm uses **attributed gains** to track the real change in the objective function, i. e., conductance of the clustering. If this change gets bad after a round of node moves, the label propagation will stop and the algorithm will start a new round of uncoarsening.

The original `Mt-KaHyPar` algorithm was implemented with a cut-based objective function, so the attributed gain of a move was atomically updated for each incident hyperedge of the moved node. In our case, **the attributed gain of a move** is updated for each move only once: after the update of the cut weight and the original volumes. To account for concurrent changes of cut weights, during each move a difference between the new and the old cut weights and volumes made by the move is used to get the attributed gain of the concrete move. The **attributed gains** of a move is then calculated as the difference in the conductance of the clustering exactly the same way as in *conductance_local* approach and also runs in constant time for each move.

This concludes the overview of our adjustments to the `Mt-KaHyPar` algorithm. A more detailed (but even more technical) description of the implementation can be found in the `Changes Guide.md` on the GitHub repository of this project.

5 EXPERIMENTAL RESULTS

As we have already mentioned, we compare our implementation with the state of the art modularity based hypergraph clustering algorithm `HypergraphModularity`, because no other conductance-based hypergraph clustering algorithm is known to us.

All the experiments were conducted on a machine consisting of a sixteen-core Intel Xeon Silver 4216 processor running at 2.1 GHz, 100 GB of main memory, 16 MB of L2-Cache, and 22 MB of L3-Cache running Ubuntu 20.04.1. The comparison against `HypergraphModularity` was done on only one cpu on the circuit-based benchmark set `ISPD98` [2], which was previously used to evaluate the conductance-quality of a hypergraph clustering algorithm [1]. The scalability of our implementation was tested on the same machine but on 1, 2, 4, 8, 16 and 32 threads using the large circuit hypergraph `circuit5M.mtx.hgr` STOPPED HERE.

STOPPED HERE The benchmark set consists of 18 hypergraphs

6 CONCLUSION

A APPENDIX

THEOREM A.1. *Concuctance of a k-way partition $V_1, \ldots, V_k$ is the maximal conductance of a cut V_i for $i = 1, \ldots k$ with $V_i \neq \emptyset, V$.*

	k	**HypergraphModularity**		**Conductance-based clustering**			
				conductance_local, random		conductance_global, random	
		φ	Run- and I/O time	φ	Run- and I/O time	φ	Run- and I/O time
ibm02	11	0,3211	18,0384 s + 0,0363 s	**0,2662**	0,2471 s + 0,0175 s	0,9231	0,3123 s + 0,0176 s
ibm03	27	**0,1976**	10,4540 s + 0,0485 s	0,2640	0,2920 s + 0,0190 s	0,9444	0,3105 s + 0,0209 s
ibm04	28	**0,2132**	9,4372 s + 0,0571 s	0,2902	0,3500 s + 0,0215 s	0,9231	0,4271 s + 0,0193 s
ibm05	14	0,3704	13,1119 s + 0,0573 s	**0,2595**	0,4120 s + 0,0222 s	0,8636	0,4514 s + 0,0210 s
ibm06	26	**0,2399**	11,9333 s + 0,2838 s	0,2581	0,4565 s + 0,0213 s	0,9259	0,4611 s + 0,0217 s
ibm07	30	0,2361	19,4188 s + 0,0859 s	**0,2200**	0,5388 s + 0,0291 s	1,0000	0,5722 s + 0,0297 s
ibm08	26	0,3210	108,7750 s + 0,0896 s	**0,2393**	0,6975 s + 0,0330 s	1,0000	0,6873 s + 0,0297 s
ibm09	40	**0,1595**	18,1716 s + 0,1136 s	0,2193	0,6194 s + 0,0342 s	0,9063	0,6618 s + 0,0297 s
ibm10	29	**0,2105**	91,1637 s + 0,1474 s	0,2114	0,8727 s + 0,0404 s	0,8636	0,8972 s + 0,0297 s
ibm11	36	0,2033	29,0756 s + 0,1424 s	**0,1853**	0,8045 s + 0,0401 s	0,8495	0,9233 s + 0,0406 s
ibm12	29	**0,1977**	136,1306 s + 0,3668 s	0,2497	0,9654 s + 0,0429 s	1,0000	0,9946 s + 0,0427 s
ibm13	29	**0,1044**	67,8915 s + 0,4204 s	0,1928	1,0230 s + 0,0499 s	0,8211	1,1253 s + 0,0481 s
ibm14	34	0,1960	180,9870 s + 0,5436 s	**0,1927**	1,7571 s + 0,0774 s	0,9412	1,9409 s + 0,0766 s
ibm15	38	**0,1414**	321,1060 s + 0,4440 s	0,1708	2,2087 s + 0,0973 s	0,9111	2,3470 s + 0,0948 s
ibm16	37	0,2339	746,5931 s + 0,4849 s	**0,1753**	2,4574 s + 0,1036 s	0,8333	2,6329 s + 0,1015 s
ibm17	47	**0,2063**	789,4657 s + 0,4700 s	0,2071	2,9459 s + 0,1047 s	0,9412	2,9660 s + 0,1068 s
ibm18	48	**0,0944**	977,1867 s + 0,5143 s	0,1464	2,8057 s + 0,1083 s	0,8889	3,1900 s + 0,1114 s

