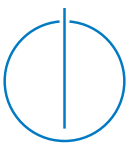
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

# Spectral Methods to Find Small Expansion Sets on Hypergraphs

Franz Rieger



DEPARTMENT OF INFORMATICS

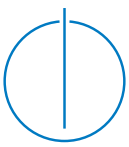
TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

# Spectral Methods to Find Small Expansion Sets on Hypergraphs

# Spektrale Methoden zum Finden kleiner Expansionsmengen auf Hypergraphen

| | |
|---|---|
| Author: | Franz Rieger |
| Supervisor: | Prof. Susanne Albers |
| Advisor: | Dr. T.-H. Hubert Chan |
| Submission Date: | 15. January 2019 |



I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15. January 2019 Franz Rieger

# Acknowledgments

This thesis was written under the supervision of Dr. T.-H. Hubert Chan at the University of Hong Kong. TODO: ideas from him

# Abstract

The problem of finding a small Edge Expansion on a graph can also be defined on hypergraphs. In this thesis approximation algorithms for obtaining sets with a small Edge Expansion are discussed and implemented.

# Contents

# 1 Introduction

...

## 1.1 Simple Graphs

In graph theory a graph $G := (V, E)$ is defined as a set of $n$ vertices $V = \{v_1, \ldots, v_n\}$ and a set of $m$ edges $E = \{e_1, \ldots, e_m\}$ where each edge $e_i = \{v_k, v_l\} \in E$ connects two vertices $v_k, v_l \in V$. Some of the simplest graphs can be seen in ... (TODO:figure)

## 1.2 Hypergraphs

This thesis will deal with a generalized form of simple graphs, namely hypergraphs.

A weighted, undirected hypergraph $H = (V, E, w)$ consists of a set of $n$ vertices $V = \{v_1, \ldots, v_n\}$ and a set of $m$ (hyper-)edges $E = \{e_1, \ldots, e_m | \forall i \in [i] : e_i \subseteq V \wedge e_i \neq \emptyset\}$ where every edge $e$ is a non-empty subset of $V$ and has a positive weight $w_e := w(e)$, defined by the weight function $w : E \to \mathbb{R}_+$.

## 1.3 Cuts

On such graphs certain properties can be described, which are of theoretical interest but also have influence on the behaviour of a system which is described by such a graph. Some of these properties are so called cuts. A cut is described by its cut-set $\emptyset \neq S \subsetneq V$ a non-empty true subset of the vertices. Interesting cuts are for example the so called Minumum cut or the maximum cut which are defined by the minimum/maximum number of edges (or their added weight for weighted graphs) going between $S$ and $V \setminus S$. Formally $MinCut(G) := \min_{\emptyset \subsetneq S \subsetneq V} \sum_{e \in E : \exists u,v \in e : u \in S \wedge v \in V \setminus S} w_e$. For the min-cut there exists a polynomial time (in the number of vertices) algorithm TODO: find citation The max cut problem is known to be NP hard

Sparse cut: crossing edge weights/ minw(S), w(V S) The cut on which this thesis focuses on is the so called Edge Expansion, which is the quotient of the summed weight of the edges crossing S and V S and the summed weight of all the edges in S.

This is also a np hard problem ? Therefore several approximation algorithms exist, which will be shown in the following Out of Chan's proof that an algorithm with certain properties exists will be used to extract that algorithm. The involved constants will be estimated in a empirical manner by running it multiple times on different random graphs (for which algorithms are evaluated)

TODO: how to work with notation of next chapter here: minium TODO: example graphs (also example dataset?) TODO: Mincut, Sparsest Cut, Edge expansion

For normal graphs Np-Hard [Kai04]

# 2 Notation

The notation used in this thesis is orientated on [Cha+16].

