Spectral Methods to Find Small Expansion Sets on Hypergraphs

Bachelor's Thesis in Informatics

Franz Rieger
Department of Informatics
Technical University of Munich
Supervisor: Prof. Dr. rer. nat. Susanne Albers
Advisor: Dr. T.-H. Hubert Chan
Submission Date: 15. March 2019

19. March 2019

Motivation

- graph theory interest
- find group of friends in social networks
- heuristics for combination games

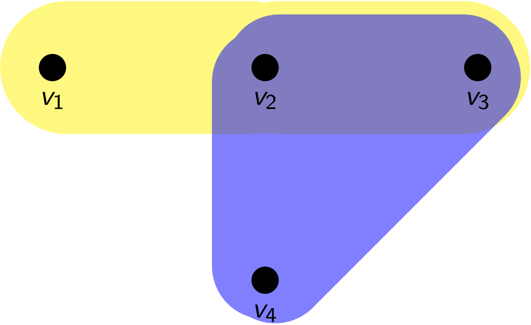
$\rightarrow$ small expansion sets

Hypergraphs

- weighted, undirected (hyper)graph $H = (V, E, w)$
- n vertices $V = \{v_1, \dots, v_n\}$
- m (hyper-)edges $E = \{e_1, \dots, e_m | \forall i \in [i] : e_i \subseteq V \wedge e_i \neq \emptyset\}$
  every edge e is non-empty subset of V
- positive edge weights $w_e := w(e)$; weight function
  $w : E \to \mathbb{R}_+$

$e_1, w_{e_1} = 0.7$

$e_2, w_{e_2} = 1.3$



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

Hypergraphs

- degree of a vertex $v \in V$: $deg(v) := |\{e \in E : v \in e\}|$
- $\forall v \in V : deg(v) = d$: hypergraph d-regular
- $\forall e \in E : |e| = r$: hypergraph r-uniform

Edge expansion

- set of edges which are cut by S : $\partial S := \{e \in E : e \cap S \neq \emptyset \wedge e \cap (V \setminus S) \neq \emptyset\}$
- weight w_v of a vertex v: $w_v := \sum_{e \in E : v \in e} w_e$
- weight $w(S)$ of a set S of vertices : $w(S) := \sum_{v \in S} w_v$
- weight $w(F)$ of a set F of edges : $w(F) := \sum_{e \in F} w_e$
- edge expansion of a set of vertices $\emptyset \neq S \subseteq V$:

$$\Phi(S) := \frac{w(\partial S)}{w(S)} \tag{1}$$

- expansion of a graph H:

$$\Phi(H) := \min_{\emptyset \subsetneq S \subsetneq V} \max\{\Phi(S), \Phi(V \setminus S)\} \tag{2}$$

Edge expansion

- non-connected graphs: $\Phi(H) = 0$

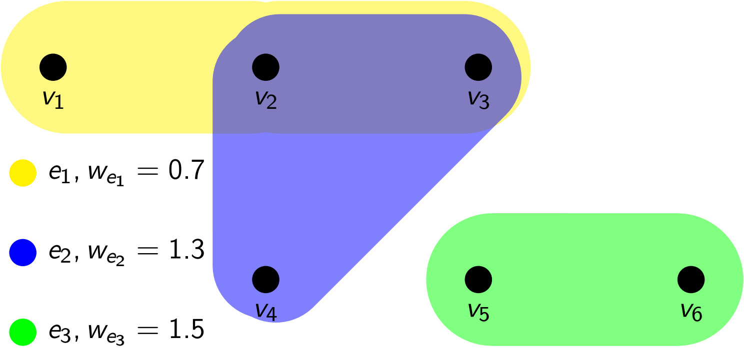


Figure: An example for a non-connected hypergraph with two connection components. For $S := \{v_5, v_6\}$ it can be verified that $\delta S = \emptyset$, hence $\Phi(S) = \Phi(V \setminus S) = 0$.