Table 1. Comparison of our conductance-based clustering with `HypergraphModularity`. k for each instance is the number of clusters found by the `HypergraphModularity` algorithm. The best results are marked in bold.

Proof. Without loss of generality, we can assume that the clusters V_i are non-empty (otherwise, we could remove the empty clusters from the partition whothout changing on eitther side of the equation discussed). Per definition of conductance of a k-way partition, there exists a subset $\emptyset \subsetneq I \subsetneq \{1,\dots,k\}$ such that $\varphi(V_1,\dots V_k) = \varphi(\cup_{i\in I} V_i)$. Without loss of generality, we can further assume that the volume of the union of the clusters V_i for $i \in I$ is less than or equal to the volume of its complement, i. e., the volume of the union of all the clusters V_i for $i \notin I$. Then we can write:

$$\begin{aligned}
\varphi(V_1,\dots,V_k) &= \frac{\omega(\partial(\cup_{i\in I} V_i))}{\min(\mathrm{vol}(\cup_{i\in I} V_i), \mathrm{vol}(\cup_{i\notin I} V_i))} = \frac{\sum\{\omega(e) : e - \text{a cutting edge of } \partial(\cup_{i\in I} V_i)\}}{\mathrm{vol}(\cup_{i\in I} V_i)} \\
&\le \frac{\sum_{i\in I} \omega(\partial V_i)}{\sum_{i\in I} \mathrm{vol}(V_i)} = \frac{\sum_{i\in I} \mathrm{vol}(V_i) \cdot \frac{\omega(\partial V_i)}{\mathrm{vol}(V_i)}}{\sum_{i\in I} \mathrm{vol}(V_i)} \le \max_{i\in I} \frac{\omega(\partial V_i)}{\mathrm{vol}(V_i)} \\
&\le \max_{i\in I} \frac{\omega(\partial V_i)}{\min(\mathrm{vol}(V_i), \mathrm{vol}(V \backslash V_i))} = \max_{i\in I} \varphi(V_i) \\
&\le \varphi(V_1,\dots,V_k)
\end{aligned}$$

Thus, all abothe written inequalities are equalities, which means that the conductance of the k-way partition is equal to the maximal conductance of a cut V_i for $i = 1,\dots k$.

It is interesting to note that with this we have also proven that the conductance of a hypergraph partition is equal to the maximum ratio of the weight of a cut V_i to the volume of the cluster V_i over all the clusters $V_1,\dots V_k$. This property, however, was not used in the implementation due to late discovery. □

B GAIN FOR LOCAL SEARCH: NAIVE?

Information needed:

- the sum of all weights of the nets: $vol(V) = const$;
- <u>for each node</u>:
  - incident nets;
  - weighted degree;
- <u>for each net e</u>:
  - number of pins in e: $|e|$;
  - number of pins of e which are currently in cluster V_i for each cluster index i: $e.\#pins(V_i)$;
- <u>for each cluster index i</u>:
  - current volume: $vol(V_i)$;
  - weights sum of cutting edges of the current V_i: $V_i.out\_weight$.

Time needed: $O(k \cdot deg(v) + k \cdot \log k)$...

