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. March 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. March 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

To introduce the reader to the topic, a short introduction to graphs and their generalization hypergraphs is given. Afterwards, the proplem of cuts, especially edge expansion, shall be introduced.

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 $v_k, v_l \in V$. A simple graph can be seen in fig. 1.1. Note that in this thesis, an edge is not displayed as a line between the vertices but as a coloured shape around the vertices.

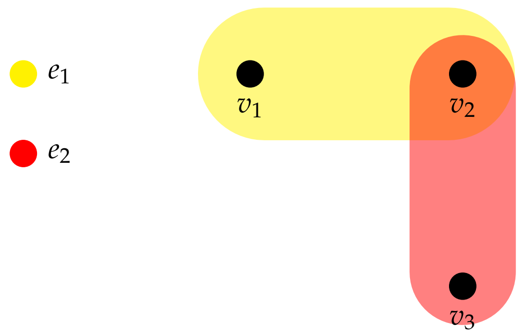

Figure 1.1: An example for a simple graph with three vertices and two edges $G = (\{v_1, v_2, v_3\}, \{\{v_1, v_2\}, \{v_2, v_3\}\})$

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 $E = \{e_1, \ldots, e_m | \forall i \in [i] : e_i \subseteq V \land e_i \neq \emptyset\}$ 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 $w : E \rightarrow \mathbb{R}_+$. An example for a hypergraph can be seen in fig. 1.2.

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 $w_F = \sum_{e \in F} w_e$.

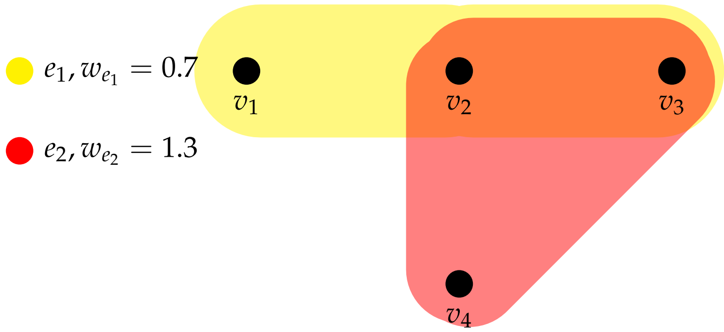

Figure 1.2: An example for a simple hypergraph with four vertices and two hyperedges $G = (\{v_1, v_2, v_3, v_4\}, \{\{v_1, v_2, v_3\}, \{v_2, v_3, v_4\}\})$

1.3 Cuts

On such hypergraphs 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 strict subset of the vertices. Interesting cuts are for example the so called minumum cut or the maximum cut which are defined by the minimum (or maximum respectively) number of edges (or their added weight for weighted graphs) going between S and $V \setminus S$. Formally, this can be expressed by the following equation:

$$MinCut(G) := \min_{\emptyset \subsetneq S \subsetneq V} \sum_{e \in E : \exists u, v \in e : u \in S \wedge v \in V \setminus S} w_e \tag{1.1}$$

For computing the minimum cut the Stoer–Wagner algorithm can be used, which has a polynomial time complexity in the number of vertices [1]. The maximum cut problem however is known to be NP-hard [2].

1.4 Edge Expansion

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 $V \setminus S$ and the summed weight of all the edges in S. The formal notation is introduuced in the following.

The set of edges which are cut by S contains all the edges, which have at least one vertex in S and at least one vertex in $V \setminus S$ and is defined as

$$\partial S := \{e \in E : e \cap S \neq \emptyset \wedge e \cap (V \setminus S) \neq \emptyset\}. \tag{1.2}$$

The edge expansion of a non-empty set of vertices $S \subseteq V$ is defined by

$$\Phi(S) := \frac{w(\partial S)}{w(S)}. \tag{1.3}$$

Observe that $\Phi(S)$ is bounded:

$$\forall \emptyset \neq S \subset V : 0 \leq \Phi(S) \leq 1 \tag{1.4}$$

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

$$\Phi(H) := \min_{\emptyset \subsetneq S \subsetneq V} \max\{\Phi(S), \Phi(V \setminus S)\}. \tag{1.5}$$

Here again, $0 \leq \Phi(H) \leq 1$ holds because of eq. (1.4).