- expansion $\Phi(H)$ is NP-hard [1]

Discrepancy ratio

- discrepancy ratio, given a non-zero vector $f \in \mathbb{R}^V$:

$$D_w(f) := \frac{\sum_{e \in E} w_e \max_{u,v \in e} (f_u - f_v)^2}{\sum_{u \in V} w_u f_u^2} \tag{3}$$

- connected to the edge expansion $\Phi(S)$ of a set S if f is indicator vector for S
  - nominator $\sum_{e \in E} w_e \max_{u,v \in e} (f_u - f_v)^2$ would sum over all the edges in δS
  - denominator $\sum_{u \in V} w_u f_u^2$ would sum over the vertices S
  - $D_w(f) = \frac{w(\delta S)}{w(S)} = \Phi(S)$
- orthogonal minimaximizer : lowest discrepancy value of k mutually orthogonal non-zero vectors:

$$\xi_k := \min_{0 \neq f_1, \ldots, f_k ; f_i \perp f_j} \max_{i \in [k]} D_w(f_i) \tag{4}$$

Approximation of small expansion sets

Theorem

(Theorem 8.1 in [2]) There exists a randomized polynomial time algorithm that, given a hypergraph $H = (V, E, w)$ *and a parameter* $k < |V|$*, outputs* k *orthonormal vectors* $f_1, \ldots, f_k$ *in the weighted space such that with high probability, for each* $i \in [k]$,

$$D_w(f_i) \leq \mathcal{O}(i\xi_i \log r) \tag{5}$$

Theorem

(Theorem 6.6 in [2]) 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 9 constructs a random set $S \subsetneq V$ in polynomial time such that with $\Omega(1)$ probability, $|S| \leq \frac{24|V|}{k}$ and

$$\phi(S) \leq C \min\{\sqrt{r \log k}, k \log k \log \log k \sqrt{\log r}\} \cdot \sqrt{\xi}, \quad (6)$$

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

Deducted from [2]

Algorithm 1 Find Small Expansion Set

function SmallExpansionSet(H, k)

$f_1 \ldots, f_k$:=SampleSmallVectors(H, k)

return SmallSetExpansion(H, $f_1 \ldots, f_k$)

Brute-force graph expansion

Algorithm 2 Brute-force edge expansion on a hypergraph

---

function BruteForceEdgeExpansion($H := (V, E, w)$)
  $bestS := null$
  $lowestExpansion := \infty$
  for $\emptyset \neq S \subsetneq V$ **do**
    $expansion := \max\{\Phi(S), \Phi(V \setminus S)\}$
    if $expansion < lowestExpansion$ **then**
      $lowestExpansion := expansion$
      $bestS := S$
  return $bestS$

Runtime complexity: $2^{|V|} - 2 = 2^n - 2 \in O(2^n)$ combinations for $\emptyset \neq S \subsetneq V$

Brute-force set expansions

Algorithm 3 Brute-force expansion of sets for every size

function BruteForceEdgeExpansionSizesSets($H := (V, E, w)$)
    $\mathit{bestSofSize} := \{\}$
    $\mathit{lowestExpansionOfSize} := \{1 : \infty, 2 : \infty, \ldots, n-1 : \infty\}$
    for $\emptyset \neq S \subsetneq V$ **do**
        $\mathit{expansion} := \Phi(S)$
        if $\mathit{expansion} < \mathit{lowestExpansionOfSize}[|S|]$ **then**
            $\mathit{lowestExpansionOfSize}[|S|] := \mathit{expansion}$
            $\mathit{bestSofSize}[|S|] := S$
    return $\mathit{bestSofSize}$

Random Hypergraphs

- r-uniform
- d-regular
- unique edges guaranteed
- connected guaranteed
- guaranteed to terminate
- polynomial time complexity
- all possible graphs
- all with equal probability

Adding random edges

Algorithm 4 Generate by adding random edges

function GenerateAddRandomEdges(n, r, $numberEdges$, $weightDistribution$)
  $E := \emptyset$
  $V := \{v_1, \dots, v_n\}$
  $w = \{\}$
  for $1, \dots, numberEdges$ **do**
    $nextEdge := sample(V, r)$
    $E := E \cup \{nextEdge\}$
    $weight(nextEdge) := sample(weightDistribution)$
  return $H = (V, E, w)$