The weight $w_v$ of a vertex $v$ is defined by summing up the weights of its edges: $w_v = \sum_{e \in E: v \in e} w_e$. Accordingly, a subset $S \subseteq V$ of vertices has weight $w_S := \sum_{v \in S}$ and a subset $F \subseteq E$ of edges has weight $w_F = \sum_{e \in F} w_e$. The set of edges which are cut by $S$ is defined as $\partial S := \{e \in E : e \cap S \neq \emptyset \wedge e \cap V \setminus S \neq \emptyset\}$, which contains all the edges, which have at least one vertex in $S$ and at least one vertex in $V \setminus S$. The edge expansion of a non-empty set of vertices $S \subseteq V$ is defined by

$$\Phi(S) := \frac{w(\partial S)}{w(S)}. \tag{2.1}$$

Observe that $\forall \emptyset \neq S \subset V : 0 \leq \Phi(S) \leq 1$. The first inequality holds because the edge-weights are positive. The second inequality holds because $W(S) \geq W(\partial S)$, as $W(S)$ takes at least every edge (and therefore the corresponding weight), which is also considered by $W(\partial S)$, into account.

With this, the expansion of a graph $H$ is defined as

$$\Phi(H) := \min_{\emptyset \subsetneq S \subsetneq V} \max\{\Phi(S), \Phi(V \setminus S)\}. \tag{2.2}$$

Here again, $0 \leq \Phi(H) \leq 1$ holds. For not connected graphs $\Phi(H) = 0$, which can be verified by observing a $S$ which only contains vertices of one connection component. Therefore, only connected graphs shall be of interest here. Observe that for a graph $H$, which is obtained by connecting two connection components with edge with small weight, $\Phi(H)$ takes a small value. For a fully connected graph with equal edge-weights, $\partial S$ (and therefore $\Phi(S)$) will be big for every $S \subsetneq V$.

The weight matrix can be denoted as

$$W = \begin{pmatrix} w_{v_1} & 0 & 0 & \dots & 0 \\ 0 & w_{v_2} & 0 & \dots & 0 \\ 0 & 0 & w_{v_3} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & w_{v_n} \end{pmatrix} \in \mathbb{R}_{0+}^{n \times n}$$

The discrepancy ratio of a graph, given a non-zero vector $f \in \mathbb{R}^V$ is defined as

$$D_w(f) := \frac{\sum_{e \in E} w_e \max_{u,v \in e}(f_u - f_v)^2}{\sum_{u \in V} w_u f_u^2}$$

.

In the weighted space, in which the discrepancy ratio is defined like above, for two vectors $f, g \in \mathbb{R}^V$ the inner product is defined as $\langle f, g \rangle_w := f^T W g$. Accordingly, the norm is $||f||_w = \sqrt{\langle f, f \rangle_w}$. If $\langle f, g \rangle_w = 0$, $f$ and $g$ are said to be orthonormal in the weighted space.

# 3 Algorithms

In the following chapter different approaches for generating small expansion sets $S$ will be discussed. TODO: why phi (S) and not phi(H)?

## 3.1 Brute force

One obvoius approach is to brute-force the problem:

**Algorithm 1** Brute-force

```
best_S := null
lowest_expansion := inf
for ∅ ≠ S ⊊ V do
    expansion := Φ(S)
    if expansion < lowest_expansion then
        lowest_expansion := expansion
        best_S := S
return best_S
```

Correctness: This as this algorithm iterates over all $\emptyset \neq S \subsetneq V$, it computes $\arg\min_{\emptyset \subsetneq S \subsetneq V} \Phi(S)$.

TODO: what else to prove?

Complexity: There are $2^{|V|} - 2 = 2^n - 2 \in O(2^n)$ combinations for $\emptyset \neq S \subsetneq V$, namely all the $2^{|V|}$ subsets of $V$ excluding the empty set $\emptyset$ and $V$ itself. Therefore, this algorithm is of exponential time complexity in $n$ and is therefore not efficient for larger graphs.