In order to understand the edge expansion of a graph better, some special cases shall be considered. For not connected graphs $\Phi(H) = 0$ holds, which can be verified by observing a S which only contains vertices of one connection component, for example in fig. 1.3. Therefore, only connected graphs shall be of interest here.

Observe that for a graph H, which is obtained by connecting two connection components with an edge with small weight, $\Phi(H)$ takes a small value, which can be seen when S is chosen to be one of the previously seperated connection components. For a fully connected graph with equal edge-weights, ∂S will be big for every $S \subsetneq V$. Therefore $\Phi(S)$ and ultimately also $\Phi(H)$ will take a large value.

The problem of computing the expansion $\Phi(H)$ on a hypergraph is NP-hard, as it is already NP-hard on 2-uniform-graphs, a special case of hypergraphs [3]. However, there exist polynomial time approximation algorithms for some relaxations of this problem, one of them will be focused on here: For certain applications, it can be interesting to find small expansion sets S, where the vertices are strongly connected within the set but only have a weak connection to the rest of the vertices. Small refers to the number of vertices, so $|S|$ should be low. In the presented algorithm sets which have at max a constant fraction $\frac{1}{c}$ of the total number of vertices $|V|$ are computed, formally $|S| \leq \frac{|V|}{c}$. Furthermore, strong and weak connections are determined by $\Phi(S)$

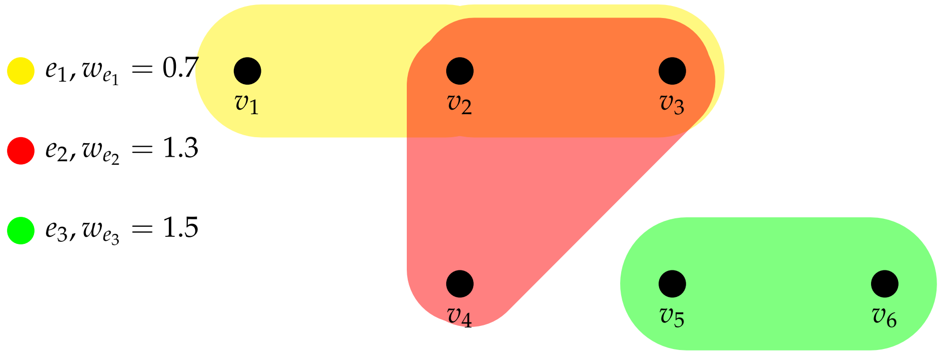

Figure 1.3: An example for a non connected hypergraph with two connection components. For $S := \{v_5, v_6\}$ it can be verified that $\delta S = 0$, hence $\Phi(S) = \Phi(V \setminus S) = 0$.

here. Finding such a S will be achieved by algorithm 9, which was deducted from results from Chan in [4].

The involved constants will be estimated in a empirical manner by running it multiple times on different random graphs (for which algorithms are evaluated)

Sparse cut: crossing edge weights/ minw(S), w(V S)

TODO: other approximations? TODO: Mincut, Sparsest Cut, Edge expansion

2 Notation

The notation used in this thesis is orientated on [4].

The weight matrix can be denoted as

$$W = \begin{pmatrix} w_{v_1} & 0 & 0 & \dots & 0 \\ 0 & w_{v_2} & 0 & \dots & 0 \\ 0 & 0 & w_{v_3} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & w_{v_n} \end{pmatrix} \in \mathbb{R}_{0+}^{n \times n}. \tag{2.1}$$

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

$$D_w(f) := \frac{\sum_{e \in E} w_e \max_{u,v \in e} (f_u - f_v)^2}{\sum_{u \in V} w_u f_u^2}. \tag{2.2}$$

Observe that $0 \leq D_w(f) \leq 2$ [4].

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 as well as the edge expansion of hypergraphs are discussed.

3.1 Brute force

One obvoius approach for generating the edge expansion $\Phi(H)$ of a hypergraph H is to brute-force the problem like in algorithm 1.

Algorithm 1 Brute-force edge expansion on a hypergraph

function BruteForceEdgeExpansion($H := (V, E, w)$)
  $best\_S := null$
  $lowest\_expansion := \infty$
  for $\emptyset \neq S \subsetneq V$ **do**
    $expansion := \max \Phi(S), \Phi(V \setminus S)$
    if $expansion < lowest\_expansion$ **then**
      $lowest\_expansion := expansion$
      $best\_S := S$
  return $best\_S$

As algorithm 1 iterates over all $\emptyset \neq S \subsetneq V$, it computes $\arg\min_{\emptyset \subsetneq S \subsetneq V} \max\left(\Phi(S), \Phi(V \setminus S)\right)$. 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. Hence, this algorithm is of exponential time complexity in n and is therefore not efficient for larger graphs as evaluated in ....