$numberEdges = \lceil \frac{nd}{r} \rceil$ according to eq. (9)

Algorithm 5 Generate random graph with resampling

```
function GenerateRandomGraph(n, r, d, weightDistribution)
    G := GenerateAddRandomEdges(n, r, nd/r, weightDistribution)
    while not connected(G) do
        G := GenerateAddRandomEdges(n, r, nd/r, weightDistribution)
    return G := (V, E)
```

Algorithm 6 Generate random graph by creating a spanning tree

```
function GenerateWithSpanningTree(n, r, d, weightDistribution)
    V := {v_1, ..., v_n}
    w = {}
    firstEdge := choice(V, r)
    w(firstEdge) = sample(weightDistribution)
    E := {firstEdge}
    while {v ∈ V | deg(v) = 0} ≠ ∅ do                                        ▷ create tree
        if |{v ∈ V | deg(v) = 0}| ≥ r − 1 then
            nextEdgeTreeVertex := choice({v ∈ V | deg(v) = 1})               ▷ get one tree node
            nextEdgeVertices :=
                choice({v ∈ V | deg(v) = 0}, r − 1) ∪ {nextEdgeTreeVertex}
        else
            nextEdgeVertices := {v ∈ V | deg(v) = 0} ∪
                choice({v ∈ V | deg(v) = 1}, r − |{v ∈ V | deg(v) = 0}|)
        nextEdge := nextEdgeVertices
        E := E ∪ {nextEdge}
        w(nextEdge) := sample(weightDistribution)
    while |{v ∈ V | deg(v) < d}| ≥ r do                                       ▷ fill up degrees
        smallestDegreeVertices := {v ∈ V | deg(v) = min_{u∈V} deg(u)}
        if |smallestDegreeVertices| >= r then
            nextEdgeVertices := sample(smallestDegreeVertices, r)
                                                                              ▷ draw without replacement
        else
            secondSmallestDegreeVertices := {v ∈ V | deg(v) = min_{u∈V} deg(u) + 1}
            nextEdgeVertices :=
                sample(secondSmallestDegreeVertices, r − |smallestDegreeVertices|)
            nextEdgeVertices := smallestDegreeVertices ∪ nextEdgeVertices
        nextEdge := nextEdgeVertices
        E := E ∪ {nextEdge}
        w(nextEdge) := sample(weightDistribution)
    return G := (V, E, w)
```

Implementation

- focus on demonstrating feasibility, not runtime performance
- Python + NumPy + SciPy
- own implementation of hypergraphs, vertices and edges
- https://github.com/riegerfr/Bachelor-s-thesis-edge-expansion/tree/master/hypergraph-implementation

Runtime input graph sizes

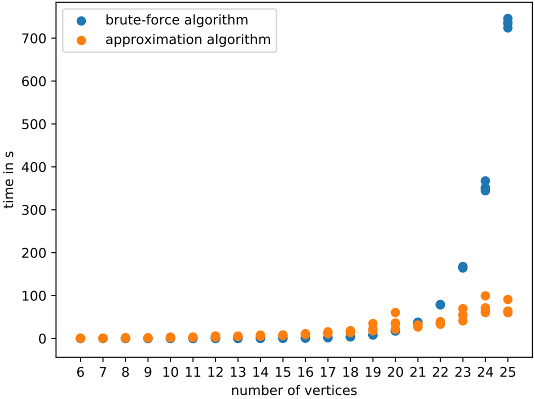


Figure: Plot of the number of vertices n in the graphs against the time for computing solutions. It can be seen that the brute-force algorithm takes a long time for larger graphs, while the approximation algorithm's time only increases slowly.

