



DEPARTMENT OF COMPUTER SCIENCE

# Exponential-time Algorithms for Approximating the Hardcore Partition Function



A dissertation submitted to the University of Bristol in accordance with the requirements of the degree  
of Master of Engineering in the Faculty of Engineering.

Thursday 4<sup>th</sup> May, 2023


---

# Abstract

In a graph, an independent set is a subset of the vertices such that no edges are run between vertices included in the set. We look at counting independent sets when each vertex has weights associated with it depending on whether it is included in an independent set. The partition function takes a graph as input, and sums over the weights of each independent set. In the hardcore model, we give each vertex a weight of  $\lambda$  for being in the independent set and 1 for being out. This paper aims to generalise an algorithm of Goldberg, Lapinskas and Richerby for approximating the partition function on graphs when  $\lambda = 1$  to any  $\lambda > 0$ . For the base case, we find families of graphs for which the partition function can be approximated in polynomial time. We analyse the behaviour of the main algorithm by finding functions that capture the complexity of inputs.

# Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

 Thursday 4<sup>th</sup> May, 2023

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>FPTAS for Sparse Graphs</b>	<b>3</b>
2.1	FPTAS for the hardcore model . . . . .	3
2.2	Reduction to the hardcore model . . . . .	4
2.3	The approximation algorithm . . . . .	5
2.4	Proof of correctness . . . . .	5
<b>3</b>	<b>Families of Graphs with Subcritical Connective Constant</b>	<b>8</b>
<b>4</b>	<b>Branching on arbitrary graphs</b>	<b>12</b>
4.1	Associated average degree . . . . .	12
4.2	Preprocessing algorithms . . . . .	13
4.3	Graph decompositions . . . . .	17
4.4	The approximation algorithm . . . . .	18
4.5	Proof of correctness . . . . .	18
4.6	Correctness of potential functions . . . . .	20
<b>5</b>	<b>Pre-potentials for values of Lambda</b>	<b>26</b>
<b>6</b>	<b>Conclusion</b>	<b>28</b>
<b>A</b>	<b>Generating code for decreasing functions</b>	<b>30</b>
<b>B</b>	<b>Verification for decreasing functions</b>	<b>33</b>
<b>C</b>	<b>Verification for potential functions</b>	<b>35</b>
<b>D</b>	<b>Pre-potential functions</b>	<b>37</b>

---

# List of Figures

2.1	Graph $G$ and the realisation of $\phi$ on $G$ . . . . .	4
4.1	An example of the standard decomposition of a graph. . . . .	17
4.2	An example of the extended decomposition of a graph. . . . .	18

---

# List of Tables

3.1	Critical values for $\lambda$ with associated subcritical connective constant, degree and 2-degree bounds . . . . .	10
5.1	Critical values for $\lambda$ with associated running time constant $\sigma$ for general graphs and $\sigma_b$ for bipartite graphs. . . . .	27
D.1	A pre-potential function on general graphs for $\lambda = 2$ . . . . .	37
D.2	A pre-potential function on bipartite graphs for $\lambda = 2$ . . . . .	38

---

# Ethics Statement

This project did not require ethical review, as determined by my supervisor, Dr. John Lapinskas.

---

# Supporting Technologies

- I used Python and Sage-Math to find decreasing functions in Chapter 3 and potential functions in Chapter 5. Some of this code is included in Appendices A,B and C.



---

# Notation and Acronyms

$\lambda$	: Parameter of the hardcore model.
FPTAS	: Fully Polynomial Time Approximation Scheme.
$\Gamma_G(v)$	: Neighbourhood of vertex $v$ in graph $G$ .
$\Gamma_G^2(v)$	: 2-Neighbourhood of vertex $v$ in graph $G$ .
$d_G(v)$	: Degree of vertex $v$ in graph $G$ .
$d_G^2(v)$	: 2-degree of vertex $v$ in graph $G$ .
$\delta(G)$	: Minimum degree of a graph $G$ .
$\Delta(G)$	: Maximum degree of a graph $G$ .
$\mathcal{I}(G)$	: Set of Independent Sets of graph $G$ .
$Z(\mathcal{G})$	: Partition function for weighted graph $\mathcal{G}$ .
$Z_\lambda(G)$	: Univariate hardcore partition function of graph $G$ with parameter $\lambda$ .
$\mathcal{D}_{a,b}$	: Family of graphs such that for each $G \in \mathcal{D}_{a,b}$ , if $v \in V$ has $d_G(v) \geq a$ , then $d_G^2(v) \leq b$ .
$\omega(\lambda)$	: Highest valid degree bound for $\lambda$ .
$\omega_2(\lambda)$	: Highest valid 2-degree bound for $\lambda$ .
$T_{SAW}(v, G)$	: Self-Avoiding Walk Tree of $G$ rooted at vertex $v$ .
$N_G(v, \ell)$	: Number of vertices at depth $\ell$ in $T_{SAW}(v, G)$ .

---

# Chapter 1

## Introduction

In a graph, an independent set is a subset of the vertices such that there are no edges running between any of the vertices included in the set. We denote the set of independent sets in a graph  $G$  with  $\mathcal{I}(G)$ .  $\#IS$  is the problem of counting the number of independent sets in a graph, and  $\#BIS$  is the same for a bipartite graph. We can generalise these problems by introducing weights to the vertices for being in or out of an independent set. The hardcore model is the problem when each vertex has weight  $\lambda > 0$  when it is in the independent set, and 1 otherwise. Then the *hardcore partition function* defines the weight of independent sets in the hardcore model.

**Definition 1** (Hardcore partition function). *For  $\lambda > 0$ , and graph  $G$ , the hardcore partition function is the function  $Z_\lambda$  where*

$$Z_\lambda(G) = \sum_{I \in \mathcal{I}(G)} \lambda^{|I|}.$$

In this paper, we generalise the algorithm of Goldberg, Lapinskas and Richerby [2] for approximating  $Z_\lambda(G)$  from the case when  $\lambda = 1$  to  $\lambda > 0$ . Along the way, we will want to change the weights on vertices so that they are not all equal. For this, we define *weighted graphs*.

**Definition 2** (Weighted Graph). *A weighted graph  $\mathcal{G}$  is a tuple  $(G, w_+, w_-, W)$  where  $G = (V, E)$  is a graph,  $w_+$  and  $w_-$  are functions from the vertices of  $G$  to positive real numbers, and  $W$  is a positive integer. We say that  $G$  is the underlying graph of  $\mathcal{G}$ , and we call  $w_+(v)$  the in-weight of  $v$ , and  $w_-(v)$  the out-weight of  $v$ .*

Then the weight of independent sets in a weighted graph is given as follows.

**Definition 3** (Weighted Hardcore Partition Function). *Let  $\mathcal{G} = (G, w_+, w_-, W)$  be a weighted graph. The weighted hardcore partition function is the function defined by*

$$Z(\mathcal{G}) = W \sum_{I \in \mathcal{I}(G)} \prod_{v \in I} w_+(v) \prod_{v \in V(G) \setminus I} w_-(v).$$

We note that  $Z((G, \lambda, \mathbf{1}, 1)) = Z_\lambda(G)$ , where  $\lambda$  is the constant function at  $\lambda$ , and  $\mathbf{1}$  is the constant function at 1.

Rather than finding these values exactly, we want to approximate them. An *error tolerance* defines how close we need to be to the true value.

**Definition 4** (Error Tolerance). *An error tolerance is a value  $\varepsilon \in \mathbb{Q}$  such that  $0 < \varepsilon < 1$ .*

Then an  $\varepsilon$ -approximation defines the range around the true value we must be in.

**Definition 5** ( $\varepsilon$ -approximation). *An  $\varepsilon$ -approximation to a value  $P$  is a value  $N$  such that  $(1 - \varepsilon)N \leq P \leq (1 + \varepsilon)N$ .*

For the base case of the algorithm, we will use a family of algorithms which approximate  $Z_\lambda(G)$  in time polynomial in the input size, and the error tolerance.

**Definition 6** (FPTAS). *A fully polynomial time approximation scheme to a value  $P$  is a family of algorithms  $\mathcal{A}$  with parameter  $\varepsilon$  such that  $\mathcal{A}_\varepsilon$  is an  $\varepsilon$ -approximation to  $P$ , and each  $\mathcal{A}_\varepsilon$  runs in time polynomial in the input size, and  $1/\varepsilon$ .*

---

In Chapter 2, we look at an FPTAS for the hardcore model, and reduce the problem of finding the weight of independent sets in a  $\lambda$ -balanced weighted graph to the hardcore model.

In Chapter 3, we look at families of graphs that we can find the weight of independent sets in time polynomial in the input size.

In Chapter 4, we look at the main approximation algorithm, and prove that we can use functions on the input to analyse the running time.

In Chapter 5, we find potential functions for  $\lambda > 0$  and therefore bound the running time of the algorithm.

---

## Chapter 2

# FPTAS for Sparse Graphs

In this chapter we deal with the ‘base case’ of the exponential time algorithm. This is when the graph is part of a family of graphs that are sparse enough that  $Z_\lambda(G)$  can be approximated in time polynomial in the size of the input.

### 2.1 FPTAS for the hardcore model

We first define the *self-avoiding walk tree* starting at a vertex in a graph.

**Definition 7** (Self-Avoiding Walk Tree). *Let  $G$  be a graph and  $v \in G$  a vertex of  $G$ . The self-avoiding walk tree of  $G$  rooted at  $v$  is denoted  $T_{SAW}(v, G)$ . The vertices of  $T_{SAW}(v, G)$  are the self-avoiding walks/simple paths starting from vertex  $v$ . The children of any vertex  $P$  are exactly the simple paths that extend  $P$  by one vertex. The root is the length-0 path  $v$ .  $N_G(v, \ell)$  denotes the number of vertices at distance  $\ell$  from the root of  $T_{SAW}(v, G)$ .*

The reason we consider the self-avoiding walk tree is that if we can put a bound on the number of leaves of the tree, we can ensure that any graph from the family is sparse enough that there exists a polynomial time algorithm for  $Z_\lambda(G)$  for any  $G \in \mathcal{F}$ . To bound the number of leaves, we bound the *connective constant* of the family.

**Definition 8** (Connective Constant). [3] *The connective constant of a family of graphs  $\mathcal{F}$  is at most  $\kappa$  if there exist real numbers  $a$  and  $c$  such that for every  $G \in \mathcal{F}$ , and every vertex  $v \in G$ , for every integer  $\ell \geq a \log(|V(G)|)$ , we have  $\sum_{i=1}^{\ell} N_G(v, i) \leq c\kappa^\ell$ . The connective constant of  $\mathcal{F}$  is the infimum of all such  $\kappa$ .*

For an example, consider the family of graphs  $\mathcal{F}$  to be the family of all binary trees. Take a graph  $G \in \mathcal{F}$ . When the self-avoiding walk tree is constructed from any vertex  $v \in G$ , the root can have 3 children, but every other vertex can have at most 2 children, as the maximum degree of a vertex in  $G$  is 3. So for all  $i \geq 1$ , we have that  $N_G(v, i) \leq 3 \cdot 2^{\ell-1}$ . Therefore,

$$\sum_{i=1}^{\ell} N_G(v, i) \leq 3 \cdot (2^0 + 2^1 + \dots + 2^{\ell-1}) \leq 3 \cdot 2^\ell.$$

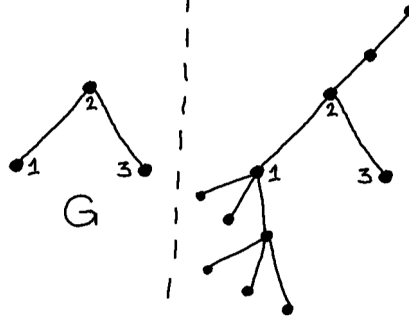
Thus, by choosing  $a = 1$  and  $c = 3$ , we find that the connective constant of  $\mathcal{F}$  is at most 2. A similar analysis can be given to show that the family of  $n$ -regular graphs for  $n \geq 3$  has connective constant at most  $n - 1$ . The connective constant enables us to bound the number of leaves in the self-avoiding walk tree, which constrains the complexity of any graph in the family.

There is a useful bound on the connective constant that will guarantee that an algorithm for  $Z_\lambda(G)$  exists. This bound is where the connective constant  $\kappa$  is called *subcritical* with respect to constant  $\lambda > 0$ .

**Definition 9** (Subcritical connective constant). *The connective constant  $\kappa \geq 1$  of a family of graphs  $\mathcal{F}$  is subcritical with respect to  $\lambda > 0$  if*

$$\lambda < \frac{\kappa^\kappa}{(\kappa - 1)^{\kappa+1}}.$$

By a result of Sinclair et al.,


 Figure 2.1: Graph  $G$  and the realisation of  $\phi$  on  $G$ .

**Theorem 10.** [4] For a family of graphs  $\mathcal{F}$  with subcritical connective constant with respect to  $\lambda > 0$ , there is an FPTAS  $\text{ApproxZUni}_{\lambda, \mathcal{F}}$  for  $Z_\lambda(G)$  for  $G \in \mathcal{F}$ .

However, we cannot assume that the input graph will be an instance of the hardcore model. We need a more general algorithm for computing  $Z(\mathcal{G})$  on  $\lambda$ -balanced graphs.

**Definition 11** ( $\lambda$ -balance). A weighted graph  $\mathcal{G} = (G, w_+, w_-, W)$  is  $\lambda$ -balanced for some  $\lambda > 0$  if for each  $v \in V(G)$ ,  $w_+(v) \leq \lambda w_-(v)$ .

We now define a reduction from  $\lambda$ -balanced graphs from a family  $\mathcal{F}$  with subcritical connective constant with respect to  $\lambda > 0$  to the hardcore model on  $\lambda$ .

## 2.2 Reduction to the hardcore model

This is the reduction detailed by Goldberg, Lapinskas and Richerby [2]. The reduction works by taking the input weighted graph  $\mathcal{G}$  and simulates the current weights on the vertices by attaching depth-2 trees to each vertex. This is achieved by defining *weight maps* for each vertex in  $G$ .

**Definition 12** (Weight Map). A weight map on a graph  $G$  is a map  $\phi$  from the vertices of  $G$  to (possibly empty) finite multisets of non-negative integers.

The *realisation* of a weight map is the graph formed by for each  $v \in V(G)$  and each  $n \in \phi(v)$ , adding a star on  $n$  vertices to the graph, and connecting its centre to  $v$ .

**Definition 13** (Realisation of a weight map). The realisation of a weight map  $\phi$  defined on a graph  $G$  is the graph formed by starting with a copy of  $G$ . Then for each vertex  $v \in G$ , and each  $n \in \phi(v)$ , form a star with  $n$  leaves with new vertices. The centre of each star is then joined to  $v$  with an edge.

An example of this construction is shown in Figure 2.1, on the graph  $G = \{\{1, 2, 3\}, \{\{1, 2\}, \{2, 3\}\}$  with weight map  $\phi$  where  $\phi(1) = \{0, 0, 3\}$ ,  $\phi(2) = \{1\}$  and  $\phi(3) = \emptyset$ .

However to use this reduction, we have to make sure that any graph that can be generated as a realisation of a weight map still comes from a family with subcritical connective constant.

**Definition 14.** Let  $\mathcal{F}$  be a family of graphs and  $y \in \mathbb{Z}_{>0}$ . Then for any graph  $G = (V, E)$ , let  $\Phi_y(G)$  be the set of all weight maps  $\phi$  on  $G$ , such that for all  $v \in V$ ,  $\sum_{i \in \phi(v)} (i + 1) \leq y|V|^2$ . Then define  $\mathcal{F}_y^+$  to be the set of all realisations of weight maps in  $\bigcup_{G \in \mathcal{F}} \Phi_y(G)$

Then,

**Lemma 15.** Let  $\kappa \in \mathbb{R}_{\geq 1}$ ,  $\lambda \in \mathbb{R}_{>0}$ , and  $y \in \mathbb{Z}_{>0}$ . Let  $\mathcal{F}$  be a family of graphs with connective constant  $\kappa$  subcritical with respect to  $\lambda$ . Then  $\mathcal{F}_y^+$  also has subcritical connective constant with respect to  $\lambda$ .

*Sketch Proof.* The proof uses the fact that any self-avoiding walk in a realisation of a weight map can only have at most two edges at the start and two at the end of the walk in the added edges. Since  $\phi \in \Phi_y(G)$ , there are only at most  $yn^2$  choices for each step in the walk outside  $G$ . This limits the number of self-avoiding walks from any vertex.

## 2.3 The approximation algorithm

Now we can define the FPTAS  $\text{ApproxZ}_{\lambda, \mathcal{F}}(\mathcal{G}, \varepsilon)$  for  $Z(\mathcal{G})$ . For all  $\lambda > 0$ , we write  $\Lambda_t = 1 + \lambda(1 + \lambda)^{-t}$  for any  $t \in \mathbb{Z}_{>0}$ . Note that for all  $\lambda, t > 0$ , we have  $1 < \Lambda_{t+1} < \Lambda_t$ .

---

**Algorithm 1**  $\text{ApproxZ}_{\lambda, \mathcal{F}}(\mathcal{G}, \varepsilon)$ 


---

**Parameters:** Value  $\lambda > 0$  and family of graphs  $\mathcal{F}$  that has subcritical connective constant with respect to  $\lambda$ .

**Inputs:**  $\lambda$ -balanced weighted graph  $\mathcal{G} = (G, w_+, w_-, W)$  such that  $G \in \mathcal{F}$ , and error tolerance  $\varepsilon \in \mathbb{Q}$  with  $0 < \varepsilon < 1$ .