For the purpose of analyzing the graph creation algorithms in chapter 4, it can be insightful to observe the lowest expansion of each possible size (in the number of vertices) like in algorithm 2.

Algorithm 2 Brute-force edge expansion of a hypergraph for every size

function BruteForceEdgeExpansionSizes($H := (V, E, w)$)
  $best\_S_o f_s ize := \{\}$
  $lowest\_expansion\_of\_size := \{1 : \infty, 2 : \infty, \ldots, n-1 : \infty\}$
  for $\emptyset \neq S \subsetneq V$ **do**
    $expansion := \max \Phi(S), \Phi(V \setminus S)$
    if $expansion < lowest\_expansion\_of\_size[|S|]$ **then**
      $lowest\_expansion\_of\_size[|S|] := expansion$
      $best\_S\_of\_size[|S|] := S$
  return $best\_S\_of\_size$

In order to analyze the results from algorithm 9, which only computes $\Phi(S)$, the expansion of a set S, not the whole graph, it makes sense to compare it with the best result possible for the same size of S. Therefore, just the line for the computation of the expansion algorithm 2 needs to be changed to get algorithm 3.

Algorithm 3 Brute-force edge expansion of sets for every size

function BruteForceEdgeExpansionSizes($H := (V, E, w)$)
  $best\_S_o f_s ize := \{\}$
  $lowest\_expansion\_of\_size := \{1 : \infty, 2 : \infty, \ldots, n-1 : \infty\}$
  for $\emptyset \neq S \subsetneq V$ **do**
    $expansion := \Phi(S)$
    if $expansion < lowest\_expansion\_of\_size[|S|]$ **then**
      $lowest\_expansion\_of\_size[|S|] := expansion$
      $best\_S\_of\_size[|S|] := S$
  return $best\_S\_of\_size$

For algorithm 2 and algorithm 3 the above argument of for exponential complexity holds as well.

3.2 Orthonormal vectors

As described in [4], an algortihm for generating a random small expansion set can be derived. For that, a set of orthonormal vectors $\{f_1, \ldots, f_k\}$ needs to be created, where each vector $f_i \in \mathbb{R}^V$ gives a value to each vertex. Because of fact 3.2.1 it will be of importance for algorithm 4 that $\xi := \max_{s \in [k]} D_w(f_s)$ is small.

Algorithm 4 Small Set Expansion (according to Algorithm 1 in [4])

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

Fact 3.2.1 *(Theorem 6.6 in [4]) Given 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 following holds: Algorithm 4 constructs a random set $S \subsetneq V$ in polynomial time such that with $\Omega(1)$ probability, $|S| \leq \frac{24|V|}{k}$ and*

$$\phi(S) \leq C \min\{\sqrt{r \log k}, k \log k \log\log k \sqrt{\log r}\} \cdot \sqrt{\xi}, \tag{3.1}$$

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

With algorithm 8 to create orthonormal vectors, algorithm algorithm 9 can be executed. This combination results in algorithm 9 according to ...

$$\begin{aligned} \underset{g}{\text{minimize}} \quad & \text{SDPval} := \sum_{e\in E} w_e \max_{u,v\in e} ||\vec{g_u} - \vec{g_v}||^2 \\ \text{subject to} \quad & \sum_{u\in V} w_v ||\vec{g_v}||^2 = 1, \\ & \sum_{u\in V} w_v f_i(v) \vec{g_v} = \vec{0}, \quad \forall i \in [k-1] \end{aligned} \tag{3.2}$$

Algorithm 5 Orthogonal Separator (combination of Lemma 18 and algorithm Theorem 10 in [5] (also Fact 6.7 in [4]))