TODO: refine brute-force to only $\phi(S)$ not $\phi(H)$ possibly with $a < |S| < b$

## 3.2 Orthonormal vectors

As described in [Cha+16], the following algorithm can be used:

**Fact 3.2.1** *Theorem 6.6 in [Cha+16] Given an a hypergraph $H = (V, E, w)$ and k vectors $f_1, f_2, \ldots, f_k$ which are orthonormal in the weighted space with $\max_{s \in [k]} D_w(f_s) \leq \xi$, the*

**Algorithm 2** Small Set Expansion (according to Algorithm 1 in [Cha+16])

**function** SMALLSETEXPANSION($G := (V, E, w), f_1, \ldots, f_k$)
  assert $\xi == \max_{s\in[k]}\{D_w(f_s)\}$
  assert $\forall f_i, f_j \in \{f_1, \ldots, f_k\} \subset \mathbb{R}^n, i \neq j : f_i$ and $f_j$ orthonormal in weighted space
  **for** $i \in V$ **do**
    **for** $s \in [k]$ **do**
      $u_i(s) := f_s(i)$
  **for** $i \in V$ **do**
    $\tilde{u}_i := \frac{u_i}{||u_i||}$
  $\hat{S} :=$ ORTHOGONALSEPARATOR($\{\tilde{u}_i\}_{i\in V}, \beta = \frac{99}{100}, \tau = k$ )
  **for** $i \in S$ **do**
    **if** $\tilde{u}_i \in \hat{S}$ **then**
      $X_i := ||u_i||^2$
    **else**
      $X_i := 0$
  $X :=$ sort $list(\{X_i\}_{i\in V})$
  $V := [i]_{\text{in order of X}}$
  $S := \arg\min_{\{P:O \text{ is prefix of } V\}} \phi(O)$
  return $S$

*following holds. algorithm 2 constructs a random set $S \subsetneq V$ in polynomial time such that with $\Omega(1)$ probability, $|S| \leq \frac{24|V|}{k}$ and*

$$\phi(S) \leq C \min\{\sqrt{r \log k}, k \log k \log\log k \sqrt{\log r}\} \cdot \sqrt{\xi},$$

*where C is an absolute constant and $r := \max_{e\in E} |e|$.*

With the following way to create orthonormal vectors, algorithm ... can be executed. This results in ... according to ...

$$\begin{aligned} \underset{g}{\text{minimize}} \quad & \text{SDPval} := \sum_{e\in E} w_e \max_{u,v\in e} ||\vec{g_u} - \vec{g_v}||^2 \\ \text{subject to} \quad & \sum_{u\in V} w_v ||\vec{g_v}||^2 = 1, \\ & \sum_{u\in V} w_v f_i(v) \vec{g_v} = \vec{0}, \quad \forall i \in [k-1] \end{aligned} \tag{3.1}$$

SDP

So for a given Graph $H$ we can find a small expansion set with the following algorithm:

**Algorithm 3** Orthogonal Separator (combination of Lemma 18 and algorithm Theorem 10 in [LM14] (also Fact 6.7 in [Cha+16]))

**function** OrthogonalSeparator($\{\tilde{u}_i\}_{i\in V}, \beta = \frac{99}{100}, \tau = k$)
  $l := \lceil \frac{\log_2 k}{1-\log_2 k} \rceil$
  $g \sim \mathcal{N}(0, I_n)$ where each component $g_i$ is mutually independent and sampled from $\mathcal{N}(0,1)$
  $w :=$SampleAssignments$(l, V, \beta)$
  **for** $i \in V$ **do**
    $W(u) := w_1(u)w_2(u)\cdots w_j(u)$
  **if** $n \geq 2^l$ **then**
    $word := random(\{0,1\}^l)$ uniform
  **else**
    $words := set(w(i) : i \in V)$ no multiset
    $words \cup = \{w_1, \ldots, w_{|V|-|words|} \in \{0,1\}^l\}$ random choice
    $word := random(words)$ uniform
  $r := uniform(0,1)$
  $S := \{i \in V : ||i||^2 \geq r \wedge W(u) = word\}$
  return $S$