- (i) If  $\varepsilon \leq 3n\lambda(\max\{2, 1 + \lambda\})^{-n}$ , calculate  $Z(\mathcal{G})$  exactly by brute force and return it.
  - (ii) Form a weighted graph  $\mathcal{G}' = (G', w_+, w_-, W')$  as follows: for each  $v \in G$  with  $w_+(v) \leq \frac{\varepsilon}{3n}w_-(v)$ , delete  $v$  from  $G$  and multiply  $W$  by  $w_-(v)$ . Let  $G'$  be the graph after this process. Let  $S = V(G) \setminus V(G')$ , and  $W' = Ww_-(S)$ .
  - (iii) For each  $v \in V(G')$ , calculate  $a_{0,v}, \dots, a_{n,v}$  as follows. First set  $x_v \leftarrow \lambda \frac{w_-(v)}{w_+(v)}$ . Then for  $t = 0$  to  $n$ , let  $a_{t,v} = \lfloor \log_{\Lambda_t} x_v \rfloor$ , and update  $x_v \leftarrow x_v / \Lambda_t^{a_{t,v}}$ .
  - (iv) Define weight map  $\phi$  on  $G'$  by mapping each  $v \in V(G')$  to the multiset of exactly  $a_{t,v}$  copies of  $t$  for each  $t$  from 0 to  $n$ . Let  $G''$  be the realisation of  $\phi$  on  $G'$ .
  - (v) Return  $\text{ApproxZUni}_{\lambda, \mathcal{F}_{\lfloor 5(1+\lambda) \rfloor}}(G'', \varepsilon/3) \cdot W' \lambda^{-n} \prod_{v \in V(G')} (w_+(v) / \prod_{i=0}^n (1 + \lambda)^{ia_{i,v}})$ .
- 

First, in step (i), if  $\varepsilon \leq 3n\lambda(\max\{2, 1 + \lambda\})^{-n}$ , note that computing  $Z(\mathcal{G})$  exactly takes time  $\text{poly}(2^n)$ , as  $2^n$  is an upper bound on the number of independent sets in  $G$ . In this case this is also  $\text{poly}(1/\varepsilon)$ , so this is fully polynomial. We take  $\max\{2, 1 + \lambda\}$  for the case where  $\lambda < 1$ , as otherwise  $\text{poly}(2^n)$  is not  $\text{poly}(1/\varepsilon)$ .

In step (ii), we can delete the vertices  $v \in V(G)$  with  $w_+(v) \leq \frac{\varepsilon}{3n}w_-(v)$ , as the in-weight of the vertex is small enough compared to the out-weight that we can simply remove it and multiply by  $w_-(v)$ , and by Lemma 16, still get an  $\varepsilon$ -approximation.

Then the aim of steps (iii) and (iv) is to simulate the out-weights  $w_-(v)$  of vertices  $v \in G'$ . It does this by attaching stars on  $0 \leq t \leq n$  leaves. In the hardcore model, when  $v$  is included in an independent set, we cannot include the centre of the star, so the total weight of independent sets in the star is  $(1 + \lambda)^t$ . If  $v$  is not included in the independent set, we can include independent set in the star consisting of only the centre, so the total weight is  $\lambda + (1 + \lambda)^t$ . This places slightly more weight on when  $v$  is not included in the independent set. As  $t$  increases, the difference get relatively smaller, so we have more precision on the weight we can put on  $v$ .

Lastly, in step (v), we can find the value of  $Z_\lambda(G'')$ , and adjust based on the number and size of stars we have added.

## 2.4 Proof of correctness

First, clearly when  $\varepsilon \leq 3n\lambda \max\{2, 1 + \lambda\}^{-n}$ ,  $\text{ApproxZ}_{\lambda, \mathcal{F}}(\mathcal{G}, \varepsilon)$  returns  $Z(\mathcal{G})$ , which is an  $\varepsilon$ -approximation of  $Z(\mathcal{G})$  no matter the value of  $\varepsilon$ . Otherwise, when  $\varepsilon > 3n\lambda(1 + \lambda)^{-n}$ , we build correctness in three steps.

- In step (ii),  $Z(\mathcal{G})$  is approximately  $Z(\mathcal{G}')$ .
- In step (iv),  $G'' \in \mathcal{F}_{\lfloor 5(1+\lambda) \rfloor}$  and  $\mathcal{F}_{\lfloor 5(1+\lambda) \rfloor}$  has subcritical connective constant with respect to  $\lambda$ , so  $\text{ApproxZUni}_{\lambda, \mathcal{F}_{\lfloor 5(1+\lambda) \rfloor}}(G'', \varepsilon/3)$  is approximately  $Z_\lambda(G'')$ .
- $Z_\lambda(G'')$  is approximately  $Z(\mathcal{G}')/C$  where

$$C = W' \lambda^{-n} \prod_{v \in V(G')} \left( w_+(v) / \prod_{i=0}^n (1 + \lambda)^{ia_{i,v}} \right)$$

Then putting this all together, we find that as  $\text{ApproxZ}_{\lambda, \mathcal{F}}$  outputs  $Z_\lambda(G'') \cdot C$ , it does indeed approximate  $Z(\mathcal{G})$ .

**Lemma 16.** Let  $\lambda > 0$  and  $\mathcal{F}$  be a family of graphs with subcritical connective constant with respect to  $\lambda$ . Let  $\mathcal{G} = (G, w_+, w_-, W)$  be a non-empty  $\lambda$ -balanced weighted graph with  $G \in \mathcal{F}$ . Take  $\varepsilon \in \mathbb{Q}$  with  $3n\lambda(1+\lambda)^{-n} < \varepsilon < 1$ . Then in step (ii) of **ApproxZ $_{\lambda, \mathcal{F}}$** , we have  $(1 - \varepsilon/3)Z(\mathcal{G}) \leq Z(\mathcal{G}') \leq Z(\mathcal{G})$ .

*Sketch Proof.* Take  $\mathcal{G}'$  as defined in **ApproxZ $_{\lambda, \mathcal{F}}$** ( $\mathcal{G}, \varepsilon$ ). Let  $S = V(G) \setminus V(G')$ , so  $W' = Ww_-(S)$ . So  $Z(\mathcal{G}')$  is exactly  $Z(\mathcal{G})$  restricted to independent sets in  $G'$ . So clearly  $Z(\mathcal{G}') \leq Z(\mathcal{G})$ . Then as all vertices  $v$  with  $w_+(v) \leq \frac{\varepsilon}{3n}w_-(v)$  were removed in step 1, we find that  $w_-(S) \geq (1 - \varepsilon/3) \sum_{S' \subseteq S} w_+(S')w_-(S \setminus S')$ . By combining independent sets by their parts in  $S$  and  $S'$ , we find that  $Z(\mathcal{G}') \geq (1 - \varepsilon/3)Z(\mathcal{G})$  as required.

**Lemma 17.** Let  $\lambda > 0$  and let  $\mathcal{F}$  be a family of graphs with subcritical connective constant with respect to  $\lambda$ . For every non-empty  $\lambda$ -balanced weighted graph  $\mathcal{G} = (G, w_+, w_-, W)$  with  $G \in \mathcal{F}$ , and every  $\varepsilon \in \mathbb{Q}$  with  $3n\lambda(1+\lambda)^{-n} < \varepsilon < 1$ , after step (iv) of **ApproxZ $_{\lambda, \mathcal{F}}$** ( $\mathcal{G}, \varepsilon$ ):

$$(i) \text{ for each } v \in V(G'), (1 - \frac{\varepsilon}{3n}) \frac{\lambda w_-(v)}{w_+(v)} \leq \prod_{t=0}^n \Lambda_t^{a_{t,v}} \leq \frac{\lambda w_-(v)}{w_+(v)}.$$

$$(ii) G'' \in \mathcal{F}_{\lfloor 5(1+\lambda) \rfloor}^+.$$

*Sketch Proof.* Pick a  $v \in V(G')$ . Then for each  $0 \leq t \leq n+1$ , set

$$x_{t,v} = \frac{\lambda w_-(v)}{w_+(v)} \prod_{i=0}^{t-1} \Lambda_i^{-a_{i,v}} \quad (2.1)$$

Each  $x_{i,v}$  holds the value of  $x_v$  at the end of the  $i-1$ th iteration of step (iii). Then by the definition of  $a_{t,v} = \lfloor \log_{\Lambda_t} x_{t,v} \rfloor$ , we find that  $1 \leq \frac{\lambda w_-(v)}{w_+(v)} \prod_{i=0}^n \Lambda_i^{-a_{i,v}} \leq \Lambda_n$ , so the upper bound follows.

To show that  $G'' \in \mathcal{F}_{\lfloor 5(1+\lambda) \rfloor}^+$ , we need to show

$$\sum_{t \in \phi(v)} (t+1) = \sum_{t=0}^n a_{t,v}(t+1) \leq \lfloor 5(1+\lambda) \rfloor n^2.$$

We do this by bounding each  $a_{t,v}$  from above. We note that for each  $1 \leq t \leq n+1$ , that  $1 \leq x_{t,v} < \Lambda_{t-1}$ . Using some rules about logarithms, we find for all  $t \geq 1$

$$a_{t,v} = \lfloor \log_{\Lambda_t} x_v \rfloor \leq \lfloor (\log \Lambda_{t-1}) / \log \Lambda_t \rfloor \leq \lfloor 2(1+\lambda) \rfloor.$$

Then for  $t = 0$ , we note that from step (ii), we must have that  $w_+(v)/w_-(v) > \varepsilon/3n > \lambda(1+\lambda)^{-n}$ . So  $a_{0,v} = \lfloor \log_{1+\lambda} \lambda w_-(v)/w_+(v) \rfloor \leq n$ . Therefore,

$$\sum_{t \in \phi(v)} (t+1) = \sum_{t=0}^n a_{t,v}(t+1) \leq n + \lfloor 2(1+\lambda) \rfloor \sum_{i=1}^n (i+1) \leq \lfloor 5(1+\lambda) \rfloor n^2.$$

So  $G'' \in \mathcal{F}_{\lfloor 5(1+\lambda) \rfloor}^+$ .

**Lemma 18.** Let  $\lambda > 0$  and let  $\mathcal{F}$  be a family of graphs with subcritical connective constant with respect to  $\lambda$ . For every non-empty  $\lambda$ -balanced weighted graph  $\mathcal{G} = (G, w_+, w_-, W)$  with  $G \in \mathcal{F}$ , and every  $\varepsilon \in \mathbb{Q}$  with  $3n\lambda(1+\lambda)^{-n} < \varepsilon < 1$ . In step (iv) of **ApproxZ $_{\lambda, \mathcal{F}}$** ( $\mathcal{G}, \varepsilon$ ),

$$(1 - \varepsilon/3)Z(\mathcal{G}') \leq Z_\lambda(G'')W'\lambda^{-n} \prod_{v \in V(G')} (w_+(v) / \prod_{i=0}^n (1+\lambda)^{ia_{i,v}}) \leq Z(\mathcal{G}').$$

*Sketch Proof.* Let  $\mathcal{G}'$  and  $G''$  be defined as in **ApproxZ $_{\lambda, \mathcal{F}}$** . Let

$$M = Z_\lambda(G'')W'\lambda^{-n} \prod_{v \in V(G')} (w_+(v) / \prod_{i=0}^n (1+\lambda)^{ia_{i,v}}).$$

For the attached stars in  $G''$ , let  $T$  be a star with  $t$  leaves. Then  $Z_\lambda(T) = \lambda + (1+\lambda)^t$ . The weight of independent sets in  $T$  not including  $T$ 's centre is  $(1+\lambda)^t$ . So

$$Z_\lambda(G'') = \sum_{I \in \mathcal{I}(G)} \lambda^{|I|} \left( \prod_{v \in I} \prod_{i=0}^n (1+\lambda)^{ia_{v,i}} \right) \left( \prod_{v \in V(G') \setminus I} \prod_{i=0}^n (\lambda + (1+\lambda)^i)^{a_{v,i}} \right).$$

The first product considers stars where the vertex of  $G'$  the star is connected to is in  $I$ , so only independent sets not including the centre are counted. The second product does the same but including the centre. Then by some algebraic manipulation, we find that

$$M = W' \lambda^{-n} \sum_{I \in \mathcal{I}(G)} \lambda^{|I|} \left( \prod_{v \in I} w_+(v) \right) \left( \prod_{v \in V(G') \setminus I} w_+(v) \prod_{i=0}^n \Lambda_i^{a_{i,v}} \right).$$

By Lemma 17, this implies that  $M \leq Z(\mathcal{G}')$ . The same argument holds for the lower bound.

**Theorem 19.** *Let  $\lambda > 0$  and  $\mathcal{F}$  be a family of graphs with subcritical connective constant with respect to  $\lambda$ . Then there is an FPTAS for  $Z(\mathcal{G})$  where  $\mathcal{G} = (G, w_+, w_-, W)$  is a  $\lambda$ -balanced weighted graph with  $G \in \mathcal{F}$ .*

*Proof.* We first show that  $\text{ApproxZ}_{\lambda, \mathcal{F}}$  is a FPTAS. Let  $\varepsilon \in (0, 1)$  and  $n := V(G)$ . If  $\varepsilon \leq 3n\lambda(\max\{2, 1 + \lambda\})^{-n}$ , then the runtime is that of the brute force algorithm, which is  $\text{poly}(2^n) = \text{poly}(1/\varepsilon)$ . Otherwise, if  $\varepsilon > 3n\lambda(\max\{2, 1 + \lambda\})^{-n}$ , each of steps 1-4 take polynomial time in  $n$  and  $1/\varepsilon$ . By Lemma 17,  $\mathcal{F}_{\lfloor 5(1+\lambda) \rfloor}^+$  has subcritical connective constant with respect to  $\lambda$ , so  $\text{ApproxZUni}_{\lambda, \mathcal{F}_{\lfloor 5(1+\lambda) \rfloor}^+}$  takes time polynomial in  $n$  and  $1/\varepsilon$ . So  $\text{ApproxZ}_{\lambda, \mathcal{F}}$  takes time polynomial in  $n$  and  $1/\varepsilon$  in all cases.

Considering the correctness of the algorithm, combining Lemma 16 and Lemma 18 gives

$$(1 - \varepsilon/3)^2 Z(\mathcal{G}) \leq Z_\lambda(G'') W' \lambda^{-n} \prod_{v \in V(G')} (w_+(v) / \prod_{i=0}^n (1 + \lambda)^{ia_{i,v}}) \leq Z(\mathcal{G}).$$

Then, by Lemma 17 and the correctness of  $\text{ApproxZUni}_{\lambda, \mathcal{F}_{\lfloor 5(1+\lambda) \rfloor}^+}(G'', \varepsilon/3)$ , we find that the output of  $\text{ApproxZ}_{\lambda, \mathcal{F}}(\mathcal{G}, \varepsilon)$  lies between  $(1 - \varepsilon/3)^3 Z(\mathcal{G})$  and  $(1 + \varepsilon/3) Z(\mathcal{G})$ . So  $\text{ApproxZ}_{\lambda, \mathcal{F}}$  does indeed approximate  $Z(\mathcal{G})$ .  $\square$



---

## Chapter 3

# Families of Graphs with Subcritical Connective Constant

In this chapter, we will find families of graphs such that there exists an FPTAS for  $Z(\mathcal{G})$  where the underlying graph comes from the family of graphs. This will allow the main algorithm to take a  $\lambda$ -balanced weighted graph  $\mathcal{G} = (G, w_+, w_-, W)$  and either approximate  $Z(\mathcal{G})$  in time polynomial in the size of the graph, or exploit properties of the graph. To do this, we must show that the family of graphs has subcritical connective constant with respect to  $\lambda$ . This requires that we bound the number of leaves in the self-avoiding walk tree  $T_{SAW}(v, G)$  for any  $G$  in the family, rooted at any vertex  $v \in G$ . Again, we follow the analysis in [2] for the first half of this chapter, then generalise to find families for all  $\lambda > 0$ .

To help with the analysis of these trees, we introduce some notation. Let  $T = (V, E, r)$  be a tree  $(V, E)$  rooted at vertex  $r \in V$ .  $L(T)$  denotes the leaves of  $T$ . For any  $v \in V$ ,  $C(v)$  denotes the set of  $v$ 's children, vertices  $w$  that are adjacent to  $v$  but not on the unique path from  $r$  to  $v$ .  $D(v)$  denotes  $v$ 's depth, the length of the unique path from  $r$  to  $v$ .

We introduce the notion of a  $\kappa$ -decreasing function that for rooted tree  $T = (V, E, r)$ , assigns real weights to each vertex  $v \in V$  such that the sum of weights on the children of any vertex  $w \in V \setminus r$  is at most  $\kappa$  times the weight on  $w$ .

**Definition 20** ( $\kappa$ -decreasing functions). *Let  $T = (V, E, r)$  be a rooted tree and  $\kappa \geq 1$ . A function  $\theta : V \rightarrow \mathbb{R}$  is  $\kappa$ -decreasing on  $T$  if for each vertex  $x \in V$ ,  $1/\kappa \leq \theta(x) \leq 1$ , and for every  $x \in V \setminus r$ ,*

$$\sum_{y \in C(x)} \theta(y) \leq \kappa \theta(x).$$

The lower bound  $\theta(x) \geq 1/\kappa$  is arbitrary, any positive lower bound on the values of  $\theta$  suffices. However, it is of use with the proof of the next lemma.

**Lemma 21.** *Let  $G$  be a graph and let  $\kappa > 1$ . Suppose that for every  $v \in G$ , there exists a  $\kappa$ -decreasing function on  $T_{SAW}(v, G)$ . Then  $\Delta(G) \leq \kappa^2 + 1$ .*

*Proof.* Suppose for a contradiction, that there is some vertex  $u \in V(G)$  such that  $d_G(u) > \kappa^2 + 1$ . As  $\kappa > 1$ ,  $d_G(u) > 1$ , so  $u$  has a neighbour  $v$ . Let  $\theta$  be a  $\kappa$ -decreasing function on  $T_{SAW}(v, G)$ . Consider the vertex  $x$  corresponding to the walk  $vu$  in  $G$ . For each adjacent vertex  $w \in \Gamma_G(u) \setminus v$ ,  $x$  has a child corresponding to the walk  $vuw$ , so  $x$  has more than  $\kappa^2$  children. However, by the definition of  $\kappa$ -decreasing functions, we have that  $\sum_{y \in C(x)} \theta(y) < \kappa \frac{1}{\kappa} = 1 \leq \kappa \theta(x)$ . This contradicts the definition of  $\theta$ , so it must be the case that  $d_G(u) \leq \kappa^2 + 1$ .  $\square$

This bound on  $\Delta(G)$  helps with the main lemma of this chapter, which uses the existence of  $\kappa$ -decreasing functions to put an upper bound on the connective constant on a family of graphs.