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, β)
 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 6 Sample Assignments (proof of Lemma 18 in [5])

function SAMPLEASSIGNMENTS(l, V, β)
 $\lambda := \frac{1}{\sqrt{\beta}}$
 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 7 Rounding Algorithm for Computing Eigenvalues (Algorithm 3 in [4])

function SampleRandomVector($H, f_1, \ldots, f_{k-1}$)
  Solve SDP 3.2 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 8 Procedural Minimizer

function SampleSmallVectors(H, k)
  $f_:1 = \frac{\vec{1}}{||\vec{1}||_w}$
  for $i = 2, \ldots, k$ **do**
    $f_i :=$SampleRandomVector($H, f_1, \ldots, f_{i-1}$)
  return $f_1, \ldots, f_k$

SDP

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

Algorithm 9 Find Small Expansion Set

function SES(H)
  $f :=$SampleRandomVectors(H)
  return SmallSetExpansion(H, f)

4 Random Hypergraphs

In order to evaluate the algorithms of chapter 3 hypergraphs are require as inputs. However, instead of creating a few hypergraphs by hand, they shall be randomly generated in order to have a diverse array of graphs.

The initial intention for creating random r-uniform, d-regular, connected hypergraphs with no doubled edges in an effective manner which is guaranteed to terminate showed to be a non-trivial trivial challenge. Thetrefore, several different approaches which fulfill some of these cruteria will be discussed and their resulting graphs shall also be analyzed by their edge expansion.

Algorithm 10 Generate simple random graph

> **function** GENERATERANDOMGRAPH($n, rank, numberEdges, weightDistribution$)
> $E := \emptyset$
> $V := \{v_1, \ldots, v_n\}$
> $w = \{\}$
>   **for** $1, \ldots, numberEdges$ **do**
> $nextEdge := sample(V, rank)$ ▷ draw without replacement
> $E := E \uplus \{nextEdge\}$
> $weight[nextEdge] := sample(weightDistribution)$
>   return $H = (V, E, w)$

To start with, a simple algorithm to generate graphs which follows some of the intentions shall be discussed. Algorithm 10 is terminating, of polynomial time complexity 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 would be 0 which does make the graph possibly non-regular and also not connected. Also, the algorithm does not guarantee to have no doubled edges.

This idea can be improved by ensuring the degree of the vertices do not exceed d like in algorithm 11. This 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. An example of such a situation can be seen in fig. 4.1. Also, it does not guarantee to have no doubled edges.

Algorithm 11 Generate random graph with upper bound on degrees

function GenerateRandomGraphBoundDegrees($n, rank, d, weightDistribution$)
  $E := \emptyset$
  $V := \{v_1, \dots, v_n\}$
  $w = \{\}$
  while $|\{v \in V | deg(v) < d\}| \geq rank$ **do**
    $nextEdge := sample(\{v \in V | deg(v) < d\}, rank)$ ▷ draw without replacement
    $E := E \uplus \{nextEdge\}$
    $weight[nextEdge] := sample(weightDistribution)$
  return $H = (V, E, w)$

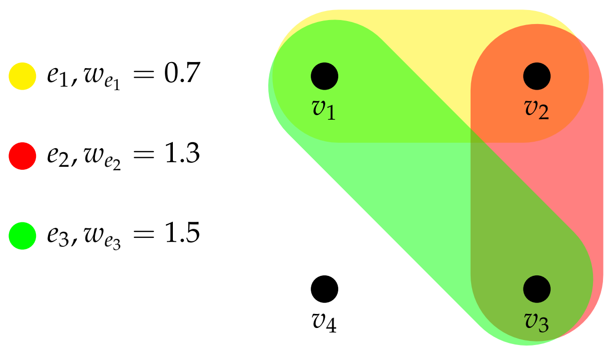

Figure 4.1: An example for a non connected 2-uniform hypergraph which could have been created by algorithm 11.

Algorithm 12 Generate random hypergraph, sampling from lowest degrees

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 $G := (V, E, w)$

To overcome these problems, the edges could only be sampled from the vertices with the smallest degrees like in algorithm 12 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 into d r-matchings. Again this graph is not neccessarily connected and some edges might be doubled.

To solve this, there are again several options: Algorithm 13 resamples the whole graph, if the .... and ... conditions are not met. This way, the algorithm loses the property of guaranteed terminating. The time complexity would depend on the probability of creating a graph which fulfills the requirements. However, this shall not be analyzed here.