Runtime for different k

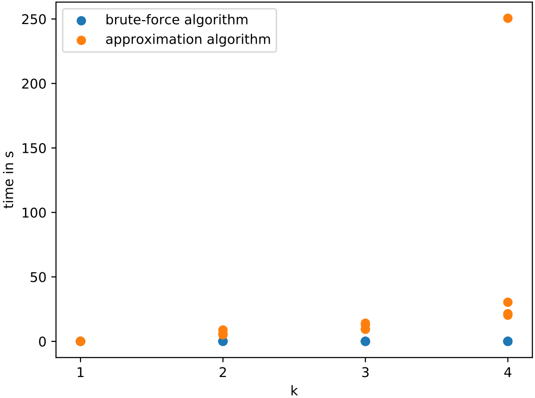


Figure: Plot of k against the runtime for the approximation algorithm with the constant brute-force time for comparison. For higher k outliers occur.

Small expansion sizes

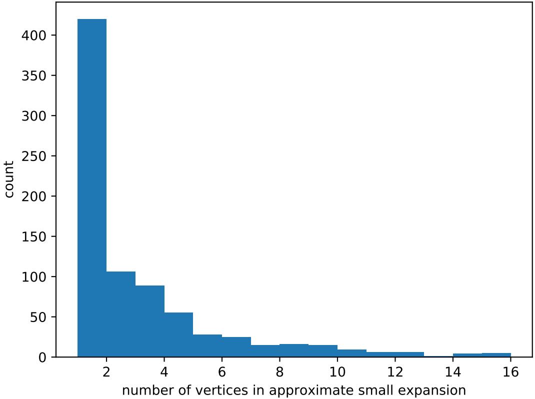


Figure: Plot of sizes of the expansions generated by the approximation algorithm.

Random graphs expansion comparison

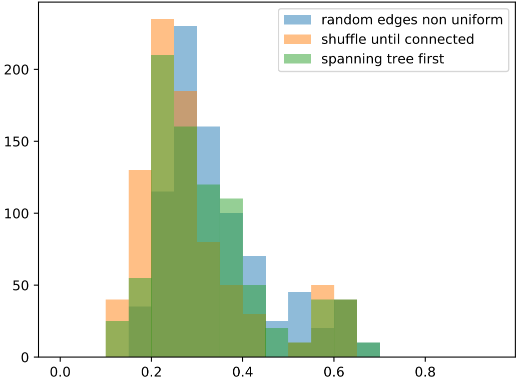


Figure: Plot of the lowest expansion values of graphs created by different algorithms for each size of the expansion set.

Estimation of C

$$C \geq \frac{\phi(S)}{\min\{\sqrt{r \log k}, k \log k \log \log k \sqrt{\log r}\} \cdot \sqrt{\xi}} \tag{7}$$

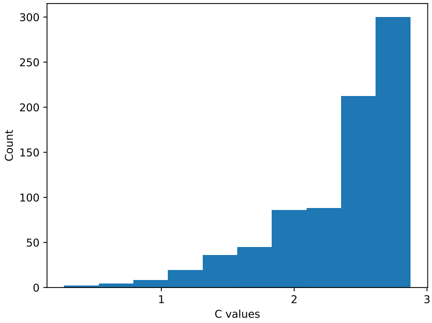


Figure: Plot of estimates for the constant C in theorem 2 for $k = 3$. Interestingly, $\max C \approx 2.71 \approx e$

Expansion comparison estimate/brute-force

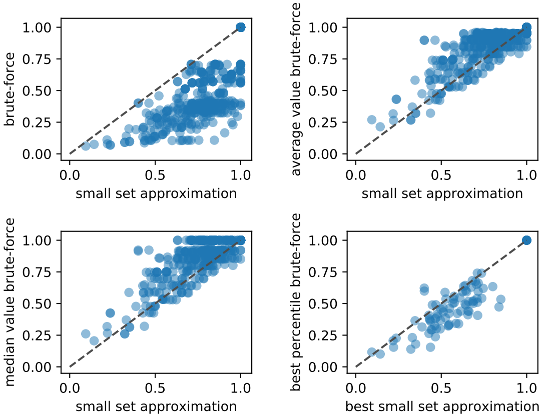


Figure: Plot of the expansion values achieved by the small expansion set approximation algorithm against the expansion set of the same size with the lowest expansion (as generated through the brute-force approach). Entries below the diagonal line signal that the expansion found by the approximation algorithm was worse than the set found by the brute force algorithm.