**Lemma 22.** *Let  $\mathcal{F}$  be a family of graphs. Let  $\kappa \geq 1$  and suppose for every  $G \in \mathcal{F}$ , there exists a  $\kappa$ -decreasing function on  $T_{SAW}(v, G)$ . Then the connective constant of  $\mathcal{F}$  is at most  $\kappa$ .*

*Proof.* We suppose that for each  $G \in \mathcal{F}$  and each  $v \in G$ , there exists  $\theta$ , a  $\kappa$ -decreasing function on  $T := T_{SAW}(v, G)$ . We look at the subtrees  $T_i$  consisting of vertices of distance at most  $i \geq 0$  from the root  $r$ , and define  $S_i := \sum_{w \in L(T_i)} \theta(w) \kappa^{-D(w)}$ . We make an inductive argument that the size of  $S_i$  is at

most  $2\kappa$  for all  $i \geq 0$ . First, for the base cases, we have that  $S_0 = \theta(r) \leq 1 \leq 2\kappa$ . Then for  $i = 1$ , by Lemma 21,  $d_G(v) \leq \kappa^2 + 1$ . So  $r$  has at most  $\kappa^2 + 1$ , thus

$$S_1 = \sum_{w \in C(r)} \theta(w) \kappa^{-1} \leq \sum_{w \in C(r)} \kappa^{-1} \leq \frac{\kappa^2 + 1}{\kappa} \leq \kappa + 1 < 2\kappa.$$

Then, for the inductive argument, suppose for  $i \geq 1$  that  $S_i < 2\kappa$ . Let  $w$  be a leaf in  $T_i$ . If  $w$  is a leaf in  $T$ , then it contributes as much to  $S_{i+1}$  as it does to  $S_i$ . Otherwise, all of  $w$ 's children are leaves in  $T_{i+1}$ . Their contribution is

$$\sum_{y \in C(w)} \theta(y) \kappa^{-D(y)} = \kappa^{-D(w)+1} \sum_{y \in C(w)} \theta(y) \leq \kappa^{-D(w)+1} \kappa \theta(w) = \kappa^{-D(w)} \theta(w).$$

Thus  $w$ 's children contribute exactly as much to  $S_{i+1}$  as they do to  $S_i$ . So  $S_{i+1} = S_i < 2\kappa$ . Since the leaves of  $T_i$  include all the vertices with depth  $i$ , we have  $N(v, i) \leq |L(T_i)|$ . Therefore,

$$N(v, i) \leq |L(T_i)| \leq \kappa \sum_{w \in L(T_i)} \theta(w) \leq \kappa^{i+1} \sum_{w \in L(T_i)} \theta(w) \kappa^{-D(w)} = \kappa^{i+1} S_i < 2\kappa^{i+2}.$$

Therefore for all  $\ell \geq 1$ ,

$$\sum_{i=1}^{\ell} N(v, i) < \sum_{i=1}^{\ell} 2\kappa^{i+2} < \frac{2\kappa^3}{\kappa - 1} \kappa^{\ell}.$$

In particular, this holds for all  $\ell \geq \log |V(G)|$ , so by choosing  $a = 1$ ,  $c = 2\kappa^3/(\kappa - 1)$ , the connective constant of  $\mathcal{F}$  is at most  $\kappa$ .  $\square$

Therefore, to find a family of graphs with subcritical connective constant with respect to  $\lambda > 0$ , we can instead find  $\kappa$ -decreasing functions for the self-avoiding walk trees on graphs in the family for  $\kappa$  subcritical with respect to  $\lambda$ . We find these functions for a family of graphs with bounded 2-degree.

**Definition 23** (2-degree). *Let  $G$  be a graph. The 2-degree,  $d_G^2(v)$ , of a vertex  $v \in V(G)$  is the sum of its neighbour's degrees.*

$$d_G^2(v) = \sum_{w \in \Gamma_G(v)} d(w).$$

We now define the family of graphs that respect the 2-degree bound.

**Definition 24.** *For values  $a, b \in \mathbb{Z}$ , let  $\mathcal{D}_{a,b}$  be the family of graphs such that for all  $G \in \mathcal{D}_{a,b}$ ,  $\delta(G) \geq 2$ , and all vertices  $v \in G$ , if  $d_G(v) \geq a$ , then  $d_G^2(v) \leq b$ .*

So we only need to consider neighbour's degrees up to  $\lfloor b/2 \rfloor$  for 2-degree bound  $b$ .

We now describe a method for finding the values of the degree bounds given  $\lambda > 0$ . Set degree bound  $a$  and 2-degree bound  $b$ , and let  $\mathcal{D}_{a,b}$  be defined as above. Let  $T := T_{SAW}(v, G)$  with root  $r$  for some  $G \in \mathcal{D}_{a,b}$  and some  $v \in V(G)$ . We try to find a function  $\psi : \mathbb{Z}_{>0}^2 \rightarrow \mathbb{R}$  that can be adapted into a  $\kappa$ -decreasing function. From  $\psi$  we define  $\theta : V(T) \rightarrow \mathbb{R}$  by setting  $\theta(r) = 1$ , and for  $v \in V(T) \setminus r$  with parent  $w \in V(T)$ , setting  $\theta(v) = \psi(d_G(v), d_G(w))$ .

The program in Appendix A uses the linear programming methods in SageMath to attempt to find a function  $\psi$ . It iterates over the possible degrees of the neighbours of a vertex of a graph in  $\mathcal{D}_{a,b}$ . Note that there is an upper bound on the degree of a vertex in  $\mathcal{D}_{a,b}$ .

**Lemma 25.** *The maximum degree of a vertex  $v$  in graph  $G$  with  $\delta(G) \geq 2$  where for any vertex with  $d_G(v) \geq a$ , we have  $d_G^2(v) \leq b$  is at most  $\lfloor b/2 \rfloor$ .*

*Proof.* Suppose vertex  $v \in G$  has degree  $d_G(v) > \lfloor b/2 \rfloor$ . Then  $d_G^2(v) \geq 2d_G(v) > 2(b/2) = b$ . Thus the 2-degree bound would be violated.  $\square$

So we only need to consider neighbour's degrees up to  $\lfloor b/2 \rfloor$  for 2-degree bound  $b$ . We let  $u$  be the parent of  $v$ . Then, we add the following constraint.

$$\sum_{w \in C(v)} \psi(d(w), d(v)) \leq \kappa \psi(d(v), d(u)).$$

Once all the constraints have been added, it attempts to solve the linear program. If a solution exists, then it can be adapted into a  $\kappa$ -decreasing function  $\theta$  on  $T$ .

- 
- Approximate the critical value of the connective constant  $\kappa$ , and let  $b$  be an estimate for a valid 2-degree bound.
  - Iterate over the possible degrees of the neighbours of a vertex  $v$ . Add a constraint to the linear program based on the possible values.
  - Try to solve the linear program. If it is unsolvable, decrease the 2-degree bound by 1 and try again, otherwise increase the bound by 1 and try again.
  - When the highest valid 2-degree bound has been found, simplify the decreasing function by forcing as many values for the degree bound to be equal, until no solution remains. Return the simplest function.

For any  $\lambda > 0$ , let  $\omega_2(\lambda)$  be the highest 2-degree bound and  $\omega(\lambda)$  be the associated degree bound such that for each  $G \in \mathcal{D}_{\omega(\lambda), \omega_2(\lambda)}$  we can compute  $Z_\lambda(G)$  in time polynomial in the input. We note that there is a lower bound for  $\omega_2(\lambda)$  for any  $\lambda > 0$

**Lemma 26.** *Let  $\lambda > 0$ , and take  $\mathcal{D}_{2,4}$ , the family of graphs consisting of components that are all cycles. Then for any  $\lambda$ -balanced weighted graph  $\mathcal{G} = (G, w_+, w_-, W)$  with  $G \in \mathcal{D}_{2,4}$ , we can compute  $Z(\mathcal{G})$  in time  $\text{poly}(|V(G)|)$ .*

*Proof.* Take  $\lambda > 0$ , and let weighted graph  $\mathcal{G}$  have underlying graph  $G \in \mathcal{D}_{2,4}$ .  $\mathcal{G}$  consists of a number of components, all of which are cycles. For each component  $C$ , we can branch on any vertex and find that both the subproblems are trees. We can compute the weight of these independent sets in time  $\text{poly}(|V(G)|)$ , and so can compute  $Z(C)$  in time  $\text{poly}(|V(G)|)$ . Then, as there are at most  $|V(G)|$  components, we can therefore compute  $Z(\mathcal{G})$  in time  $\text{poly}(|V(G)|)$ .  $\square$

Therefore, we can be sure that for any  $\lambda > 0$ ,  $\omega(\lambda) \geq 2$  and  $\omega_2(\lambda) \geq 4$ .

A sample of the critical values of  $\lambda$  with the subcritical connective constant, associated degree and 2-degree bounds is shown in Table 3.1.

$\lambda$	Connective Constant	Degree Bound	2-Degree Bound
0.988	4.175	6	26
1.024	4.076	6	25
1.054	4.000	5	24
1.107	3.872	5	23
1.160	3.757	5	22
1.219	3.641	5	21
1.285	3.523	5	20
1.321	3.463	5	19
1.403	3.339	5	18
1.497	3.213	5	17
1.606	3.085	5	16
1.688	3.000	4	15
1.878	2.829	4	14
2.097	2.669	4	13
2.341	2.526	4	12
2.494	2.450	4	11
2.934	2.273	4	10
3.564	2.093	4	9
4.000	2.000	3	8

Table 3.1: Critical values for  $\lambda$  with associated subcritical connective constant, degree and 2-degree bounds

This brings us to the main theorem of this chapter.

**Theorem 27.** *For all  $\lambda > 0$ , let  $a$  be the degree bound and  $b$  be the 2-degree bound output by the program in Appendix A. Then there exists an FPTAS **BaseCount** $(\mathcal{G}, \varepsilon)$  on  $\lambda$ -balanced weighted graphs  $\mathcal{G} = (G, w_+, w_-, W)$  where  $G \in \mathcal{D}_{a,b}$ .*

---

*Proof.* Let  $\lambda > 0$ . Then let  $a$  and  $b$  be the output from Appendix A as described above, and let  $\psi : \mathbb{Z}_{>0}^2 \rightarrow \mathbb{R}$  be the output function. The program in Appendix B takes connective constant  $\kappa$ , degree bounds  $a$  and  $b$ , and function  $\psi$  as input, and verifies that the associated function  $\theta : V(T) \rightarrow \mathbb{R}$  is a valid  $\kappa$ -decreasing function on self-avoiding walk trees  $T$  of graphs in  $\mathcal{D}_{a,b}$ . For every possible set of degree of the neighbours of a vertex  $v$ , it verifies that  $1/\kappa \leq \theta(v) \leq 1$ , and that the inequality

$$\sum_{w \in C(v)} \theta(w) \leq \kappa \theta(v)$$

holds. If it does, then  $\theta$  is a valid  $\kappa$ -decreasing function for graphs  $G \in \mathcal{D}_{a,b}$ , so by Lemma 22, the connective constant of  $\mathcal{D}_{a,b}$  is at most  $\kappa$ . Thus for any  $\lambda > 0$  such that  $\kappa$  is subcritical with respect to  $\lambda$ ,  $\text{ApproxZ}_{\lambda, \mathcal{D}_{a,b}}(\mathcal{G}, \varepsilon)$  is an FPTAS for  $Z(\mathcal{G})$  for any  $\lambda$ -balanced weighted graph  $\mathcal{G} = (G, w_+, w_-, W)$  where  $G \in \mathcal{D}_{a,b}$ .  $\square$

---

## Chapter 4

# Branching on arbitrary graphs

In this section, we describe the main algorithm for approximating  $Z_\lambda(G)$  for  $\lambda > 0$  on arbitrary and bipartite graphs  $G$ , and analyse their running time. In this section, we use  $m$  as the number of edges in a graph  $G$ , and  $n$  for the number of vertices. This chapter follows the argument of Goldberg, Lapinskas and Richerby [2], but generalised from  $\lambda = 1$  to  $\lambda > 0$  where necessary.

### 4.1 Associated average degree

First, we use the notion of *associated average degree* to put bounds on the 2-degrees of vertices in the graph from the average degree of the graph.

**Definition 28** (Associated Average Degree). [1] *Let  $G$  be a graph with average degree  $k$ . For each  $v \in V(G)$ , we define*

$$\alpha(v) = d_G(v) + |\{w \in \Gamma_G(v) \mid d_G(w) < k\}|,$$

$$\beta(v) = 1 + \sum_{w \in \Gamma_G(v), d_G(w) < k} \frac{1}{d_G(w)}.$$

The associated average degree of vertex  $v$  is given by  $\text{aad}(v) = \alpha(v)/\beta(v)$ .

It is also useful to define associated average degree in terms of numerical functions.

**Definition 29.** Let  $d \in \mathbb{Z}_{>0}$  and  $k \in \mathbb{R}_{>0}$ , and  $d \geq k$ . For all vectors  $\mathbf{x} \in \mathbb{Z}_{>0}^d$ , we define

$$a_k(\mathbf{x}) = d + |\{i \mid x_i < k\}|,$$

$$b_k(\mathbf{x}) = 1 + \sum_{i: x_i < k} \frac{1}{x_i},$$

$$\text{aad}_k^*(\mathbf{x}) = a_k(\mathbf{x})/b_k(\mathbf{x}).$$

The definitions correspond by considering  $\mathbf{x}$  as the list of the degrees of the neighbours of a vertex. So the degree of the vertex is the dimension of  $\mathbf{x}$ , and the 2-degree is the sum of  $\mathbf{x}$ . Then, by the following lemma, we use the average degree of a graph to prove the existence of a vertex with high associated average degree. This lemma is from [1], but translated to the language of #IS.

**Lemma 30.** [1] *Let  $G = (V, E)$  be a graph with average degree  $k$ . Then there is some vertex  $v \in V$  with  $d_G(v) \geq k$  and  $\text{aad}(v) \geq k$ .*

*Sketch Proof.* Let  $X = \{v \in V \mid d_G(v) \geq k\}$ . There is at least one vertex of at least average degree, so  $X \neq \emptyset$ . We let  $A = \sum_{v \in X} \alpha(v)$  and  $B = \sum_{v \in X} \beta(v)$ . Then for  $v \in V(G)$ , let  $d_{\geq k}(v) = |\{w \in \Gamma_G(v) \mid d_G(w) \geq k\}|$ , and let

$$f(v) = \begin{cases} d_{\geq k}(v) & \text{if } d(v) < k \\ d_G(v) & \text{otherwise.} \end{cases}$$

Then for each  $v \in V$ , if  $d_G(v) < k$ , then  $f(v)$  counts the number of adjacent vertices with degree at least  $k$ . Otherwise,  $f(v)$  is just the degree of  $v$ . So  $A = \sum_{v \in V} f(v)$ . Similarly,  $B = \sum_{v \in V} f(v)/d_G(v)$ . Write  $n_i$  for the number of degree- $i$  vertices in  $G$ . Then let  $n'_i$  be  $n_i$  for  $i \geq k$  and  $|\{v \in V | d(v) < k \text{ \& } d_{\geq k}(v) = i\}|$ . Then for  $S = \sum_{1 \leq i < k} (n_i - n'_i)$ , we find

$$A = \sum_{i \geq 1} in'_i = \sum_{i \geq 1} in_i - \sum_{1 \leq i < k} i(n_i - n'_i) \geq 2|E| - kS = k(|V| - S).$$

Similarly, we find that  $B \leq |V| - S$ , so  $A \geq kB$ . Therefore there must be some vertex  $v$  with  $\alpha(v) \geq k\beta(v)$ , so  $\text{aad}(v) \geq k$ .

Now that we have proved the existence of a vertex with high associated average degree, we use this to put a lower bound on the highest 2-degree of a vertex. We define the following function on the 2-degrees of vertices.

**Definition 31.** For any real number  $k \geq 2$ , let  $K = \lfloor k \rfloor + 1$ . Then let  $D_2(k)$  be the smallest integer  $z$  such that there is an integer  $d \geq K$  and tuple  $\mathbf{x} = (x_1, \dots, x_d) \in \mathbb{Z}_{\geq 2}^d$  with  $\sum_{i=1}^d x_i = z$  and  $\text{aad}_K^*(\mathbf{x}) > k$ .

Since  $\sum_{i=1}^d x_i$  corresponds to the 2-degree of a vertex,  $D_2(k)$  corresponds to the smallest 2-degree such that there exists a set of neighbour degrees such that the vertex has  $\text{aad}$  at least  $k$ . We now use this definition to lower bound 2-degree from the average degree of a graph.

**Lemma 32.** For any real number  $k \geq 2$ , let  $K = \lfloor k \rfloor + 1$ . Every graph  $G$  with minimum degree 2 and average degree in  $(k, K]$  contains a vertex with degree at least  $K$  and 2-degree at least  $D_2(k)$ .