Calculating the gain for moving node v from V_i to V_j:

(1) (temporary) adjust volumes of V_i and V_j: $\Theta(1)$

$$vol(V_i)' \leftarrow vol(V_i) - \deg_\omega(v)$$

$$vol(V_j)' \leftarrow vol(V_j) + \deg_\omega(v)$$

(2) (temporary) adjust $Vi.out\_weight$ and analogously for $Vj.out\_weight$ (but with reversed signs of $\omega(e)$ and $0, |e| - 1$ in conditions in ifs): $\Theta(\deg(v))$

```
For each incident to $v$ net $e \in E$:
    if $1 = e.\#pins(V_i) < |e|$:
        $e$ is a cutting net for the cut $V_i$, $v$ - its last pin in $V_i$
        $\Longrightarrow$ $e$ won't be a cutting edge after removal of $v$:
            $Vi.out\_weight' \leftarrow V_i.out\_weight - \omega(e)$
    else if $1 < e.\#pins(V_i) = |e|$:
        $e$ was not a cutting net for the cut $V_i$, but it will be after the removal of $v$:
            $Vi.out\_weight' \leftarrow V_i.out\_weight + \omega(e)$
```

(3) calculate new conductances of cuts V_i and (analogously) V_j: $\Theta(1)$

$$\varphi(V_i) = \frac{V_i.out\_weight'}{\min\{vol(V_i)', vol(V) - vol(V_i)'\}}$$

The question is, what should we take for gain from moving v:

(1) decrease in maximal conductance between the cuts V_i and V_j: *easy*;
(2) decrease in the overall maximal conductance: a PQ with k current conductances $\varphi(V_1), \ldots, \varphi(V_k)$ - additional $O(k \log k)$ for finding the best V_j.

$\Longrightarrow$ maybe the best according to criterion 1 from the best according to criterion 2... (***I believe, I should try all these variants and possibly some other...***)

A PQ with k current conductances $\varphi(V_1), \ldots, \varphi(V_k)$ is needed (?) to calculate the contribution of a net to the overall conductance of the partition (?):

C CONTRIBUTION OF A NET TO THE OVERALL CONDUCTANCE

Objective function: conductance of the partition (also naive?):

- each edge e from the most expensive conduction-wise cut V_i contributes by

$$\frac{\omega(e)}{\max\{vol(V_i), vol(V) - vol(V_i)\}}$$

- all other edges contribute by 0 (to ensure that the sum of all contributions is the current conductance...)

REFERENCES

[1] Ali Aghdaei, Zhiqiang Zhao, and Zhuo Feng. Hypersf: Spectral hypergraph coarsening via flow-based local clustering. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2021.

[2] Charles J. Alpert. The ispd98 circuit benchmark suite. In *Proceedings of the 1998 International Symposium on Physical Design*, ISPD '98, page 80–85, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 158113021X. doi: 10.1145/274535.274546. URL https://doi.org/10.1145/274535.274546.

[3] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *2006 47th annual IEEE symposium on foundations of computer science (FOCS'06)*, pages 475–486. IEEE, 2006.

[4] Philip S Chodrow, Nate Veldt, and Austin R Benson. Generative hypergraph clustering: From blockmodels to modularity. *Science Advances*, 7(28):eabh1303, 2021.

[5] Lars Gottesbüren, Tobias Heuer, Peter Sanders, and Sebastian Schlag. Scalable shared-memory hypergraph partitioning. *CoRR*, abs/2010.10272, 2020. URL https://arxiv.org/abs/2010.10272.

[6] Sungwoong Kim, Sebastian Nowozin, Pushmeet Kohli, and Chang Yoo. Higher-order correlation clustering for image segmentation. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper_files/paper/2011/file/98d6f58ab0dafbb86b083a001561bb34-Paper.pdf.

[7] Kyle Kloster and David F. Gleich. Heat kernel based community detection. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 1386–1395, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329569. doi: 10.1145/2623330.2623706. URL https://doi.org/10.1145/2623330.2623706.

[8] Longlong Lin, Ronghua Li, and Tao Jia. Scalable and effective conductance-based graph clustering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 4471–4478, 2023.

[9] Veronica Poda and Catherine Matias. Comparison of modularity-based approaches for nodes clustering in hypergraphs. *Peer Community Journal*, 4, 2024.

[10] Daniel A Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on computing*, 42(1):1–26, 2013.

[11] Ze Tian, TaeHyun Hwang, and Rui Kuang. A hypergraph-based learning algorithm for classifying gene expression and arraycgh data with prior knowledge. *Bioinformatics*, 25(21):2831–2838, 07 2009. ISSN 1367-4803. doi: 10.1093/bioinformatics/btp467. URL https://doi.org/10.1093/bioinformatics/btp467.

[12] Dengyong Zhou, Jiayuan Huang, and Bernhard Schölkopf. Learning with hypergraphs: Clustering, classification, and embedding. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems*, volume 19. MIT Press, 2006. URL https://proceedings.neurips.cc/paper_files/paper/2006/file/dff8e9c2ac33381546d96deea9922999-Paper.pdf.