Applications

- Hypergraph representation of social networks: users as vertices, interactions as edges
- Small expansion approximation: group of friends
- Other application: small expansion set as heuristics in combination games

Wrap up:

- edge expansion NP-hard
- approximation algorithm for small expansion set
- random hypergraphs
- find group of friends in social networks

Thank you! Questions?

References I

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.

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.

References II

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.

Backup slides

Creation of orthogonal vectors with low discrepancy ratio

Algorithm 7 Procedural Minimizer

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

SDP

Semidefinite programming problem for minimizing g (SDP 8.3 in [2])

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

Algorithm 8 Sample Random Vector (Algorithm 3 in [2])

function SampleRandomVector($H, f_1, \ldots, f_{i-k}$)
 Solve SDP 1 to generate vectors $\vec{g_v} \in \mathbb{R}^n$ for $v \in V$
 $\vec{z} := sample(\mathcal{N}(0, I_n))$ ▷ random gaussian vector
 for $v \in V$ **do**
  $f(v) := \langle \vec{g_v}, \vec{z} \rangle$
 return f

Calculating a small expansion set

Algorithm 9 Small Set Expansion (according to Algorithm 1 in [2])

function SmallSetExpansion($G := (V, E, w), f_1, \ldots, f_k$)
  for $v \in V, s \in [k]$ **do**
    $u_v(s) := f_s(v)$
  for $v \in V$ **do**
    $\tilde{u}_v := \frac{u_v}{||u_v||}$
  $\hat{S} :=$ OrthogonalSeparator($\{\tilde{u}_v\}_{v \in V}, \beta = \frac{99}{100}, \tau = k$)
  for $v \in V$ **do**
    if $\tilde{u}_v \in \hat{S}$ **then**
      $X_v := ||u_v||^2$
    else
      $X_v := 0$
  $X :=$ sort $list(\{X_v\}_{v \in V})$
  $V := [v]_{\text{in order of X}}$
  $S := \arg\min_{\{P \text{ is prefix of } V\}} \phi(P)$
  return S

Algorithm 10 Orthogonal Separator (combination of Lemma 18 and algorithm of Theorem 10 in [3]; also Fact 6.7 in [2])

---

function OrthogonalSeparator($\{\tilde{u}_v\}_{v \in V}, \beta = \frac{99}{100}, \tau = k$)
  $l := \lceil \frac{\log_2 k}{1 - \log_2(1 + \frac{2}{\log_2 k})} \rceil$
  $w :=$SampleAssignments$(\{\tilde{u}_v\}_{v \in V}, l, V, \beta)$
  for $v \in V$ **do**
    $W(v) := w_1(v) w_2(v) \cdots w_l(v)$
  if $n \geq 2^l$ **then**
    $word := random(\{0,1\}^l)$ ▷ uniform
  else
    $words := set(W(v) : v \in V)$ ▷ no multiset
    $words = words \uplus \{w_1, \ldots, w_{|V| - |words|} \in \{0,1\}^l\}$
    $word := random(words)$ ▷ uniform
  $r := uniform(0,1)$
  $S := \{v \in V : ||\tilde{u}_v||^2 \geq r \wedge W(u) = word\}$
  return S

Algorithm 11 Sample Assignments (proof of Lemma 18 in [3])

function SampleAssignments($\{\tilde{u}_v\}_{v \in V}, l, V, \beta$)

  $\lambda := \frac{1}{\sqrt{\beta}}$

  $k := |\tilde{u}|$ ▷ number of entries for each vertex

  $g :=$ sample($\mathcal{N}(0, I_k)$) ▷ all components g_i mutually indep.

  poisson_process $:= N(\lambda)$ ▷ Poisson process on $\mathbb{R}$ w. rate λ

  for $i = 1, 2, \ldots, l$ **do**

    for $v \in V$ **do**

      $t := \langle g, \tilde{u}_v \rangle$

      poisson_count $:=$ *poisson_process*(t)