*Sketch Proof.* Let  $A > k$  be the average degree of  $G$ . By Lemma 30, there is some vertex  $v \in G$  with  $d_G(v) \geq A$  and  $\text{aad} \geq A$ . As  $d_G(v)$  is an integer, we must have  $d_G(v) \geq K$ . Then let  $\mathbf{x} = (x_1, \dots, x_{d_G(v)})$  be a tuple of the degrees of the neighbours of  $v$ . Then  $\text{aad}(v) = \text{aad}_A^*(\mathbf{x})$ . As the elements of  $\mathbf{x}$  are integers,  $\text{aad}_A^*(\mathbf{x}) = \text{aad}_K^*(\mathbf{x})$ . But then by Definition 31, we have that  $d_G^2(v) = \sum_{i \leq d_G(v)} x_i \geq D_2(k)$ .

## 4.2 Preprocessing algorithms

To analyse the running time of the algorithm, we use a potential function that measures the complexity of the input graph. If we can solve  $Z(\mathcal{G})$  in time polynomial in  $|V(G)|$ , the potential function should return 0. We also require that when we branch on vertices, the resulting subproblems should have potential significantly less than that of the original graph. To do this, we use a potential function that is *piecewise-linear*.

**Definition 33** (Piecewise-linear). A function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is piecewise-linear if there exists a function  $g : \mathbb{R}^n \rightarrow \mathbb{R}$ , a series of boundary points  $k_0 < k_1 < \dots < k_n$  and a series of linear functions  $f_0, f_1, \dots, f_{n+1}$  such that if  $k_i < g(x_1, \dots, x_n) \leq k_{i+1}$ , then  $f(x_1, \dots, x_n) = f_{i+1}(x_1, \dots, x_n)$ .

For our potential functions, we take  $m$  and  $n$  as inputs. We set  $g = 2m/n$ , the average degree of the vertices in  $G$ . Furthermore, we require the linear functions  $f_0, \dots, f_n$  to be *good slices*.

**Definition 34** (Good slice). A function  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  is a good slice if it is of the form  $f(m, n) = \rho m + \sigma n$  for real numbers  $\rho, \sigma$  with  $\rho \geq 0$  and  $\rho \geq -\sigma$ , but if  $\rho = 0$  then  $\sigma \neq 0$ . Given a good slice  $f$ , we also use  $f$  to denote the function from graphs to  $\mathbb{R}$  given by  $f(G) = f(|E(G)|, |V(G)|)$ . We also use it for the function from weighted graphs  $\mathcal{G} = (G, w_+, w_-, W)$  to  $\mathbb{R}$  given by  $f(\mathcal{G}) = f(G)$ .

The potential function that we use will, in fact, have a slice with  $\sigma = -\rho$ . This is because we will describe a method that can remove vertices of degree 0 or 1 from the graph. Then, if the average degree of the graph after these removals is 2, the graph must be 2-regular. A 2-regular graph consists of one or more components, all of which are cycles. The weight of independent sets in cycles can be computed in polynomial time, so we can set  $f(G) = 0$  when  $|E(G)| = |V(G)|$ . So for our preprocessing, we would like to efficiently remove features of the graph. However, we must be careful. For the potential functions to be useful, when we branch on a vertex of  $\mathcal{G}$  and create recursive calls on  $\mathcal{G}_{out}$  and  $\mathcal{G}_{in}$ , we require that for any good slice  $f$ ,  $f(\mathcal{G}_{out}) < f(\mathcal{G})$  and  $f(\mathcal{G}_{in}) < f(\mathcal{G})$ . However, if we have a tree component  $T$  of  $G$  that we remove by naively remove by repeatedly pruning degree-1 vertices, we can have

$$f(G - V(T)) = f(G) - \rho|E(T)| - \sigma|V(T)| = f(G) - \rho(|V(T)| - 1) - \sigma|V(T)| = f(G) + \rho > f(G).$$

So we have to keep track of the tree components we create when we branch, to ensure that when we branch it does not increase the potential. First of all, we would like to remove subsets of the vertices of the input graph that are only adjacent to at most one other vertex. For this, we define the **Prune** operation.

**Definition 35** (Prune). For  $\mathcal{G} = (G, w_+, w_-, W)$  weighted graph, let  $S \subseteq V(G)$  be such that  $|\Gamma_G(S)| \leq 1$ . Then define  $\text{Prune}(\mathcal{G}, S)$  as follows:

- If  $\Gamma_G(S) = \emptyset$ , then  $\text{Prune}(\mathcal{G}, S) = (G - S, w_+, w_-, Z((G[S], w_+, w_-, W)))$ .
- If  $\Gamma_G(S) = \{v\}$  for some  $v \in V(G)$ , then let  $S' = S \cap \Gamma_G(S)$ , and  $S'' = S \setminus \Gamma_G(S)$ . Then  $\text{Prune}(\mathcal{G}, S) = (G - S, w'_+, w'_-, W)$  where:

$$w'_+(x) = \begin{cases} w_+(v)w_-(S')Z((G[S''], w_+, w_-, 1)) & \text{if } x = v \\ w_+(x) & \text{if } x \neq v, \end{cases}$$

$$w'_-(x) = \begin{cases} w_-(v)Z((G[S], w_+, w_-, 1)) & \text{if } x = v \\ w_-(x) & \text{if } x \neq v. \end{cases}$$

For an explanation, suppose that  $|S| \subseteq V(G)$  with  $|\Gamma_G(S)| \leq 1$ . If  $\Gamma_G(S) = \emptyset$ , then we can treat it separately, and multiply by the weight of its independent sets. Otherwise, if  $\Gamma_G(S) = \{v\}$ , then we must consider whether  $v$  is in any independent set. If it is, then we cannot include any vertices in  $S$  adjacent to  $v$ . So we multiply the in-weight of  $v$  by the out-weight of  $S$ , and the weight of independent sets in the remainder of  $S$ . Otherwise, if  $v$  is not included, then we can consider any independent set in  $S$ , so we multiply the out-weight of  $v$  by the weight of independent sets in the entirety of  $S$ .

For **Prune** to be useful, we must show that for any  $\lambda$ -balanced weighted graph  $\mathcal{G}$  with underlying graph  $G$  and subgraph  $S$  of  $G$  with  $\Gamma_G(S) \leq 1$ ,  $Z(\text{Prune}(\mathcal{G}, S)) = Z(\mathcal{G})$  and that  $\text{Prune}(\mathcal{G}, S)$  remains  $\lambda$ -balanced.

**Lemma 36.** Let  $\mathcal{G} = (G, w_+, w_-, W)$  be a  $\lambda$ -balanced weighted graph. Then for  $S \subseteq V(G)$  with  $\Gamma_G(S) \leq 1$ ,  $Z(\text{Prune}(\mathcal{G}, S)) = Z(\mathcal{G})$  and  $\text{Prune}(\mathcal{G}, S)$  is  $\lambda$ -balanced

*Proof.* When  $\Gamma_G(S) = \emptyset$ , we have that  $Z(\text{Prune}(\mathcal{G}, S)) = Z((G - S, w_+, w_-, W)) \cdot Z((G[S], w_+, w_-, W)) = Z(\mathcal{G})$ .  $\text{Prune}(\mathcal{G}, S)$  is also  $\lambda$ -balanced, as it is a subgraph of  $\mathcal{G}$ .

When  $\Gamma_G(S) = \{v\}$  for some  $v \in V(G)$ , let  $\mathcal{G}' = \text{Prune}(\mathcal{G}, S)$ . Then by considering for  $I \in \mathcal{I}(G)$  whether  $v \in I$ , we find

$$Z(\mathcal{G}) = w_+(v)w_-(\Gamma_G(v))Z(G - v - \Gamma_G(v)) + w_-(v)Z(G - v).$$

So then

$$\begin{aligned} Z(\mathcal{G}) &= w_+(v)w_-(\Gamma_G(v))Z((G[S''], w_+, w_-, 1)) + w_-(v)Z(G - S - v)Z((G[S'], w_+, w_-, 1)), \\ &= w'_+(v)w_-(\Gamma_{\mathcal{G}'}(v) \setminus S)Z(G - S - v - \Gamma_{\mathcal{G}'}(v)) + w'_-(v)Z(G - S - v) = Z(\mathcal{G}'). \end{aligned}$$

Then to show  $\text{Prune}(\mathcal{G}, S)$  is  $\lambda$ -balanced, we note  $\mathcal{G}$  is  $\lambda$ -balanced, so  $w_+(x) \leq \lambda w_-(x)$  for all  $x \in V(\mathcal{G})$ . So let  $x \in V(\mathcal{G}')$ . If  $x \neq v$ , then

$$w'_+(x) = w_+(x) \leq \lambda w_-(x) = \lambda w'_-(x).$$

If  $x = v$ , then  $w'_+(x) = w_+(v)w_-(S')Z((G[S''], w_+, w_-, 1))$ ,  $w'_-(x) = w_-(v)Z((G[S], w_+, w_-, 1))$ . Then as

$$\begin{aligned} &w_-(S')Z((G[S''], w_+, w_-, 1)) \\ &= \sum_{I \in \mathcal{I}(G[S]), I \cap S' = \emptyset} w_+(I)w_-(S \setminus I) \leq \sum_{I \in \mathcal{I}(G[S])} w_+(I)w_-(S \setminus I) \\ &= Z((G[S], w_+, w_-, 1)). \end{aligned}$$

Then by the definition of  $\lambda$ -balance,  $w_+(v) \leq \lambda w_-(v)$ , so

$$w'_+(v) = w_+(v)w_-(S')Z((G[S''], w_+, w_-, 1)) \leq \lambda w_-(v)Z((G[S], w_+, w_-, 1)) = \lambda w'_-(v).$$

Thus  $\text{Prune}(\mathcal{G}, S)$  is  $\lambda$ -balanced. □

**Algorithm 2**  $\text{ExactCount}(\mathcal{G}, Y)$ **Input:** Weighted graph  $\mathcal{G} = (G, w_+, w_-, W)$  and subset  $Y$  of  $V(G)$  such that  $G - Y$  is a forest.**Output:**  $Z(\mathcal{G})$ .

- (i) For all independent sets  $I \in \mathcal{I}(G)$ , compute  $Z(\mathcal{G} - Y - \Gamma_G(I))$ .
- (ii) Return  $\sum_{I \in \mathcal{I}(G)} w_+(I)w_-(Y \setminus I)Z(\mathcal{G} - Y - \Gamma_G(I))$ .

To execute  $\text{Prune}(\mathcal{G}, S)$ , we need a method of computing  $Z(\mathcal{G})$  exactly, to find the value of  $Z((G[S'], w_+, w_-, 1))$  and  $Z((G[S''], w_+, w_-, 1))$ . This is given by  $\text{ExactCount}$  in Algorithm 2. For it to be useful, we must find the running time.

**Lemma 37.** *Let  $\mathcal{G} = (G, w_+, w_-, W)$  be a weighted graph and  $Y \subseteq V(G)$  such that  $G - Y$  is a forest. Then  $\text{ExactCount}(\mathcal{G}, Y)$  returns  $Z(\mathcal{G})$  and has running time  $2^{|Y|}\text{poly}(|V(G)|)$ .*

*Sketch Proof.* For each independent set  $I$  contained in  $Y$ , the algorithm calculates the weight of independent sets in  $G - Y$ , with the vertices adjacent to  $I$  removed. To analyse the running time, we see that  $Y$  has at most  $2^{|Y|}$  independent sets by choosing vertices of  $Y$ . Then as  $G - Y$  is a forest, the weight of independent sets in  $G - Y$  can be calculated in time polynomial in  $|V(G)|$ . Thus the overall running time is  $2^{|Y|}\text{poly}(|V(G)|)$ .

Now that we can remove vertices of degree less than 2 from  $G$ , we can revisit the result of Chapter 3, to remove the condition that the minimum degree of input graph  $G$  is at least 2.

**Theorem 38.** *For all  $\lambda > 0$ , there is an FPTAS for  $Z(\mathcal{G})$  on  $\lambda$ -balanced weighted graphs  $\mathcal{G} = (G, w_+, w_-, W)$  where for each  $v \in G$  with  $d_G(v) \geq \omega(\lambda)$ ,  $d_G^2(v) \leq \omega_2(\lambda)$ .*

*Proof.* Let  $\varepsilon$  be an error tolerance, and let  $G$  be a graph such that for any vertex  $v$  in  $G$  with  $d_G(v) \geq \omega(\lambda)$ , we have  $d_G^2(v) \leq \omega_2(\lambda)$ . Then let  $\mathcal{G}$  be a  $\lambda$ -balanced weighted graph with underlying graph  $G$ . If  $\delta(G) \geq 2$ , we can apply  $\text{BaseCount}$  immediately. Otherwise, let  $Y = \{v \in V(G) \mid d_G(v) \leq 1\}$ . Then let  $\mathcal{G}' = \text{Prune}(\mathcal{G}, Y)$ . By Lemma 36,  $Z(\mathcal{G}') = Z(\mathcal{G})$ , and  $\mathcal{G}'$  is  $\lambda$ -balanced. Thus  $\text{BaseCount}(\mathcal{G}', \varepsilon)$  returns an  $\varepsilon$ -approximation for  $Z(\mathcal{G})$  in polynomial time.  $\square$

Now, we return to the preprocessing. The features of the graph that we will remove with  $\text{Prune}$  will be what are called *near-forests*.

**Definition 39** (Near-forest). *A graph  $G$  is a near-forest if it is the empty graph or there is a vertex  $v \in G$  such that  $G - v - \Gamma_G(v)$  is a forest.*

For these purposes, we consider the empty graph to be a forest. Then, using the running time of  $\text{ExactCount}$ , we can find the running time of  $\text{Prune}$  in terms of the maximum degree of the graph.

**Lemma 40.** *Let  $\mathcal{G} = (G, w_+, w_-, W)$  be a  $n$ -vertex weighted graph with maximum degree at most  $\Delta$ . Let  $X \subseteq V(G)$  span a near-forest in  $G$  with  $|\Gamma_G(X)| \leq 1$ . Then  $\text{Prune}(\mathcal{G}, X)$  can be computed in time at most  $2^\Delta \text{poly}(n)$ .*

*Proof.* If  $G[X]$  is a forest, let  $S = \emptyset$ . Otherwise, as  $X$  is a near-forest, there is a vertex  $v \in X$  such that  $G - v - \Gamma_G(v)$  is a forest, let  $S = \{v\} \cup \Gamma_G(v)$ . In either case,  $G[X] - S$  is a forest, so by Lemma 37, for every  $Y \subseteq X$ , we have that  $Z(\mathcal{G}[Y]) = \text{ExactCount}(\mathcal{G}[Y], S)$ . If  $S = \emptyset$ , we can compute  $Z(\mathcal{G}[Y])$  in time  $\text{poly}(n)$ . Otherwise, since  $v$  has degree at most  $\Delta$ , we can compute  $Z(\mathcal{G}[Y])$  in time  $2^\Delta \text{poly}(n)$ . To compute  $\text{Prune}$ , we need to compute  $Z(\mathcal{G}[Y])$  for 2 subsets  $Y$ , so we can compute  $\text{Prune}(\mathcal{G}, X)$  in time  $2 \cdot 2^\Delta \text{poly}(n) = \text{poly}(n)$ .  $\square$

At the end of the preprocessing, we will want the graph to be *reduced*.

**Definition 41** (Reduced graphs). *A graph  $G$  is reduced if for all non-empty subsets  $X$  of  $V(G)$  with  $G[X]$  a near-forest,  $|\Gamma_G(X)| \geq 2$ .*

For this, we introduce  $\text{Reduce}$ , which outputs a reduced graph. This is detailed in Algorithm 3.

**Lemma 42.** *Let  $\mathcal{G} = (G, w_+, w_-, W)$  be a  $\lambda$ -balanced weighted graph with no tree components and maximum degree at most  $\Delta$ . Then  $\text{Reduce}(\mathcal{G})$  returns a  $\lambda$ -balanced reduced weighted graph  $\mathcal{H}$  such that*



**Algorithm 3**  $\text{Reduce}(\mathcal{G})$ **Inputs:** Non-empty weighted graph  $\mathcal{G}$  with no tree components.**Output:** Reduced weighted graph  $\mathcal{G}_k$ .In this algorithm, we write  $\mathcal{G}_i = (G_i, w_+^i, w_-^i, W_i)$ .

- (i) Set  $i = 1$  and  $\mathcal{G}_1 = \mathcal{G}$ .
- (ii) If either  $G_i$  or some subgraph  $G_i - v$  for vertex  $v \in V(G)$  contains a component which is a near forest, then:
  - (a) Let  $\mathcal{G}_{i+1} = \text{Prune}(\mathcal{G}_i, V(C))$ .
  - (b) Increment  $i$  and go to (ii).
- (iii) Return  $\mathcal{G}_i$ .

- $Z(\mathcal{G}) = Z(\mathcal{H})$ ,
- $\mathcal{H}$  has maximum degree at most  $\Delta$ ,
- For any good slice  $f$ ,  $f(\mathcal{H}) \leq f(\mathcal{G})$ .

The running time of  $\text{Reduce}(\mathcal{G})$  is  $2^\Delta \text{poly}(n)$ .

*Sketch Proof.* By Lemma 36,  $\text{Prune}$  preserves  $\lambda$ -balance. Since  $\mathcal{H}$  is formed from  $\mathcal{G}$  by repeated calls to  $\text{Prune}$ ,  $\mathcal{H}$  is  $\lambda$ -balanced and has maximum degree at most  $\Delta$ , and  $Z(\mathcal{G}) = Z(\mathcal{H})$ .

Then, suppose that  $\mathcal{H}$  does not satisfy the condition in step (ii), but  $\mathcal{H}$  is not reduced, that is, it contains a subset  $X$  of the vertices that is a near-forest, such that  $|\Gamma_H(X)| \leq 1$ . If  $|\Gamma_H(X)| = \emptyset$ , then clearly  $G_i$  has a component that is a near forest, which would have been removed. Similarly, if  $|\Gamma_H(X)| = \{v\}$ , then  $G_i - v$  contains a near-forest which would have been removed. So  $\mathcal{H}$  must be reduced.