**Algorithm 4** Sample Assignments (proof of Lemma 18 in [LM14])

**function** SampleAssignments$(l, V, \beta)$
  $\lambda := \frac{1}{\sqrt{\beta}}$
  **for** $j = 1, 2, \ldots, l$ **do**
    **for** $i \in V$ **do**
      $t_i := \langle g, \tilde{u}_i \rangle$
      $poisson\_count_i := N(t_i, \lambda)$ where N is a poisson process on $\mathbb{R}$ (same process for each call)
      **if** $poisson\_count_i \mod 2 == 0$ **then**
        $w_j(i) := 1$
      **else**
        $w_j(i) := 0$
  return $w$

**Algorithm 5** Rounding Algorithm for Computing Eigenvalues (Algorithm 3 in [Cha+16])

**function** SAMPLERANDOMVECTORS($H$)
  Solve SDP 3.1 to generate vectors $\vec{g_v} \in \mathbb{R}^n$ for $v \in V$
  $\vec{z} := sample(\mathcal{N}(0, 1^n))$
  **for** $v \in V$ **do**
    $f(v) :=< \vec{g_v}, \vec{z} >$
  return f

**Algorithm 6** Find Small Expansion Set

**function** SES($H$)
  $f :=$SAMPLERANDOMVECTORS($H$)
  return SMALLSETEXPANSION($H, f$)

# 4 Random Hypergraphs

The initial intention for creating random r-uniform, d-regular, connected hypergraphs in an effective manner which is guaranteed to terminate showed to be not as a trivial task, which is why several different approaches will be used and their resulting graphs also characterized by their edge expansion

To start with, a simple algorithm to generate graphs which follows some of the intentions will be discussed:

**Algorithm 7** Generate random graph

**function** GENERATERANDOMGRAPH($n, rank, numberEdges, weightDistribution$)
  $E := \emptyset$
  $V := \{v_1, \ldots, v_n\}$
  **for** $1, \ldots, numberEdges$ **do**
    $nextEdgeVertices := sample(V, rank)$ ▷ draw without replacement
    $nextEdgeWeight := sample(weightDistribution)$
    $nextEdge := (nextEdgeVertices, nextEdgeWeight)$
    $E := E \cup \{nextEdge\}$

The algorithm is terminating, quick and the resulting graph is r-uniform and all the possible graphs can be constructed. But it might never sample one vertex $v \in V$, therefore the rank of this vertex will be 0 which does not make the graph regular and also not connected. Also, it does not guarantee to have no doubled edges.

This idea can be improved by ensuring the degree of the vertices do not exceed $d$:

The algorithm is terminating, quick and the resulting graph is r-uniform and all the possible graphs can be constructed. However it is not guaranteed that this graph is connected and it is possible that some ($< rank$) vertices do not have degree $d$ in the end, because they have not been sampled before. TODO: example Also, it does not guarantee to have no doubled edges.

To overcome these problems, the edges could only be sampled from the vertices with the smallest degrees:

The algorithm is guaranteed to terminate, quick and the resulting graph is r-uniform and d-regular. However, not all the possible graphs can be constructed: This algorithm basically constructs the edges by d r-matchings. But not every graph can be dissembled

**Algorithm 8** Generate random graph

**function** GENERATERANDOMGRAPH($n, rank, d, weightDistribution$)
$E := \emptyset$
$V := \{v_1, \ldots, v_n\}$
**while** $|\{v \in V | deg(v) < d\}| \geq rank$ **do**
$nextEdgeVertices := sample(\{v \in V | deg(v) < d\}, rank)$ ▷ draw without replacement
$nextEdgeWeight := sample(weightDistribution)$
$nextEdge := (nextEdgeVertices, nextEdgeWeight)$
$E := E \cup \{nextEdge\}$