        ▷ # events between $t = 0$ and t_v

      if *poisson_count* mod $2 == 0$ **then**

        $w_i(v) := 1$

      else

        $w_i(v) := 0$

  return w

Notation

- vectors $f, g \in \mathbb{R}^V$ the inner product is defined as $\langle f, g \rangle_w := f^T W g$.
- If $\langle f, g \rangle_w = 0$, f and g are said to be orthogonal
- norm $||f||_w := \sqrt{\langle f, f \rangle_w}$
- weight matrix W of a hypergraph

$$W := \begin{pmatrix} w_{v_1} & 0 & 0 & \dots & 0 \\ 0 & w_{v_2} & 0 & \dots & 0 \\ 0 & 0 & w_{v_3} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & w_{v_n} \end{pmatrix} \in \mathbb{R}_{0+}^{n \times n} \tag{8}$$

- $0 \leq D_w(f) \leq 2$ [2]
- $0 \leq \Phi(H) \leq 1$
- d-regular, r-uniform hypergraphs (handshaking theorem):

$$nd = mr \tag{9}$$

- path between two vertices $v_1, v_k \in V$: list of vertices $v_1, v_2 \ldots, v_k$ where each tuple of vertices following another is connected by an edge, i.e. $\forall i \in [k-1] \exists e \in E : u_i, u_{i+1} \in e$
- connected component : $S \subseteq V . \forall u, v \in S . \exists path(u, v)$
- $S = V$: hypergraph connected

Runtime rank/degree-combinations

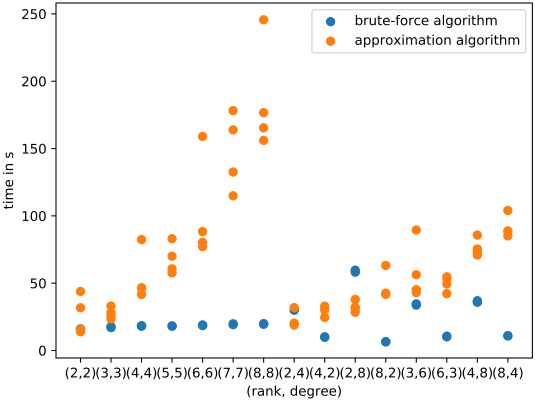


Figure: Plot of runtimes for different (rank, degree) combinations.

Algorithm 12 Generate random hypergraph, sampling from lowest degrees

```
function GenerateSampleSmallestDegrees(n, r, d, weightDistribution)
    E := ∅
    V := {v_1, ..., v_n}
    while |{v ∈ V | deg(v) < d}| ≥ r do
        smallestDegreeVertices := {v ∈ V | deg(v) = min_{u∈V} deg(u)}
        if |smallestDegreeVertices| >= r then
            nextEdgeVertices := sample(smallestDegreeVertices, r)
        else
            secondSmallestDegreeVertices := {v ∈ V | deg(v) = min_{u∈V} deg(u) + 1}
            nextEdgeVertices :=
                sample(secondSmallestDegreeVertices, r − |smallestDegreeVertices|)
            nextEdgeVertices := smallestDegreeVertices ∪ nextEdgeVertices
        nextEdgeWeight := sample(weightDistribution)
        nextEdge := nextEdgeVertices
        E := E ∪ {nextEdge}
        w(e) := nextEdgeWeight
    return G := (V, E, w)
```

Algorithm 13 Generate by randomly swapping edges

function GenerateSwapEdges($n, r, d, weightDistribution$)
  $G :=$ GenerateSampleSmallestDegrees($n, r, d, weightDistribution$)
  while not connected(G) **do**
    $e, f := sample(E, 2)$
    $u := sample(e)$
    $v := sample(f)$
    if $connected\_component(u) \neq connected\_component(v)$ **then**
      $e := (e \cup \{v\}) \setminus \{u\}$
      $f := (f \cup \{u\}) \setminus \{v\}$
  return $G := (V, E, w)$