Lastly for the preprocessing, we need a method to remove any tree components that may be in the graph. For this, we define the *tree removal* of a graph.

**Definition 43** (Tree Removal). *Let  $G$  be a graph with tree components  $T_1, T_2, \dots, T_r$ . Then the tree removal of  $G$  is the graph  $\text{TR}(G) = G - V(T_1) - \dots - V(T_r)$ . The tree removal of a weighted graph  $\mathcal{G} = (G, w_+, w_-, W)$  is defined as*

$$\text{TR}(\mathcal{G}) = \left( \text{TR}(G), w_+, w_-, W \cdot \prod_{i=1}^r \text{ExactCount}(\mathcal{T}_i, \emptyset) \right),$$

where  $\mathcal{T}_i = (T_i, w_+, w_-, 1)$ .

The following lemma proves that  $\text{TR}(\mathcal{G})$  has the properties that we require.

**Lemma 44.** *Let  $\mathcal{G} = (G, w_+, w_-, W)$  be a weighted graph and  $\text{TR}(\mathcal{G})$  have the underlying graph  $H$ . Then*

- (i)  $H \subseteq G$ ,
- (ii)  $Z(\text{TR}(\mathcal{G})) = Z(\mathcal{G})$ ,
- (iii) If  $\mathcal{G}$  is  $\lambda$ -balanced, then  $\mathcal{H}$  is as well,
- (iv)  $\text{TR}(\mathcal{G})$  can be computed in time  $\text{poly}(|V(G)|)$ .

*Proof.* Since  $H = G - V(T_1) - \dots - V(T_r)$ ,  $H \subseteq G$ , and so if  $\mathcal{G}$  is  $\lambda$ -balanced, then  $\mathcal{H}$  is  $\lambda$ -balanced. Then

$$Z(\text{TR}(\mathcal{G})) = \prod_{i=1}^r \text{ExactCount}(\mathcal{T}_i, \emptyset) \cdot Z(\text{TR}(G)) = Z(\mathcal{T}_1) \cdot \dots \cdot Z(\mathcal{T}_r) \cdot Z(\mathcal{G} - V(T_1) - \dots - V(T_r)) = Z(\mathcal{G}).$$

Lastly, by Lemma 37, since each  $T_i$  is a tree,  $Z(\mathcal{T}_i)$  can be computed in time  $\text{poly}(|V(G)|)$ . Since there are at most  $|V(G)|$  tree components, we can compute  $\text{TR}(\mathcal{G})$  in time  $\text{poly}(|V(G)|)$ .  $\square$

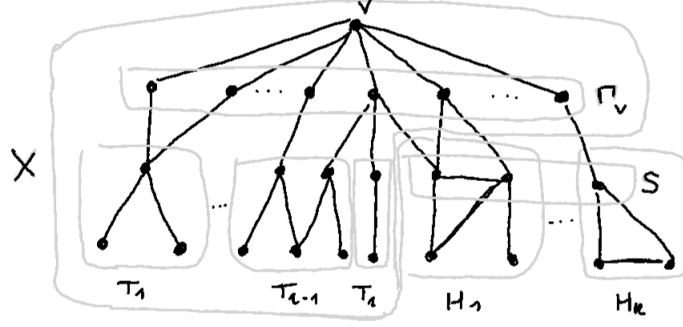


Figure 4.1: An example of the standard decomposition of a graph.

### 4.3 Graph decompositions

We now define two decompositions of graphs that will help to describe the algorithm. The first of these is the *standard decomposition* of a graph  $G$  from a vertex  $v \in G$ .

**Definition 45** (Standard decomposition). *Let  $G$  be a connected graph, and let  $v \in V(G)$ . The standard decomposition of  $G$  from  $v$  is the tuple  $(\Gamma_v, S, X, H_1, \dots, H_k; T_1, \dots, T_\ell)$ , where:*

- $\Gamma_v = \Gamma_G(v)$ .
- $H_1, \dots, H_k$  are the non-tree components of  $G - v - \Gamma_G(v)$ .
- $T_1, \dots, T_\ell$  are the tree components of  $G - v - \Gamma_G(v)$ .
- $S = (V(H_1) \cup V(H_2) \cup \dots \cup V(H_k)) \cap \Gamma_G^2(v)$ .
- $X = \{v\} \cup \Gamma_v \cup (V(T_1) \cup V(T_2) \cup \dots \cup V(T_k))$ .

An example of the standard decomposition of a graph is given in Figure 4.1. We note that if  $G$  is reduced, then there is a bound on the size of  $|S|$ .

**Lemma 46.** *Let  $G$  be a connected, reduced graph, and let  $v \in V(G)$ . Then in the standard decomposition of  $G$  from  $v$ ,  $|S| \geq 2$ .*

*Proof.* Since  $G[X] - v - \Gamma_G(v)$  is the union of trees,  $G[X]$  is a near forest. So by the definition of being reduced,  $|S| = |\Gamma_G(X)| \geq 2$ .  $\square$

The standard decomposition of a graph allows for the analysis of branching, where we pick some  $v \in G$ , and partition sets according to whether or not they are in the independent set, giving the relation

$$Z(\mathcal{G}) = w_-(v)Z(\mathcal{G} - v) + w_+(v)w_-(\Gamma_G(v))Z(\mathcal{G} - v - \Gamma_G(v)).$$

The standard decomposition allows us to bound the efficiency of this branching operation in terms of  $d_G(v)$ ,  $d_G^2(v)$  and  $|S|$ . However, the smaller  $|S|$  is, the less efficient the branching is. If  $|S| = 2$ , then suppose  $S = \{x, y\}$ . There is a strategy introduced in [1] that branches on  $y$  rather than  $v$ . However, when  $G$  is bipartite, branching on  $y$  is not as efficient. This leads to the definition of the *extended decomposition*.

**Definition 47** (Extended decomposition). *Let  $G$  be a connected graph, and let  $v \in V(G)$ . Suppose  $|S| = 2$  from the standard decomposition of  $G$  from  $v$ . The extended decomposition of  $G$  from  $v$  is the standard decomposition of  $G$  from  $v$ , along with the tuple  $(X^+, P, x, y, z, H)$ , where:*

- $\{x, y\} = S$ , and  $x$  is lexicographically less than  $y$ ,
- If  $d_{G-X}(y) \geq 2$ , then  $z = y$ , otherwise  $z$  is the unique closest vertex to  $y$  in  $G - X$  with  $d_{G-X}(z) \geq 2$ ,
- $P$  is the unique  $y - z$  path in  $G - X$ ,

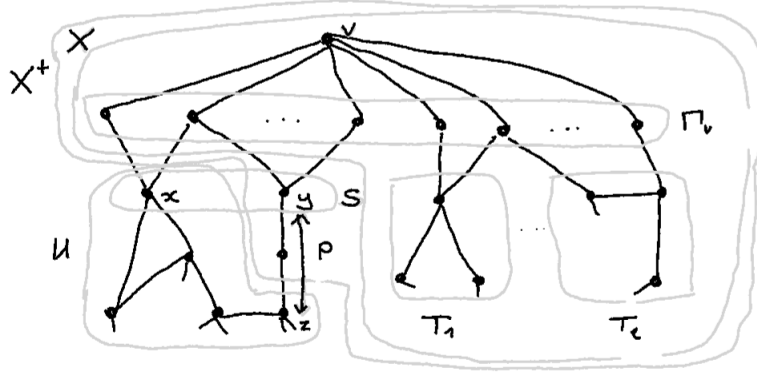


Figure 4.2: An example of the extended decomposition of a graph.

- $X^+ = X \cup V(P) \setminus \{z\}$ ,
- $H = G - X^+$ .

An example of the extended decomposition of a graph is given in Figure 4.2. By using the extended decomposition, we can branch on  $z$  rather than  $y$ , which will be more efficient. This means that the worst case of the analysis will be when  $|S| = 3$  for any graph.

**Lemma 48.** *Let  $G$  be a connected reduced graph, let  $v \in V(G)$  and suppose that in the standard decomposition of  $G$  from  $v$ ,  $|S| = 2$ . Then in the extended decomposition of  $G$  from  $v$ , we have  $x \notin V(P)$  and  $d_H(z) \geq 2$ .*

*Proof.* Suppose otherwise, that  $x \in V(P)$ . Then  $d_{G-x}(y) = 1$ . Let  $P'$  be the sub-path of  $P$  from  $y$  to  $x$ , and  $U = (X \cup V(P')) \setminus \{x\}$ . Then  $G[U]$  is a near-forest, as we can delete  $v$  as  $\Gamma_G(v)$  to obtain the forest  $(P' - x) \cup T_1 \cup \dots \cup T_\ell$ . However,  $\Gamma_G(U) = \{x\}$ , which contradicts the fact that  $G$  is reduced. So  $z \neq x$ . If  $z = y$ , then we would have  $X^+ = X$ , so  $d_H(z) = d_{G-X}(y) \geq 2$ . Otherwise,  $z \notin \{x, y\}$ , so  $d_H(z) = d_{G-X}(z) - 1 \geq 2$ .  $\square$

## 4.4 The approximation algorithm

We can now define the main approximation algorithm,  $\text{Count}_\lambda$ , in algorithm 4.

## 4.5 Proof of correctness

We must now prove correctness and running time for  $\text{Count}_\lambda$ .

**Lemma 49.** *Let  $\mathcal{G} = (G, w_+, w_-, W)$  be a non-empty  $\lambda$ -balanced weighted graph with no tree components, and let  $\varepsilon \in \mathbb{Q}$  with  $0 < \varepsilon < 1$ . Then,*

- (i)  $\text{Count}_\lambda(\mathcal{G}, \varepsilon)$  returns an  $\varepsilon$ -approximation of  $Z(\mathcal{G})$ ,
- (ii) Within time  $\text{poly}(n, 1/\varepsilon)$ ,  $\text{Count}_\lambda(\mathcal{G}, \varepsilon)$  either terminates or computes the arguments  $\mathcal{G}_{\text{in}}$  and  $\mathcal{G}_{\text{out}}$  to its recursive calls,
- (iii) At each recursive call  $\text{Count}_\lambda(\mathcal{H}, \varepsilon)$ ,  $\mathcal{H}$  is non-empty and  $\lambda$ -balanced and has no tree components.

*Sketch Proof.* Let  $\lambda > 0$ . First, we look at condition (iii). The possible arguments  $\mathcal{H}$  to a recursive call are either  $\mathcal{G}_{\text{in}}$  or  $\mathcal{G}_{\text{out}}$ . Each definition of  $\mathcal{G}_{\text{in}}$  and  $\mathcal{G}_{\text{out}}$  are formed from  $\mathcal{G}$  by deleting vertices and calling a combination of **Prune**, **Reduce** and **TR**. From the definition, deleting vertices preserves  $\lambda$ -balance. By Lemma 36, Lemma 42 and Lemma 44 respectively, each of the operations preserve  $\lambda$ -balance, so  $\mathcal{H}$  must also be  $\lambda$ -balanced. Since the last operation is always a call to **TR**,  $\mathcal{H}$  will have no tree components. Now for (i) and (ii), we must show correctness and running time for each step of  $\text{Count}_\lambda$ .

---

**Algorithm 4**  $\text{Count}_\lambda(\mathcal{G}, \varepsilon)$ 


---

**Parameters:**  $\lambda > 0$ .

**Inputs:** Non-empty,  $\lambda$ -balanced weighted graph  $\mathcal{G} = (G, w_+, w_-, W)$  with no tree components, and error tolerance  $\varepsilon$ .

**Outputs:** An  $\varepsilon$ -approximation for  $Z(\mathcal{G})$ .

(i) If  $\Delta(G) \geq 11$ :

(a) Let  $v$  be the lexicographically least vertex of degree at least 11 in  $G$ , and let

$$\begin{aligned} \mathcal{G}_{out} &= (G_{out}, \cdot, \cdot, W_{out}) \leftarrow \text{TR}(\mathcal{G} - v), \\ W'_{out} &\leftarrow w_-(v), \\ C_{out} &\leftarrow \text{If } G_{out} \text{ is empty then } W_{out}, \text{ else } \text{Count}_\lambda(\mathcal{G}_{out}, \varepsilon), \\ \mathcal{G}_{in} &= (G_{in}, \cdot, \cdot, W_{in}) \leftarrow \text{TR}(\mathcal{G} - v - \Gamma_G(v)), \\ W'_{in} &\leftarrow w_+(v) \cdot w_-(\Gamma_G(v)), \\ C_{in} &\leftarrow \text{If } G_{in} \text{ is empty then } W_{in}, \text{ else } \text{Count}_\lambda(\mathcal{G}_{in}, \varepsilon). \end{aligned}$$

(b) Return  $W'_{out} \cdot C_{out} + W'_{in} \cdot C_{in}$ .

(ii) Replace  $\mathcal{G}$  with  $\text{Reduce}(\mathcal{G})$ .

(iii) If  $G$  has no vertices of degree at least  $\omega(\lambda)$  and 2-degree  $\omega_2(\lambda)$ , return  $\text{BaseCount}(\mathcal{G}, \varepsilon)$ .

(iv) If  $G$  has average degree at most  $\omega(\lambda) - 1$ , let  $v$  be the lexicographically least vertex of degree  $\omega(\lambda)$  and 2-degree greater than  $\omega_2(\lambda)$ . Otherwise, let  $v$  be lexicographically least vertex maximising  $d_G^2(v)$  subject to  $d_G(v) \geq 2m/n$ .

(v) Let  $G_v$  be the component of  $G$  containing  $v$ , and let  $S$  be as defined in the standard decomposition of  $G_v$  from  $v$ . If  $|S| \geq 3$ , then let

$$\begin{aligned} \mathcal{G}_{out} &= (G_{out}, \cdot, \cdot, W_{out}) \leftarrow \text{TR}(\mathcal{G} - v), \\ W'_{out} &\leftarrow w_-(v), \\ \mathcal{G}_{in} &= (G_{in}, \cdot, \cdot, W_{in}) \leftarrow \text{TR}(\mathcal{G} - v - \Gamma_G(v)), \\ W'_{in} &\leftarrow w_+(v) \cdot w_-(\Gamma_G(v)). \end{aligned}$$

(vi) Otherwise, if  $|S| = 2$ , consider the extended decomposition of  $G_v$  from  $v$ , and let

$$\begin{aligned} \mathcal{G}_{out} &= (G_{out}, \cdot, \cdot, W_{out}) \leftarrow \text{TR}(\text{Prune}(\mathcal{G} - z, X^+), \\ W'_{out} &\leftarrow w_-(v), \\ \mathcal{G}_{in} &= (G_{in}, \cdot, \cdot, W_{in}) \leftarrow \text{TR}(\text{Prune}(\mathcal{G} - z - \Gamma_G(z), X^+ \setminus \Gamma_G(z))), \\ W'_{in} &\leftarrow w_+(z) \cdot w_-(\Gamma_G(z)). \end{aligned}$$

(vii) Define

$$\begin{aligned} C_{out} &\leftarrow \text{If } G_{out} \text{ is empty then } W_{out}, \text{ else } \text{Count}_\lambda(\mathcal{G}_{out}, \varepsilon), \\ C_{in} &\leftarrow \text{If } G_{in} \text{ is empty then } W_{in}, \text{ else } \text{Count}_\lambda(\mathcal{G}_{in}, \varepsilon). \end{aligned}$$

(viii) Return  $W'_{out} \cdot C_{out} + W'_{in} \cdot C_{in}$ .

---

Let  $\mathcal{G}$  be a  $\lambda$ -balanced weighted graph, and  $\varepsilon$  be an error tolerance. If  $\Delta(G) \geq 11$ , step (i) is executed.  $\mathcal{G}_{out}$  and  $\mathcal{G}_{in}$  are computed by using the TR operation. By Lemma 44,

$$\begin{aligned} Z(\mathcal{G}) &= w_-(v)Z(\mathcal{G} - v) + w_+(v)w_-(\Gamma_G(v))Z(\mathcal{G} - v - \Gamma_G(v)) \\ &= W'_{out}Z(\mathcal{G} - v) + W'_{in}Z(\mathcal{G} - v - \Gamma_G(v)) \\ &= W'_{out}Z(\mathcal{G}_{out}) + W'_{in}Z(\mathcal{G}_{in}). \end{aligned}$$

Since TR preserves  $\lambda$ -balance,  $\mathcal{G}_{out}$  and  $\mathcal{G}_{in}$  are  $\lambda$ -balanced and have no tree components, so we can give them as input to  $\text{Count}_\lambda$ . And again by Lemma 44, each of  $Z(\mathcal{G}_{out})$  and  $Z(\mathcal{G}_{in})$  can be computed in time  $\text{poly}(n)$ .

Otherwise, step (ii) is run, which by Lemma 42 runs in time  $2^{\Delta(G)}\text{poly}(n) = \text{poly}(n)$ , and  $\text{Reduce}(\mathcal{G})$  is  $\lambda$ -balanced and  $Z(\mathcal{G})$  is unchanged. Since  $\mathcal{G}$  is now reduced, we have  $\delta(G) \geq 2$ , so if  $G \in \mathcal{D}_{\omega(\lambda), \omega_2(\lambda)}$ , we can run  $\text{ExactCount}$  on  $\mathcal{G}$  in time  $\text{poly}(n, 1/\varepsilon)$ .

Clearly, step (iv) can be computed in time  $\text{poly}(n)$ .

By noting that  $\Gamma_{G-z}(X^+) = \{x\}$  and  $\Gamma_{G-z-\Gamma_G(z)}(X^+ \setminus \Gamma_G(z)) \subseteq \{x\}$ , by using the same argument as for part (i), we find that steps (v) and (vi) are correct. Step (v) also has the same argument for the running time. For the running time of step (vi), we note that  $|\Gamma_{G-z}(X^+)| = 1$ . Moreover,  $G[X^+] - v - \Gamma_G(v)$  is a forest, so  $G[X^+]$  is a near-forest. So by Lemma 37 and Lemma 36,  $\mathcal{G}_{out}$  can be computed in time  $\text{poly}(n)$ . Similarly,  $G[X^+ \setminus \Gamma_G(z)]$  is a near-forest, so  $\mathcal{G}_{in}$  can also be computed in time  $\text{poly}(n)$ .