This only works, if the probability for meeting the conditions are bigger than some constant, regardless of the parameters. (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 13 Generate random graph with resampling

function GenerateRandomGraph($n, rank, d, weightDistribution$)
  $G :=$ GenerateRandomGraph($n, rank, d, weightDistribution$)
  while $\not$Connected(G) or $\exists e, f \in E.e = f$ **do**
    $G :=$ GenerateRandomGraph($n, rank, d, weightDistribution$)
  return $G := (V, E)$

ii)

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.

random graph model:[6], [7] TODO: define quick

TODO: Discuss different approaches of generating, their limitations

TODO: Analyze Φ for different random- classes? (and explain?)

Algorithm 14 Generate random graph

function GenerateRandomGraph($n, rank, d, weightDistribution$)
  $G :=$ GenerateRandomGraph($n, rank, d, weightDistribution$)
  while $\not$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 $G := (V, E, w)$

Table 4.1: Comparison the properties of the graphs by the creation algorithms.

property Algorithm	11	1
d-regular	no	yes
r-uniform	yes	
no doubled edges		
connected		
terminating		
polynomial time complexity		
all possible graphs		

Algorithm 15 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 $G := (V, E, w)$

4.1 random edges with discard if not connected

4.2 connected edges with discard if not connected or not regular

5 Implementation

The discussed algorithms were implemented in order to verify the results.

5.1 Technologies

The focus of the implementation was less on performance optimization but on demonstrating feasibility. Therefore, Python 3 (todo: citation) in combination with several libraries was used. For vector representation and operations, NumPy proved useful and was therefore used. In order to optimize the SDP (todo: cite), the commonly used SciPy was chosen over tools like cvxpy or cvxopt as their implementation proved problematic due to lack of information about them as they are less known.

For storing the results of the evaluation, Pickle was used. In order to create graphs, Matplotlib, in special PyPlot was utilized.

5.2 Code

For representing hypergraphs, a(n?) own implementation was created in order to comfortably being able to implement the graph creation algorithms as well as the small expansion algoeithms. Therefore, several classes were used to represent the graph.

In order to reprecent the vertices, the class *vertex* is used. As important attributes, it contains the set of edges it belongs to and the vertice's weight w_v for a vertex v (defined like equation ...). Except for its constructor, the method *add_to_edge* is used by the construction algorithms when a new edge, containing v, is added. Additionally, for the resampling in algorithm ..., the method *recompute_weights_degrees* is needed to update the attributes of the vertex after an edge is changed.

Edges are represented by the class *edge*, which contains an attribute *weight* to represent the weight w_e of an edge e and the set of vertices.

A whole Graph H is encapsuled by the class *graph*, which contains sets of the vertices as well as edges and as needed for the graph construction algorithm ... also a set of the connection components of the graph.

Connection components are represented by the class *ConnectionComponent*, which contains the set of vertices in that component.

Furthermore, to generate small expansions, a static poisson process in positive as well as negative time is required. Therefore, the class *Poisson_Process* was created. The constructor takes λ and then calculates the times when the events happened after as well as before the time 0. With the method *get_number_events_happened_until_t*, the number of events which happened between $t_0 = 0$ and the given t is returned. For convenient handling of the vectors f generated by algorithm 8, *vertex_vector* is used to access $f(v)$.

The implementation can be found on ...

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...

As the graphs generated by the algorithms ... have regular? ranks and uniform degrees, it needs to be determined which combination of r and d should be chosen. For easier evaluation of performance depending on the number of vertices, the rank and degree were chosen to be $r = 3$ and $d = 3$. This can't be chosen freely, as For other combinations no r-uniform d-regular graphs exist on .. vertices, as $nd = mr$. This can be verified in the following way: If a "connection" is defined to be the where an edge and a vertex connect, one can count these connections from the vertices' perspective by summing up the degrees of all the vertices (which equals to $n * r$ for uniform/regular graphs). But one can also consider the count of connections from the edges perspective by summing up the ranks of the edges, which equates to $m * r$ for uniform graphs. (todo: source) As all of the variables in equation... need to be non-negative integers, one can ensure to never violate that constraint for any n by setting $d = r$.

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 ...