**Algorithm 9** Generate random graph

**function** GENERATERANDOMGRAPH($n, rank, d, weightDistribution$)
$E := \emptyset$
$V := \{v_1, \ldots, v_n\}$
**while** $|\{v \in V | deg(v) < d\}| \geq rank$ **do**
$smallestDegreeVertices := \{v \in V | deg(v) = \min_{u \in V} deg(u)\}$
**if** $|smallestDegreeVertices| >= rank$ **then**
$nextEdgeVertices := sample(smallestDegreeVertices, rank)$ ▷ draw without replacement
**else**
$secondSmallestDegreeVertices := \{v \in V | deg(v) = \min_{u \in V} deg(u) + 1\}$
$nextEdgeVertices := sample(secondSmallestDegreeVertices, rank - |smallestDegreeVertices|)$
$nextEdgeVertices := smallestDegreeVertices \cup nextEdgeVertices$
$nextEdgeWeight := sample(weightDistribution)$
$nextEdge := nextEdgeVertices$
$E := E \cup \{nextEdge\}$
$w(e) := nextEdgeWeight$
return $G := (V, E, w)$

into d r-matchings. Again this graph is not neccessarily connected and some edges might be doubled.

To solve this, there are several options: One could i) resample whole graph (if probability is > constant), losing the terminating property. ii) resample some edges (from different connection components), ideally only strongly connected vertices, also losing the terminating property. Proof: all vertices are strongly connected to their

connection component? iii) creating a spanning tree first and then sampling further

i)

**Algorithm 10** Generate random graph

**function** GenerateRandomGraph($n, rank, d, weightDistribution$)
  $G :=$ GenerateRandomGraph($n, rank, d, weightDistribution$)
  **while** ∄Connected(G) or $\exists e, f \in E.e = f$ **do**
    $G :=$ GenerateRandomGraph($n, rank, d, weightDistribution$)
  return $G := (V, E)$

ii)

**Algorithm 11** Generate random graph

**function** GenerateRandomGraph($n, rank, d, weightDistribution$)
  $G :=$ GenerateRandomGraph($n, rank, d, weightDistribution$)
  **while** ∄Connected(G) or $\exists e, f \in E.e = f$ **do**
    $e, f := sample(E, 2)$
    $u := sample(e)$
    $v := sample(f)$
    $e := (e \cup \{v\}) \setminus \{u\}$
    $f := (f \cup \{u\}) \setminus \{v\}$
  return $G := (V, E, w)$

iii)

However it is not guaranteed that this graph is connected and it is possible that some ($< rank$) vertices do not have degree $d$ in the end, because they have not been sampled before.

TODO: define quick

TODO: Discuss different approaches of generating, their limitations

TODO: Analyze $\Phi$ for different random- classes? (and explain?)

## 4.1 random edges with discard if not connected

## 4.2 connected edges with discard if not connected or not regular

**Algorithm 12** Generate random graph

**function** GENERATERANDOMGRAPH($n, rank, d, weightDistribution$)
$V := \{v_1, \ldots, v_n\}$
$E := choice(V, rank)$
**while** $\{v \in V | deg(v) = 0\} \neq \emptyset$ **do**
**if** $|\{v \in V | deg(v) = 0\}| \geq rank$ **then**
$nextEdgeTreeVertex := choice(\{v \in V | deg(v) = 1\})$ ▷ get one tree node
$nextEdgeVertices := choice(\{v \in V | deg(v) = 0\}, rank - 1) \cup \{nextEdgeTreeVertex\}$
**else**
$nextEdgeVertices := \{v \in V | deg(v) = 0\}, \cup choice(\{v \in V | deg(v) > 0\}, |\{v \in V | deg(v) = 0\}|)$
$nextEdgeWeight := sample(weightDistribution)$
$nextEdge := nextEdgeVertices$
$E := E \cup \{nextEdge\}$
$w(e) := nextEdgeWeight$
**while** $|\{v \in V | deg(v) < d\}| \geq rank$ **do**
$smallestDegreeVertices := \{v \in V | deg(v) = \min_{u \in V} deg(u)\}$
**if** $|smallestDegreeVertices| >= rank$ **then**
$nextEdgeVertices := sample(smallestDegreeVertices, rank)$ ▷ draw without replacement
**else**
$secondSmallestDegreeVertices := \{v \in V | deg(v) = \min_{u \in V} deg(u) + 1\}$
$nextEdgeVertices := sample(secondSmallestDegreeVertices, rank - |smallestDegreeVertices|)$
$nextEdgeVertices := smallestDegreeVertices \cup nextEdgeVertices$
$nextEdgeWeight := sample(weightDistribution)$
$nextEdge := nextEdgeVertices$
$E := E \cup \{nextEdge\}$
$w(e) := nextEdgeWeight$
return $G := (V, E, w)$