## 4.6 Correctness of potential functions

We now move on to show that the potential functions we use indeed bound the running time of  $\text{Count}_\lambda$ . To do this, we show that for each step of  $\text{Count}_\lambda$ , for any good slice, that the arguments passed to the recursive calls  $\mathcal{G}_{out}$  and  $\mathcal{G}_{in}$  have significantly reduced potential as compared to  $\mathcal{G}$ . We start by analysing  $\mathcal{G}_{out}$  and  $\mathcal{G}_{in}$  for step (v).

**Lemma 50.** *Let  $\mathcal{G}$  be a weighted graph with underlying graph  $G$  which is connected and reduced. Let  $v \in V(G)$ . Suppose  $f(m, n) = \rho m + \sigma n$  is a good slice. Let  $\mathcal{G}_{out} = \text{TR}(\mathcal{G} - v)$  and  $\mathcal{G}_{in} = \text{TR}(\mathcal{G} - v - \Gamma_G(v))$ , as in step (v). Consider the standard decomposition of  $G$  from  $v$ . Then*

$$f(\mathcal{G}) - f(\mathcal{G}_{out}) = \rho d_G(v) + \sigma,$$

$$f(\mathcal{G}) - f(\mathcal{G}_{in}) \geq \rho \left\lceil \frac{d_G^2(v) + d_G(v) + |S|}{2} \right\rceil + \sigma(1 + d_G(v)).$$

If  $G$  is bipartite, then

$$f(\mathcal{G}) - f(\mathcal{G}_{in}) \geq \begin{cases} \rho d_G^2(v) + \sigma(1 + d_G(v)) & \text{if } \sigma \geq 0 \\ \rho d_G^2(v) + \sigma \left\lfloor \frac{d_G^2(v) + d_G(v) + 2 - |S|}{2} \right\rfloor & \text{if } \sigma < 0. \end{cases}$$

*Sketch Proof.* Let  $G_{out} = \text{TR}(G - v)$  be the underlying graph of  $\mathcal{G}_{out}$ , and  $G_{in} = \text{TR}(G - v - \Gamma_G(v))$  be the underlying graph of  $\mathcal{G}_{in}$ . By the definition of reduced graphs, so any subgraph  $Y \subseteq V(G)$  that is a tree, we have  $|\Gamma_G(Y)| \geq 2$ , so  $G - v$  has no tree components, so  $G_{out} = G - v$ . Thus  $f(G) - f(G_{out}) = \rho d_G(v) + \sigma$ . For  $G_{in}$ , considering the standard decomposition of  $G_v$  from  $v$ , we have that  $G_{in} = G - v - \Gamma_G(v) - V(T_1) - \dots - V(T_\ell)$ . We let  $m_1$  be the number of edges in  $G[\Gamma_G(v)]$  and  $m_2$  be the number of edges between  $\Gamma_G(v)$  and  $\Gamma_G^2(v)$ . Using this notation, we have

$$f(G) - f(G_{in}) = \rho(d_G(v) + m_1 + m_2) + \sigma(1 + d_G(v)) + \sum_{i=1}^{\ell} f(T_i).$$

By the definition of a good slice and the fact that  $d_G^2(v) = d_G(v) + 2m_1 + m_2$ , we find that

$$f(G) - f(G_{in}) \geq \rho(d_G^2(v) - m_1) + \sigma(1 + d_G(v) + \ell). \quad (4.1)$$

Since  $G$  is reduced, each tree component  $T_i$  has at least 2 edges connected to  $\Gamma_G(v)$ , and each vertex of  $S$  sends at least 1 edge to  $\Gamma_G(v)$  so  $m_2 \geq 2\ell + |S|$ . Therefore,  $m_1 \leq \left\lfloor \frac{d_G^2(v) - d_G(v) - |S|}{2} \right\rfloor - \ell$ . By (4.1),

$$f(G) - f(G_{in}) \geq \rho \left\lceil \frac{d_G^2(v) + d_G(v) + |S|}{2} \right\rceil + \sigma(1 + d_G(v) + \ell).$$

Then, by the definition of a good slice,  $\rho\ell + \sigma\ell \geq 0$ , so the result follows. For the bipartite case, the first inequality follows by the same argument. Otherwise, suppose  $\sigma < 0$ , as  $m_2 \geq 2\ell + |S|$ ,  $\ell \leq \left\lfloor \frac{d_G^2(v) - d_G(v) - |S|}{2} \right\rfloor$ . So by 4.1

$$f(G) - f(G_{in}) \geq \rho d_G^2(v) + \sigma \left\lfloor \frac{d_G^2(v) + d_G(v) + 2 - |S|}{2} \right\rfloor.$$

The result follows.

We now move onto the analysis for step (vi). We need the following lemma to prove the bounds.

**Lemma 51.** *Let  $H$  be a graph, and let  $z \in V(H)$  with  $d_H(z) \geq 2$ , and suppose  $H - z$  has no tree components. Let  $m_1$  be the number of edges in  $H[\Gamma_H(z)]$  and let  $m_2$  be the number of edges between  $\Gamma_H(z)$  and  $\Gamma_H^2(z)$ . Let  $r$  be the number of tree components of  $H - z - \Gamma_H(z)$ . Then  $m_1 + m_2 \geq r + 2$ .*

*Sketch Proof.* If  $H$  is not connected, consider only the component of  $H$  that contains  $z$ . Let  $H^- = H - z - \Gamma_H(z)$ , and let  $T_1, \dots, T_r$  be the tree components of  $H^-$ . Each component of  $H^-$  sends at least one edge to  $\Gamma_H(z)$ , so  $H^-$  has at most  $m_2$  components, so  $m_2 \geq r$ . If  $m_2 \geq r + 2$ , then  $m_1 + m_2 \geq r + 2$ , so we're done. Otherwise, we have 2 cases.

**Case 1:  $m_2 = r$ .** Suppose  $m_2 = r$ . If  $m_1 \geq 2$ , we're done. Otherwise,  $H[\Gamma_H(z)]$  is a forest, and each  $T_i$  sends one edge to  $\Gamma_H(z)$ , so  $H - z$  is a forest, which is a contradiction.

**Case 2:  $m_2 = r + 1$ .** Suppose  $m_2 = r + 1$ . If  $m_1 \geq 1$ , we're done. Otherwise, suppose  $m_1 = 0$ . If the only components of  $H^-$  are the trees  $T_1, \dots, T_r$ , then one component  $T_i$  receives 2 edges from  $\Gamma_H(z)$ ,  $e_1$  and  $e_2$ . If we remove  $e_1$ , then  $H - z - e_1$  is a forest. Since  $d_H(z) \geq 2$ , it has at least 2 tree components. Re-adding  $e_1$ , we still have a tree component, which is a contradiction. Otherwise, if we have  $r + 1$  components, being  $T_1, \dots, T_r$  trees, and  $C$  which is not a tree. Each has exactly one edge from  $\Gamma_H(z)$ . Since  $d_H(z) \geq 2$ , we have at least one tree component, contradicting the hypothesis.

Now we can prove the following bounds for  $\mathcal{G}_{out}$  and  $\mathcal{G}_{in}$  in step (vi) of  $\text{Count}_\lambda$ .

**Lemma 52.** *Let  $\mathcal{G}$  be a weighted graph with underlying graph  $G$  which is connected and reduced. Let  $v \in V(G)$ . Suppose  $f(m, n) = \rho m + \sigma n$  is a good slice. Consider the standard decomposition of  $G$  from  $v$ , and let  $|S| = 2$ . Consider the extended decomposition of  $G$  from  $v$ , and let*

$$\begin{aligned} \mathcal{G}_{out} &= \text{TR}(\text{Prune}(G - z, X^+)), \\ \mathcal{G}_{in} &= \text{TR}(\text{Prune}(G - z - \Gamma_G(z), X^+ \setminus \Gamma_G(z))), \end{aligned}$$

as in step (vi) of  $\text{Count}_\lambda$ . Then

$$\begin{aligned} f(\mathcal{G}) - f(\mathcal{G}_{out}) &\geq \rho(1 + d_G(v)) + \sigma, \\ f(\mathcal{G}) - f(\mathcal{G}_{in}) &\geq \rho \left\lfloor \frac{d_G^2(v) + d_G(v) + 4}{2} \right\rfloor + \sigma(1 + d_G(v)). \end{aligned}$$

If  $G$  is bipartite, then

$$f(\mathcal{G}) - f(\mathcal{G}_{in}) \geq \begin{cases} \rho(1 + d_G^2(v)) + \sigma(1 + d_G(v)) & \text{if } \sigma \geq 0 \\ \rho d_G^2(v) + \sigma \left\lfloor \frac{d_G^2(v) + d_G(v) - 2}{2} \right\rfloor & \text{if } \sigma < 0. \end{cases}$$

*Sketch Proof.* Consider the extended decomposition of  $G_v$  from  $v$ . Let  $G_{out} = \text{TR}(G - z - X^+) = \text{TR}(H - z)$  be the underlying graph of  $\mathcal{G}_{out}$ , and  $G_{in} = \text{TR}(G - z - \Gamma_G(z) - X^+)$  be the underlying graph of  $\mathcal{G}_{in}$ . We show that  $H - z$  has no tree components. Suppose that  $C$  was a tree component of  $H - z$ . As  $G$  is reduced,  $|\Gamma_G(V(C))| \geq 2$ . As  $\Gamma_H(V(C)) \subseteq \{z\}$ ,  $C$  must send an edge to  $X^+$ , so  $x \in V(C)$ . Then, let  $C^+$  be  $G[V(C) \cup X^+]$ . So  $C^+$  is a component of  $G - z$ .  $C^+ - v - \Gamma_G(v) = T_1 \cup \dots \cup T_\ell \cup (P - z) \cup C$  is a forest, so  $C^+$  is a near forest. However,  $\Gamma_G(V(C^+)) = \{z\}$ , contradicting the fact that  $G$  is reduced. So  $G_{out} = \text{TR}(H - z) = H - z$ . By the definition of a good slice, and the fact that  $d_H(z) \geq 2$ , we find that  $f(G_{out}) \geq f(H) - \rho$ , and by letting  $m$  be the number of edges between  $X^+$  and  $V(H)$ , we find  $f(G) - f(H) = (\rho d_G(v) + \sigma) + (f(G[X^+] - v) + \rho m)$ . Now, considering  $f(G[X^+] - v)$ , let  $D_1, \dots, D_r$  be

the tree components of  $G[X^+] - v$ , and  $D'_1, \dots, D'_s$  be the non-tree components. By the definition of a good slice, we have that

$$f(G[X^+] - v) = \sum_{i=1}^r f(D_i) + \sum_{i=1}^s f(D'_i) \geq \sigma R.$$

By the definition of reduce,  $G - v$  has no tree components, so each tree component of  $G[X^+] - v$  must be connected to  $H$  by at least one edge. So  $m \geq R$ . Thus, we have

$$f(G) - f(H) \geq (\rho d_G(v) + \sigma) + (\sigma R + \rho R) \geq \rho d_G(v) + \sigma.$$

The stated bound follows.

For  $\mathcal{G}_{in}$ , note that  $G_{in} = \text{TR}(G - z - \Gamma_G(z) - X^+) = \text{TR}(H - z - \Gamma_H(z))$ . We consider  $r$ , the number of tree components of  $H - z - \Gamma_H(z)$ ,  $m_1$ , the number of edges in  $H[\Gamma_H(z)]$  and  $m_2$ , the number of edges between  $\Gamma_H(z)$  and  $\Gamma_H^2(z)$ . By the definition of a good slice, we find that

$$f(\mathcal{G}_{in}) \leq f(H) - \rho(d_H(z) + m_1 + m_2) - \sigma(1 + d_H(z) + r).$$

Since  $H - z$  has no tree components, by Lemma 51, we have that  $m_1 + m_2 \geq r + 2$ . Therefore  $f(\mathcal{G}_{in}) \leq f(H) - \rho$ . Since  $H$  is formed from  $G - X$  by removing a (possibly empty) path  $P - z$ , we can apply the bounds from Lemma 50. The stated bounds are obtained from those.

We now define the pre-potential and potential functions that we will use.

**Definition 53.** Let  $s$  be a positive integer. Let  $-1 = k_0 < k_1 < \dots < k_s = \infty$ , and let  $f_1, \dots, f_s : \mathbb{R}^2 \rightarrow \mathbb{R}$  be linear functions. Then the pre-potential with boundary points  $k_0, \dots, k_s$  and slices  $f_1, \dots, f_s$  is the function  $f : \mathbb{R}_{\geq 0}^2 \rightarrow \mathbb{R}$  defined as follows. If  $n = 0$ , then  $f(m, n) = 0$ . Otherwise,  $f(m, n) = f_i(m, n)$  where  $k_{i-1} < 2m/n \leq k_i$ .  $f$  is a valid pre-potential if it satisfies the following properties.

(P1)  $f_1, \dots, f_s$  are good slices.

(P2)  $f_s(m, n) = \sigma_s n$  for some  $\sigma_s > 0$ .

(P3) Every integer in  $[6, k_{s-1}]$  is a boundary point.

(P4) For all  $n > 0$  and  $m > 0$ ,  $f(m, n) = \min_{j \in [s]} f_j(m, n)$ .

The potential corresponding to  $f$  is the map  $f_+$  on the class of all graphs given by

$$f_+(G) = \begin{cases} f_s(|E(G)|, |V(G)|) & \text{if } \Delta(G) \geq 11 \\ f(|E(G)|, |V(G)|) & \text{otherwise.} \end{cases}$$

If  $\mathcal{G} = (G, w_+, w_-, W)$  is a weighted graph, then we write  $f_+(\mathcal{G}) = f_+(G)$ . We say  $f_+$  is a valid potential if it is constructed this way from a valid pre-potential  $f$ .

We now use the 2-degree bound we obtain from Definition 31. This will replace the  $d_G^2(v)$  factor in the bounds.

**Definition 54.** Let  $\lambda > 0$ . For all  $k < \omega(\lambda) - 1$ , let  $D'_2(k) = \omega_2(\lambda) + 1$ . Otherwise, if  $k \geq \omega(\lambda) - 1$ , let  $D'_2(k) = \max\{2k, D_2(k)\}$ . We note that if a graph  $G$  has average degree in a range  $(k_j, k_{j+1}]$  for some boundary points of a valid pre-potential function, then for step (iv) of count,  $d_G^2(v) \geq D'_2(k_j)$ .

Now we can find the difference between the potentials of the input graphs and subproblems.

**Lemma 55.** Let  $f$  be a valid pre-potential with boundary points  $k_0, \dots, k_s$  and slices  $f_1, \dots, f_s$ , and for each  $i \in \{1, \dots, s\}$ , we have  $f_i = \rho_i m + \sigma_i n$ . Let  $\varepsilon$  be an error tolerance, let  $G_0$  be a graph, and let  $\mathcal{G}_0 = \text{TR}(G_0, \lambda, 1, 1)$ . Suppose that  $G_0$  is non-empty. Then when  $\text{Count}_\lambda(\mathcal{G}_0, \varepsilon)$  is executed, the following properties hold for every recursive call  $\text{Count}_\lambda(\mathcal{G}, \varepsilon)$ , where  $\mathcal{G} = (G, w_+, w_-, W)$ :

1.  $f^+(\mathcal{G}) \geq 0$  with equality if and only if  $\text{Count}_\lambda(\mathcal{G}, \varepsilon)$  halts in time  $\text{poly}(n, 1/\varepsilon)$  with no further recursive calls.

2. If  $\text{Count}_\lambda(\mathcal{G}, \varepsilon)$  makes recursive calls to  $\text{Count}_\lambda(\mathcal{G}_{\text{out}}, \varepsilon)$  and  $\text{Count}_\lambda(\mathcal{G}_{\text{in}}, \varepsilon)$ , then  $f^+(\mathcal{G}_{\text{out}}) < f^+(\mathcal{G})$  and  $f^+(\mathcal{G}_{\text{in}}) < f^+(\mathcal{G})$ . Moreover, the following stronger bounds hold. If  $\Delta(G) \geq 11$ , then

$$\begin{aligned} f^+(\mathcal{G}) - f^+(\mathcal{G}_{\text{out}}) &\geq f_s(0, 1), \\ f^+(\mathcal{G}) - f^+(\mathcal{G}_{\text{in}}) &\geq f_s(0, 12). \end{aligned}$$

Otherwise, there exists  $j \in [s]$  and positive integer  $x$  satisfying  $\max\{\omega(\lambda), \lfloor k_{j-1} \rfloor + 1\} \leq x \leq 10$  such that

$$\begin{aligned} f^+(\mathcal{G}) - f^+(\mathcal{G}_{\text{out}}) &\geq f_j(x, 1), \\ f^+(\mathcal{G}) - f^+(\mathcal{G}_{\text{in}}) &\geq f_j\left(\left\lfloor \frac{x + D'_2(k_{j-1}) + 3}{2} \right\rfloor, 1 + x\right). \end{aligned}$$

If  $G_0$  is bipartite, then

$$f^+(\mathcal{G}) - f^+(\mathcal{G}_{\text{in}}) \geq \begin{cases} f_j(D'_2(k_{j-1}), 1 + x) & \text{if } \sigma_j \geq 0 \\ f_j(D'_2(k_{j-1}), \lfloor \frac{x + D'_2(k_{j-1}) - 1}{2} \rfloor) & \text{if } \sigma_j < 0. \end{cases}$$

*Sketch Proof.* **Property (i).**  $G$  has no tree components and is non-empty, so  $|E(G)| \geq |V(G)| > 0$ , so by (P1),  $f_i(G) \geq 0$  for all  $i$ , so  $f^+(\mathcal{G}) \geq 0$ . If  $f^+(\mathcal{G}) = 0$ , then  $f_s(G) = 0$ , so by (P2) it must be that  $\Delta(G) < 11$ . So step (ii) is executed. If  $\text{Reduce}(\mathcal{G})$  is empty, we are done in time  $\text{poly}(n, 1/\varepsilon)$ . Otherwise, let  $i$  be such that  $f^+(\mathcal{G}) = f_i(\mathcal{G})$ . By (P4) and Lemma 42, we have that

$$f^+(\text{Reduce}(\mathcal{G})) \leq f_i(\text{Reduce}(\mathcal{G})) \leq f_i(\mathcal{G}) = f^+(\mathcal{G}) = 0.$$

However, if  $f^+(\text{Reduce}(\mathcal{G})) = 0$ , then  $\text{Reduce}(\mathcal{G})$  must be 2-regular, consisting of a union of disjoint cycles. But cycles are near-forests, contradicting the fact that  $\text{Reduce}(\mathcal{G})$  is reduced.