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

brute forcing not feasibl

todo: analyze size(number vertices) of expansions (depending on k) analyze expansion quality(number) compared to best expansion possible /average expansion through brute-force for same size estimate C

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

todo: analyze different graph generation algorithms (expansion)

analyze c with different ks, find out which side is more plausible

todo: higher k was not feasible in eq. (3.1) due to numerical issues of the implementation, therefore $|S| < \frac{24|V|}{k}$ can't be verified as k<5 here For estimating C, the inequality can be changed to

$$C \geq \frac{\phi(S)}{\min\{\sqrt{r \log k}, k \log k \log \log k \sqrt{\log r}\} \cdot \sqrt{\xi}} \tag{6.1}$$

Knowing an upper bound on C is desirable, as it would lead to small expansions.

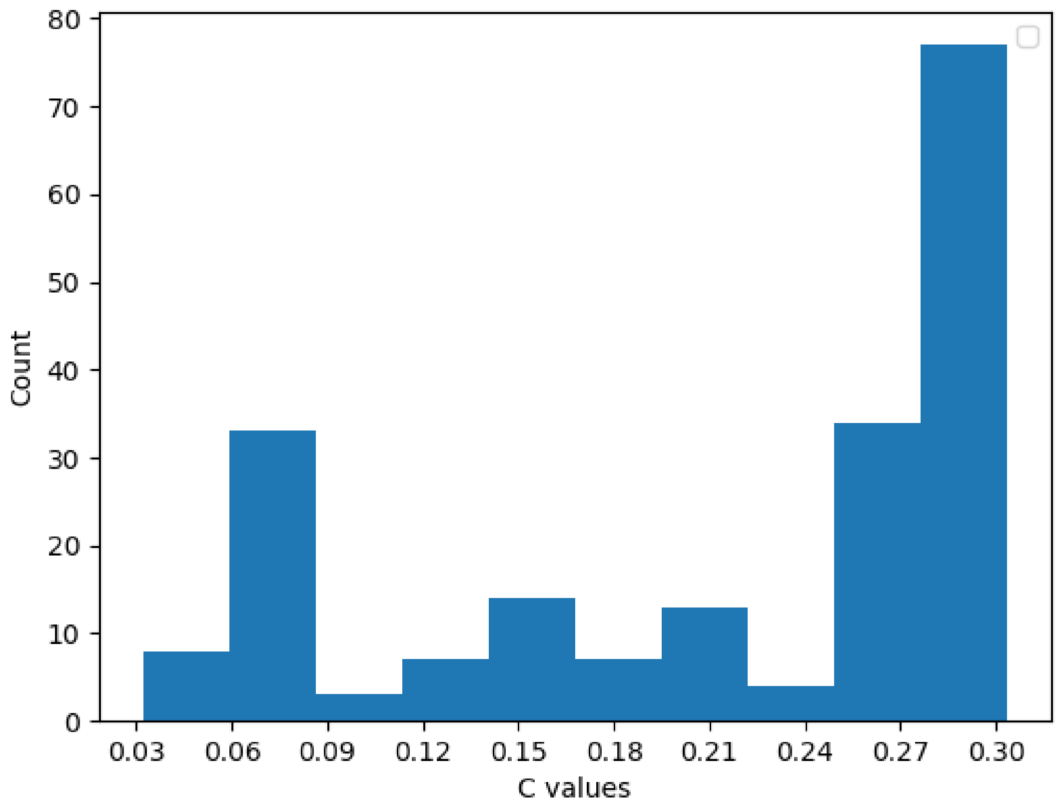

Figure 6.1: Plot of C

7 Applications

Social Networks [6] [7]

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

[1] M. Stoer and F. Wagner, "A simple min-cut algorithm," *Journal of the ACM (JACM)*, vol. 44, no. 4, pp. 585–591, 1997.

[2] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*, Springer, 1972, pp. 85–103.

[3] 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.

[4] T. H. Chan, A. Louis, Z. G. Tang, and C. Zhang, "Spectral properties of hypergraph laplacian and approximation algorithms," *CoRR*, vol. abs/1605.01483, 2016. arXiv: `1605.01483`.

[5] 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)*, K. Jansen, J. D. P. Rolim, N. R. Devanur, and C. Moore, Eds., ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 28, 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`.

[6] G. Ghoshal, V. Zlatić, G. Caldarelli, and M. E. Newman, "Random hypergraphs and their applications," *Physical Review E*, vol. 79, no. 6, p. 066 118, 2009.

[7] Z.-K. Zhang and C. Liu, "A hypergraph model of social tagging networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2010, no. 10, P10005, 2010.