# 5 Implementation

## 5.1 Technologies

Python with Nump, Scipy as optimizer

## 5.2 Code

Own hypergraph implementation
TODO: how to reference code? Github? TODO: what all to explain

# 6 Evaluation

## 6.1 Graph size

For evaluating the implementation of the algorithm ... (chan's) in comparison to the brute-force solution, the maximal size of graphs (in $n$) which can be brute-forced in a reasonable time is determined. In general it is favourable to generate as large as possible graphs, for not needing to extrapolate ... However, the brute forcing time correlates with around $n^8$ (see ...), so with the available resources, it already takes ...s for n=...

For estimating the constant in theorem ..., the a plot of as many grapsh as possible seems to be ideal. So not only for brute-forcing but also for executing algorithm ... oftenly (...), the graph shouldn't be too big, as the (improveable) time complexity of the implementation shows to be at around ... $n$...

What time is reasonable is influenced by the following trade-off: either one wants to know the expansion of

in conclusion 1 minute seems to be a reasinable time for one run of the algorithm.

So the size of the graph is fixed to 20.

In order to not generate very dense graphs but also demonstrate the hypergraph property, the rank of edges is set to 3 and the degree of vertices is set to be ...

This can't be chosen freely, as For other combinations no r-uniform d-regular graphs exist on .. vertices, as $nd = mr$ (todo: source)

## 6.2 random Generation methods

As the expansion for not connected graphs is always 0, only algorithms which guarantee connected graphs will be considered.

As it proved difficult to re-generate graphs... only the three algorithms ... , ... and ... will be used. The expansion of the graphs will be evaluated against each other via brute-force and using the approximation alorithm as well.

The edge-weight distribution is always set to be a uniform distribution on $[0.1, 1.1]$

## 6.3 title

TODO: find constants by analyzing quality? Analyze runtime of code?

# 7 Applications

TODO: groups in social network discussions (how to cite discussion?) Learning?
Rummikub: which stones fit to others (creating a hypergraph)

# 8 Resume and Further Work

Improve implementation time complexity (use different solver?) Estimate constant more efficiently Implement and compare to other algorithms Evaluate on other graphs size, denser / less dense, weights, different way of generating

# List of Figures

# List of Tables

# Bibliography

[Cha+16] T. H. Chan, A. Louis, Z. G. Tang, and C. Zhang. "Spectral Properties of Hypergraph Laplacian and Approximation Algorithms." In: *CoRR* abs/1605.01483 (2016). arXiv: 1605.01483.

[Kai04] V. Kaibel. "On the expansion of graphs of 0/1-polytopes." In: *The Sharpest Cut: The Impact of Manfred Padberg and His Work*. SIAM, 2004, pp. 199–216.

[LM14] A. Louis and Y. Makarychev. "Approximation Algorithms for Hypergraph Small Set Expansion and Small Set Vertex Expansion." In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*. Ed. by K. Jansen, J. D. P. Rolim, N. R. Devanur, and C. Moore. Vol. 28. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2014, pp. 339–355. ISBN: 978-3-939897-74-3. DOI: 10.4230/LIPIcs.APPROX-RANDOM.2014.339.