**Property (ii).** We split into cases depending on which steps of  $\text{Count}_\lambda$  are executed.

**Case 1:** Step (i) is executed, so  $\Delta(G) \geq 11$ . Then

$$\begin{aligned} f^+(\mathcal{G}_{\text{out}}) &\leq f_s(\mathcal{G}_{\text{out}}) \leq f_s(G - v) = f_s(G) - \sigma_s, \\ f^+(\mathcal{G}_{\text{in}}) &\leq f_s(\mathcal{G}_{\text{in}}) \leq f_s(G - v - \Gamma_G(v)) = f_s(G) - (1 + d_G(v))\sigma_s \leq f_s(G) - 12\sigma_s, \end{aligned}$$

as required.

For the next cases, we will have executed step (ii), so write  $\mathcal{G}' = \text{Reduce}(\mathcal{G})$ . We can assume  $\mathcal{G}'$  is non-empty, as otherwise step (iii) would be executed. We can also assume  $\Delta(\mathcal{G}') \leq \Delta(\mathcal{G}) \leq 10$ , and that  $\mathcal{G}'$  is  $\lambda$ -balanced and has no tree components. Therefore,  $f^+(\mathcal{G}') \leq f^+(\mathcal{G})$ .

**Case 2:** Step (v) is executed. Let  $j$  be such that  $k_{j-1} < 2|E(\mathcal{G}')|/|V(\mathcal{G}')| \leq k_j$ , so  $f^+(\mathcal{G}') = f_j(\mathcal{G}')$ . Then by the fact  $\mathcal{G}'$  is reduced and Lemma 50, we have that

$$f^+(\mathcal{G}) - f^+(\mathcal{G}_{\text{in}}) \geq \rho_j \left\lfloor \frac{d_G^2(v) + d_G(v) + |S|}{2} \right\rfloor + \sigma_j(1 + d_G(v)),$$

and if  $G$  is bipartite, then

$$f^+(\mathcal{G}) - f^+(\mathcal{G}_{\text{in}}) \geq \begin{cases} \rho_j d_G^2(v) + \sigma_j(1 + d_G(v)) & \text{if } \sigma_j \geq 0 \\ \rho_j d_G^2(v) + \sigma_j \left\lfloor \frac{d_G^2(v) + d_G(v) + 2 - |S|}{2} \right\rfloor & \text{if } \sigma_j < 0. \end{cases}$$

Since the average degree of  $\mathcal{G}'$  is in the range  $(k_{j-1}, k_j]$ , by definition 54,  $d_G^2(v) \geq D'_2(k_{j-1})$ . Also, when step (v) is executed, we have  $|S| \geq 3$ . So we have

$$f^+(\mathcal{G}) - f^+(\mathcal{G}_{\text{in}}) \geq \rho_j \left\lfloor \frac{D'_2(k_{j-1}) + d_G(v) + 3}{2} \right\rfloor + \sigma_j(1 + d_G(v)),$$

and if  $G$  is bipartite, then

$$f^+(\mathcal{G}) - f^+(\mathcal{G}_{\text{in}}) \geq \begin{cases} \rho_j D'_2(k_{j-1}) + \sigma_j(1 + d_G(v)) & \text{if } \sigma_j \geq 0 \\ \rho_j D'_2(k_{j-1}) + \sigma_j \left\lfloor \frac{D'_2(k_{j-1}) + d_G(v) - 1}{2} \right\rfloor & \text{if } \sigma_j < 0. \end{cases}$$



Then,  $\max\{\omega(\lambda), \lfloor k_{j-1} \rfloor + 1\} \leq d_G(v) \leq 10$ , so by choosing  $x = d_G(v)$ , we get the required bounds. The bound for  $f^+(\mathcal{G}) - f^+(\mathcal{G}_{out})$  holds following a similar argument.

**Case 3.** Step (vi) is executed. Again, if we follow a similar argument as case 2, we arrive at tighter bounds than in step 2, so we can use those bounds instead.

For the following lemma, we use some notation. For all integers  $b \geq 2$  and  $a_1, \dots, a_b \geq 0$ , we write  $\tau(a_1, \dots, a_b)$  for the unique positive solution for  $x$  in  $\sum_{i=1}^b x^{-a_i} = 1$ . To analyse the runtime when given a potential function, we use the theory of branching factors. If an algorithm recurses in one of  $b \geq 1$  ways, and  $p_i \geq 1$  is the number of subproblems created by the  $i$ th branch for  $1 \leq i \leq b$ . Then suppose if the original instance has potential  $x$ , then the  $j$ th part of the  $i$ th branch has potential  $x - y_{i,j}$ , where  $y_{i,j} > 0$ . Also suppose that when an instance has potential 0, then it can be solved without recursing further. Then the following recurrence holds for the worst-case running time  $T(x)$  when the input has potential  $x$ .

$$T(x) \leq \begin{cases} \max\{\sum_{j=1}^{p_i} T(x - y_{i,j}) \mid 1 \leq i \leq b\} + \text{poly}(n) & \text{if } x > 0 \\ \text{poly}(n) & \text{otherwise.} \end{cases}$$

Then the solution to the recurrence is

$$T(x) = O^*(\max_i \{\tau(y_{i,1}, \dots, y_{i,p_i})\}^x).$$

Now, we can use this recurrence to find the worst-case running time for inputs with potential  $x$ . We do this in the following lemma for general graphs.

**Lemma 56.** *Let  $G$  be a graph which is not a forest. Let  $f$  be a valid pre-potential with boundary points  $k_0, \dots, k_s$  and slices  $f_1, \dots, f_s$ , and for each  $i \in [s]$ , let  $f_i = \rho_i m + \sigma_i n$ . For all  $i \in [s]$  and  $d \geq \omega(\lambda)$ , let*

$$T_{i,d} = \tau \left( f_i(d, 1), f_i \left( \left\lceil \frac{d + D'_2(k_{i-1}) + 3}{2} \right\rceil, 1 + d \right) \right),$$

and

$$T = \max(\{T_{i,d} \mid i \in [s], \max\{\omega(\lambda), \lfloor k_{i-1} \rfloor + 1\} \leq d \leq 10\} \cup \{\tau(\sigma_s, 12\sigma_s)\}).$$

If  $T \leq 2$ , then  $\text{Count}_\lambda(\text{TR}(G, \lambda, 1, 1), \varepsilon)$  has running time  $O(2^{\sigma_s n})\text{poly}(n, 1/\varepsilon)$ .

*Sketch Proof.* Let  $\mathcal{G}_0 = \text{TR}((G, \lambda, 1, 1))$  be the input graph, which is non-empty. By Lemma 55, we have the following bounds. Let  $z_{i,\lambda} = \max\{\omega(\lambda), \lfloor k_{i-1} \rfloor + 1\}$ . Either,

$$\begin{aligned} f^+(\mathcal{G}_{out}) &\leq f^+(\mathcal{G}) - f_s(0, 1) \leq f^+(\mathcal{G}), \\ f^+(\mathcal{G}_{in}) &\leq f^+(\mathcal{G}) - f_s(0, 12) \leq f^+(\mathcal{G}), \end{aligned}$$

or, there exist  $i \in [s]$  and integer  $d$  with  $z_{i,\lambda} \leq d \leq 10$  such that

$$\begin{aligned} f^+(\mathcal{G}_{out}) &\leq f^+(\mathcal{G}) - f_i(d, 1) \leq f^+(\mathcal{G}), \\ f^+(\mathcal{G}_{in}) &\leq f^+(\mathcal{G}) - f_i \left( \left\lceil \frac{d + D'_2(k_{i-1}) + 3}{2} \right\rceil, 1 + d \right) \leq f^+(\mathcal{G}). \end{aligned}$$

. Then we can take the values of  $y_{i,j}$  to be the right hand side of the above inequalities. If we let  $R_\varepsilon(x)$  be the worst case running time for input with potential  $x$ , then we find that  $R_\varepsilon(x) = O(T^x)\text{poly}(n, 1/\varepsilon)$ . By hypothesis  $T = 2$ , so  $R_\varepsilon(x) = O(2^x)\text{poly}(n, 1/\varepsilon)$ . And as  $f^+$  is a potential function,  $f^+(\mathcal{G}_0) \leq f_s(\mathcal{G}_0) = \omega_s n$ . So  $R_\varepsilon(x) = O(2^{\omega_s n})\text{poly}(n, 1/\varepsilon)$ .

We also have a similar result for bipartite graphs, using the same argument.

**Lemma 57.** *Let  $G$  be a bipartite graph which is not a forest. Let  $f$  be a valid pre-potential with boundary points  $k_0, \dots, k_s$  and slices  $f_1, \dots, f_s$ , and for each  $i \in [s]$ , let  $f_i = \rho_i m + \sigma_i n$ . For all  $i \in [s]$  and  $d \geq \omega(\lambda)$ , let*

$$T_{i,d} = \begin{cases} \tau(f_i(d, 1), f_i(D'_2(k_{i-1}), 1 + d)) & \text{if } \sigma_i \geq 0 \\ \tau(f_i(d, 1), f_i(D'_2(k_{i-1}), \lfloor \frac{d + D'_2(k_{i-1}) - 1}{2} \rfloor)) & \text{otherwise,} \end{cases}$$

and

$$T = \max(\{T_{i,d} | i \in [s], \max\{\omega(\lambda), \lfloor k_{i-1} \rfloor + 1\} \leq d \leq 10\} \cup \{\tau(\sigma_s, 12\sigma_s)\}).$$

If  $T \leq 2$ , then  $\mathbf{Count}_\lambda(\mathbf{TR}(G, \boldsymbol{\lambda}, \mathbf{1}, 1), \varepsilon)$  has running time  $O(2^{\sigma_s n})\text{poly}(n, 1/\varepsilon)$ .

---

## Chapter 5

# Pre-potentials for values of Lambda

We can now introduce pre-potentials for values of  $\lambda > 0$ . We introduce the following lemma, which sets out an alternative set of conditions that, if met, define a pre-potential function.

**Lemma 58.** *Let  $f$  be a pre-potential with boundary points  $-1 = k_0, \dots, k_s = \infty$  and slices  $f_1, \dots, f_s$ , and for each  $i \in [s]$ , let  $f_i(m, n) = \rho_i m + \sigma_i n$ . Then  $f$  is a valid pre-potential of all of the following hold.*

1.  $\rho_s = 0$  and  $\sigma_s > 0$ ,
2. For all  $i \in [s-1]$ ,  $\rho_i + \sigma_i \geq 0$ ,
3. For all  $i \in [s-1]$ , at least one of  $\rho_i$  and  $\sigma_i$  is non-zero,
4. For all  $i \in [s-1]$ ,  $\rho_i \geq \rho_{i+1}$  and  $\sigma_i \leq \sigma_{i+1}$ ,
5. Every integer in  $[6, k_{s-1}]$  is a boundary point,
6. For all  $i \in [s-1]$ ,  $\rho_i = \rho_{i+1} + 2(\sigma_{i+1} - \sigma_i)/k_i$ .

*Sketch Proof.* We show that each property in Definition 53 is implied by the properties above.

- Property (P1) follows from properties (i), (ii), (iii) and (iv).
- Property (P2) follows from property (i).
- Property (P3) is property (v).

Then for property (P4), let  $i \in [s]$ , and let  $m$  be a non-negative integer and  $n$  be a positive integer such that  $k_{i-1} < 2m/n \leq k_i$ . Let  $j > i$ , then we need to show that  $f_i(m, n) \leq f_j(m, n)$ . By property (vi),

$$\rho_i - \rho_j = \sum_{\ell=i}^{j-1} (\rho_\ell - \rho_{\ell+1}) \leq \frac{2}{k_\ell} \sum_{\ell=i}^{j-1} (\sigma_{\ell+1} - \sigma_\ell) = \frac{2}{k_\ell} (\sigma_j - \sigma_i).$$

Then by property (iv),

$$f_j(m, n) - f_i(m, n) = -(\rho_i - \rho_j)m + (\sigma_j - \sigma_i)n \geq (\sigma_j - \sigma_i)(n - \frac{2}{k_\ell}m) \geq 0,$$

so  $f_i(m, n) \leq f_j(m, n)$ . A similar argument shows that  $f_j(m, n) \leq f_i(m, n)$  for  $j < i$ , so (P4) follows.

The code in Appendix C checks a candidate potential function for these properties, and if all hold, by Lemma 56, the running time of  $\text{Count}_\lambda$  is  $O(2^{\sigma_s n})\text{poly}(n, 1/\varepsilon)$ . Table 5.1 gives runtime constants for the critical values in Table 3.1.

---

$\lambda$	$\sigma$	$\sigma_b$
0.988	0.26796	0.23725
1.024	0.26796	0.23909
1.054	0.27343	0.24750
1.107	0.27635	0.24984
1.160	0.27635	0.25213
1.219	0.27931	0.25458
1.285	0.27931	0.25635
1.321	0.28345	0.25949
1.403	0.28345	0.25951
1.497	0.28708	0.26287
1.606	0.28708	0.26287
1.688	0.29333	0.27152
1.878	0.29704	0.27577
2.097	0.29704	0.27577
2.341	0.30062	0.27901
2.494	0.30062	0.27901
2.934	0.30505	0.28723
3.564	0.30505	0.28723
4.000	0.31142	0.28910

Table 5.1: Critical values for  $\lambda$  with associated running time constant  $\sigma$  for general graphs and  $\sigma_b$  for bipartite graphs.

---

## Chapter 6

# Conclusion

We now bring everything together to prove the main theorem of the paper.

**Theorem 59.** *For all  $\lambda > 0$  there is an approximation scheme for  $Z_\lambda(G)$  for any arbitrary graph.*

*Proof.* Let  $\lambda > 0$ , and  $G$  be a graph. Let  $\mathcal{G} = (G, \lambda, 1, 1)$ . The approximation scheme computes  $\mathcal{G}$  and then returns  $\text{Count}_\lambda(\mathcal{G}, \varepsilon)$ . By Lemma 49 this outputs an  $\varepsilon$ -approximation for  $Z(\mathcal{G})$ , and  $Z_\lambda(G) = Z(\mathcal{G})$ , so this is indeed an  $\varepsilon$ -approximation to  $Z_\lambda(G)$ . As we can compute  $\mathcal{G}$  from  $G$  in polynomial time, the running time of the approximation scheme is that of  $\text{Count}_\lambda$ , which by Lemma 56, is  $O(2^{\sigma_s n})\text{poly}(n, 1/\varepsilon)$  for some constant  $\sigma_s$  given by the potential function for  $\text{Count}_\lambda$ .  $\square$

One possible avenue for improvement is the program for generating decreasing functions in Appendix A. Currently, it uses an exhaustive check of all possible degrees of the neighbours of a vertex. However as  $\lambda$  approaches 0, the subcritical connective constant increases, so the maximum degree of the vertices increases. This means there are more possibilities for the degrees of the neighbours, so the code takes longer to execute. Improving this would allow finding the decreasing functions for values of  $\lambda$  closer to 0.

Another possible improvement could be a better analysis on  $\text{Count}_\lambda$  when we have  $\omega(\lambda) \geq 11$ . In this case, we could have a graph that we could approximate  $Z(\mathcal{G})$  in polynomial time, but we would instead branch on vertices of degree at least 11. A better analysis could result in a better running time for these graphs.

---

# Bibliography

- [1] Vilhelm Dahllöf, Peter Jonsson, and Magnus Wahlström. Counting models for 2SAT and 3SAT formulae. *Theor. Comput. Sci.*, 332:265–291, 02 2005.
- [2] Leslie Ann Goldberg, John Lapinskas, and David Richerby. Faster exponential-time algorithms for approximately counting independent sets. *CoRR*, abs/2005.05070, 2020.
- [3] Alistair Sinclair, Piyush Srivastava, and Yitong Yin. Spatial mixing and approximation algorithms for graphs with bounded connective constant. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, oct 2013.
- [4] Alistair Sinclair, Piyush Srivastava, Daniel Štefankovič, and Yitong Yin. Spatial mixing and the connective constant: Optimal bounds, 2014.

---

## Appendix A

# Generating code for decreasing functions

The following code is written in SageMath, and uses the linear programming capabilities to find 2-degree bounds for which there is a valid decreasing function for connective constant subcritical to  $\lambda > 0$ .

```
from math import floor, ceil
import sage.all
from sage.numerical.mip import MixedIntegerLinearProgram, MIPSolverException

def invApprox(lam):
    func = lambda x: (x**x)/((x-1)**(x+1))
    change = 1/10
    curr = 1
    while change > 1/10000:
        if func(curr + change) > lam:
            curr += change
        else:
            change = change / 10
    return round(curr,3)

def populateChildren(kappaCrit, currDeg, value, maxDeg, boundDeg, bound2Deg, children, p, prog,
    ↪ myVars):
    for i in range(0,value+1): # iterate over each value available
        children[currDeg-2] = i
        if currDeg == 2: #if degrees have all been initialised
            d = 1 + sum(children) #set the degree of v
            d0 = min(d,boundDeg + 1) # psi function only defined up to boundDeg+1
            p0 = min(p,boundDeg + 1)
            if d >= 2 and d <= maxDeg and (d <= boundDeg
                ↪ or p + sum(list(map((lambda i : children[i-2]*i),
                    ↪ list(range(2,maxDeg+1))))) <=
                ↪ bound2Deg):
                prog.add_constraint(sum([children[i-2]*myVars[min(i,boundDeg + 1)][d0] for i in
                    ↪ range(2,maxDeg+1)]))
                    ↪ <= kappaCrit * myVars[d0][p0])
        else:
            populateChildren(kappaCrit, currDeg-1, value-i, maxDeg, boundDeg, bound2Deg,
                ↪ children, p, prog, myVars)

def generateProg(kappaCrit, maxDeg, boundDeg, bound2Deg):
    prog = MixedIntegerLinearProgram(maximization=True, solver = "GLPK") #initialise linear
    ↪ program
    myVars = [prog.new_variable(integer=True, nonnegative=True) for _ in range(0,boundDeg + 2)]
    ↪ #initialise all vars
    for p in range(2,maxDeg+1): #iterate over possible degrees of the parent p
        children = [0 for _ in range(2,maxDeg+1)] #initialise array of number of children with
        ↪ degree i
```

---

```

        populateChildren(kappaCrit,maxDeg, maxDeg-1, maxDeg, boundDeg, bound2Deg, children, p,
        ↪ prog, myVars)
    prog.add_constraint(myVars[boundDeg+1][2]==1000) #normalise values
    for i in range(2, boundDeg + 1):
        for j in range(2,boundDeg +1):
            prog.add_constraint(myVars[i][j]<=myVars[i+1][j]) #force less than/equal to
    return (prog,myVars)

def findDec(lambdaVal):
    kappaCrit = invApprox(lambdaVal) #connective constant subcritical for lambda
    boundDeg = ceil(kappaCrit) #any degree under ceil(kappaCrit) has subcritical connective
    ↪ constant
    bound2Deg = int((20/lambdaVal) + 4)
    maxDeg = min(bound2Deg//2, floor((kappaCrit ** 2) + 1)) #maxDegree less than both
    found = False
    highestValidDeg = None
    lowestInvalidDeg = None
    validProg = None
    while not found:
        prog, myVars = generateProg(kappaCrit, maxDeg, boundDeg, bound2Deg)
        try:
            prog.solve()
        except MIPSolverException:
            lowestInvalidDeg = bound2Deg
            bound2Deg -= 1
        else:
            highestValidDeg = bound2Deg
            bound2Deg += 1
        maxDeg = min(bound2Deg//2,floor((kappaCrit ** 2) + 1))
        found = (highestValidDeg != None) and (lowestInvalidDeg != None)

    bound2Deg = highestValidDeg
    maxDeg = min(bound2Deg//2,floor((kappaCrit ** 2) + 1))
    prog, mymyVars = generateProg(kappaCrit, maxDeg, boundDeg, bound2Deg)
    valid = True
    i = boundDeg
    prog.solve()
    while valid:
        # while prog has a solution, keep forcing values of myVars[boundDeg+1] to be equal until no
        ↪ solutions remain
        try:
            validProg.solve()
        except:
            valid = False
        else:
            y= validProg.get_values(mymyVars)
            validProg.add_constraint(mymyVars[boundDeg+1][i] == mymyVars[boundDeg+1][i+1])
            i -= 1
            if i == 1: valid = False
    psi = prog.get_values(mymyVars)
    filename = "decreasing_func_for_" + str(float(lambdaVal)) + ".txt"
    file = open(filename, 'w', newline='')
    file.write(str(lambdaVal) + ',')
    file.write(str(kappaCrit) + ',')
    file.write(str(boundDeg) + ',')
    file.write(str(bound2Deg) + '\n')
    for i in range(2,boundDeg + 2):
        for j in range(2, boundDeg + 2):
            file.write(str(psi[i][j]))
            if j <= boundDeg: file.write(',')
    file.write('\n')

```

---



---

`findDec(2.0)`

---

## Appendix B

# Verification for decreasing functions

This python program takes a candidate  $\kappa$ -decreasing function as input, and checks that for all valid sets of degrees of the neighbours of a vertex, that the function has the required bounds. If it does, then the decreasing function is valid.

```
from math import floor

def populateChildren(kappa, psi, boundDeg, bound2Deg, maxDeg, currDeg, value,
    ↪ twoValue, children, p):
    for i in range(0, min(value+1, twoValue//currDeg) + 1):
        children[currDeg-2] = i
        if currDeg > 2:
            newValue = value - i
            if not populateChildren(kappa, psi, boundDeg, bound2Deg, maxDeg,
    ↪ currDeg-1, newValue, twoValue - (i*currDeg), children, p):
                return False
        d = 1 + sum(children)
        d0 = min(d, boundDeg+1)
        p0 = min(p, boundDeg+1)
        if d >= 2 and d <= maxDeg and (d <= boundDeg or (p + sum(list(map(lambda x :
    ↪ children[x-2]*x, list(range(2, maxDeg+1))))) <= bound2Deg)):
            if sum(list(map(lambda x :
    ↪ children[x-2]*psi[min(x, boundDeg+1)] [d0], list(range(2, maxDeg+1))))) >
    ↪ kappa * psi[d0] [p0]:
                return False
    return True

def validFile(filename):
    file = open(filename, 'r')
    params = file.readline().strip().split(',')
    lambdaVal = float(params[0])
    kappa = float(params[1])
    boundDeg = int(params[2])
    bound2Deg = int(params[3])
    psi = [None for _ in range(2, boundDeg+4)]
    for i in range(2, boundDeg+2):
        psi[i] = [None for _ in range(2, boundDeg+4)]
        row = file.readline().strip().split(',')
        for j in range(2, boundDeg+2):
            psi[i][j] = float(row[j-2])
    maxDeg = bound2Deg // 2
    print(kappa, psi, boundDeg, bound2Deg, maxDeg, maxDeg, maxDeg-1)
    for p in range(2, maxDeg + 1):
```

---

---

```
children = [0 for _ in range(2,maxDeg+1)]
if not populateChildren(kappa, psi, boundDeg, bound2Deg, maxDeg, maxDeg,
↳ maxDeg-1, bound2Deg-p, children, p):
    return False
return True

print(validFile("decreasing_func_for_2.0.txt"))
```

---

## Appendix C

# Verification for potential functions

The following python program takes a candidate potential function as input, and checks it for the conditions required in Lemma 58. If all are satisfied, the pre-potential is valid, and we can take  $\sigma_s$  as the constant in the running time.

```
from math import floor
from math import ceil
from fractions import Fraction

def isSuitable(k,z):
    K = floor(k) + 1
    if z >= K**2: return True
    for d in range(K,z//2 + 1):
        for s in range(0,d):
            if (z >= (K * s) + (2 * (d - s))) and (z <= (K * s) + ((K - 1)*(d - s))):
                q = floor((z - (K * s))/(d - s))
                d1 = (z - (K * s)) % (d - s)
                d0 = d - s - d1
                if (d + d0 + d1)/(1 + (d0/q) + (d1/(q + 1))) > k: return True
    return False

def Dtwo(k):
    K = floor(k) + 1
    for z in range(2*K, K**2):
        if isSuitable(k,z): return z
    return K**2

def validate(fileName):
    file = open(fileName)
    lines = file.read().split()
    isBipartite = lines[0] == 'Bipartite'
    lines = [line.split(',') for line in lines]
    d_0 = int(lines[1][0])
    S_0 = int(lines[1][1])
    rho = [Fraction(n) for n in lines[2]]
    sigma = [Fraction(n) for n in lines[3]]
    k = [Fraction(n) for n in lines[4]]
    s = len(rho)
    for i in range(0,s-2):
        if k[i] >= k[i + 1]:
            print("ks not ascending")
            return
    if rho[s-1] != 0 or sigma[s-1] <= 0:
        print("Fails Lemma 58(i): rho[s] not zero or sigma[s] non-positive")
        return
```

---

```

for i in range(0, s - 1):
    if rho[i] + sigma[i] < 0:
        print("Fails Lemma 58(ii): rho + sigma < 0")
        return
    if rho[i] == 0 and sigma[i] == 0:
        print("Fails Lemma 58(iii): rho and sigma 0")
        return
    if rho[i] < rho[i+1] or sigma[i] > sigma[i+1]:
        print("Fails Lemma 58(iv): rho increasing/sigma decreasing")
        return
for i in range(6, floor(k[s-2]) + 1):
    if i not in k:
        print("Lemma 58(v) at ", i)
        return
for i in range(0,s):
    if i == 0 or k[i - 1] <= d_0: D2p = S_0
    else: D2p = max(2 * k[i - 1], Dtwo(k[i - 1]))
    if i == 0: startRange = d_0
    else: startRange = max(6, floor(k[i-1]) + 1)
    for d in range(startRange, 11):
        exp1 = (d * rho[i]) + sigma[i]
        if isBipartite:
            if sigma[i] >= 0: exp2 = D2p * rho[i] + ((1 + d) * sigma[i])
            else: exp2 = D2p * rho[i] + (floor((d + D2p - 1) / 2) * sigma[i])
            else: exp2 = ceil((d + D2p + 3)/2) * rho[i] + ((1 + d) * sigma[i])
        if 2**(-exp1) + 2**(-exp2) > 1:
            print("T > 2 in Lemma 56")
            return
        if 2**(-sigma[s-1]) + 2**(-12*sigma[s-1]) > 1:
            print("T > 2 in Lemma 56")
            return
print("Valid Potential Function, running time O(" +
    ↪ str(float(sigma[len(sigma)-1])) + " * n)poly(n,1/eps)")

validate('potential-4-14.csv')
validate('potential-4-14-bipartite.csv')

```

---

---

## Appendix D

# Pre-potential functions

The following tables detail pre-potential functions for  $\lambda = 2$ . Table D.1 is a pre-potential function on general graphs, and Table D.2 is the same for bipartite graphs.

Boundary points $k_0, \dots, k_s$	Edge weights $\rho_1, \dots, \rho_s$	Vertex weights $\sigma_1, \dots, \sigma_s$
$k_0 = -1$		
$k_1 = 2.6$	$\rho_1 = 0.23142$	$\sigma_1 = -0.23142$
$k_2 = 3.75$	$\rho_2 = 0.23142$	$\sigma_2 = -0.23142$
$k_3 = 4$	$\rho_3 = 0.19484$	$\sigma_3 = -0.16283$
$k_4 = 4.13794$	$\rho_4 = 0.13493$	$\sigma_4 = -0.04301$
$k_5 = 4.44445$	$\rho_5 = 0.11625$	$\sigma_5 = -0.00437$
$k_6 = 4.57143$	$\rho_6 = 0.10092$	$\sigma_6 = 0.0297$
$k_7 = 4.8$	$\rho_7 = 0.08914$	$\sigma_7 = 0.05663$
$k_8 = 5$	$\rho_8 = 0.07928$	$\sigma_8 = 0.0803$
$k_9 = 5.10639$	$\rho_9 = 0.04188$	$\sigma_9 = 0.17379$
$k_{10} = 5.33334$	$\rho_{10} = 0.0382$	$\sigma_{10} = 0.1832$
$k_{11} = 5.5$	$\rho_{11} = 0.0349$	$\sigma_{11} = 0.19201$
$k_{12} = 5.625$	$\rho_{12} = 0.03199$	$\sigma_{12} = 0.19999$
$k_{13} = 5.83334$	$\rho_{13} = 0.02946$	$\sigma_{13} = 0.2071$
$k_{14} = 6$	$\rho_{14} = 0.02716$	$\sigma_{14} = 0.21381$
$k_{15} = 6.08696$	$\rho_{15} = 0.00419$	$\sigma_{15} = 0.28275$
$k_{16} = 6.26866$	$\rho_{16} = 0.00394$	$\sigma_{16} = 0.28349$
$k_{17} = 6.46154$	$\rho_{17} = 0.00371$	$\sigma_{17} = 0.28422$
$k_{18} = 6.54546$	$\rho_{18} = 0.00349$	$\sigma_{18} = 0.28492$
$k_{19} = 6.66667$	$\rho_{19} = 0.00329$	$\sigma_{19} = 0.28558$
$k_{20} = 6.85715$	$\rho_{20} = 0.00311$	$\sigma_{20} = 0.2862$
$k_{21} = 7$	$\rho_{21} = 0.00293$	$\sigma_{21} = 0.2868$
$k_{22} = \infty$	$\rho_{22} = 0$	$\sigma_{22} = 0.29704$

Table D.1: A pre-potential function on general graphs for  $\lambda = 2$ .

Boundary points $k_0, \dots, k_s$	Edge weights $\rho_1, \dots, \rho_s$	Vertex weights $\sigma_1, \dots, \sigma_s$
$k_0 = -1$		
$k_1 = 2.45$	$\rho_1 = 0.23142$	$\sigma_1 = -0.23142$
$k_2 = 3.75$	$\rho_2 = 0.23142$	$\sigma_2 = -0.23142$
$k_3 = 3.85$	$\rho_3 = 0.15776$	$\sigma_3 = -0.09332$
$k_4 = 4$	$\rho_4 = 0.15776$	$\sigma_4 = -0.09332$
$k_5 = 4.13794$	$\rho_5 = 0.0839$	$\sigma_5 = 0.05442$
$k_6 = 4.28572$	$\rho_6 = 0.07803$	$\sigma_6 = 0.06657$
$k_7 = 4.44445$	$\rho_7 = 0.07281$	$\sigma_7 = 0.07774$
$k_8 = 4.5$	$\rho_8 = 0.06813$	$\sigma_8 = 0.08815$
$k_9 = 4.57143$	$\rho_9 = 0.06402$	$\sigma_9 = 0.0974$
$k_{10} = 4.66667$	$\rho_{10} = 0.06036$	$\sigma_{10} = 0.10576$
$k_{11} = 4.8$	$\rho_{11} = 0.05706$	$\sigma_{11} = 0.11347$
$k_{12} = 5$	$\rho_{12} = 0.05403$	$\sigma_{12} = 0.12073$
$k_{13} = 5.10639$	$\rho_{13} = 0.03468$	$\sigma_{13} = 0.16911$
$k_{14} = 5.2174$	$\rho_{14} = 0.03309$	$\sigma_{14} = 0.17318$
$k_{15} = 5.33334$	$\rho_{15} = 0.03161$	$\sigma_{15} = 0.17704$
$k_{16} = 5.45455$	$\rho_{16} = 0.03023$	$\sigma_{16} = 0.18071$
$k_{17} = 5.5$	$\rho_{17} = 0.02894$	$\sigma_{17} = 0.18423$
$k_{18} = 5.55556$	$\rho_{18} = 0.02775$	$\sigma_{18} = 0.18749$
$k_{19} = 5.625$	$\rho_{19} = 0.02665$	$\sigma_{19} = 0.19055$
$k_{20} = 5.71429$	$\rho_{20} = 0.02563$	$\sigma_{20} = 0.19344$
$k_{21} = 5.83334$	$\rho_{21} = 0.02466$	$\sigma_{21} = 0.19619$
$k_{22} = 6$	$\rho_{22} = 0.02375$	$\sigma_{22} = 0.19884$
$k_{23} = 6.08696$	$\rho_{23} = 0.01229$	$\sigma_{23} = 0.23322$
$k_{24} = 6.17648$	$\rho_{24} = 0.01189$	$\sigma_{24} = 0.23444$
$k_{25} = 6.26866$	$\rho_{25} = 0.01151$	$\sigma_{25} = 0.23562$
$k_{26} = 6.36364$	$\rho_{26} = 0.01115$	$\sigma_{26} = 0.23675$
$k_{27} = 6.46154$	$\rho_{27} = 0.0108$	$\sigma_{27} = 0.23785$
$k_{28} = 6.5$	$\rho_{28} = 0.01047$	$\sigma_{28} = 0.23892$
$k_{29} = 6.54546$	$\rho_{29} = 0.01016$	$\sigma_{29} = 0.23993$
$k_{30} = 6.6$	$\rho_{30} = 0.00987$	$\sigma_{30} = 0.2409$
$k_{31} = 6.66667$	$\rho_{31} = 0.00959$	$\sigma_{31} = 0.24183$
$k_{32} = 6.76057$	$\rho_{32} = 0.00932$	$\sigma_{32} = 0.24272$
$k_{33} = 6.85715$	$\rho_{33} = 0.00906$	$\sigma_{33} = 0.24359$
$k_{34} = 7$	$\rho_{34} = 0.00881$	$\sigma_{34} = 0.24444$
$k_{35} = 7.07369$	$\rho_{35} = 0.00118$	$\sigma_{35} = 0.27114$
$k_{36} = 7.14894$	$\rho_{36} = 0.00115$	$\sigma_{36} = 0.27124$
$k_{37} = 7.22581$	$\rho_{37} = 0.00113$	$\sigma_{37} = 0.27134$
$k_{38} = 7.30435$	$\rho_{38} = 0.0011$	$\sigma_{38} = 0.27144$
$k_{39} = 7.38462$	$\rho_{39} = 0.00107$	$\sigma_{39} = 0.27154$
$k_{40} = 7.46667$	$\rho_{40} = 0.00105$	$\sigma_{40} = 0.27163$
$k_{41} = 7.5$	$\rho_{41} = 0.00103$	$\sigma_{41} = 0.27172$
$k_{42} = 7.53847$	$\rho_{42} = 0.001$	$\sigma_{42} = 0.2718$
$k_{43} = 7.58334$	$\rho_{43} = 0.00098$	$\sigma_{43} = 0.27189$
$k_{44} = 7.63637$	$\rho_{44} = 0.00096$	$\sigma_{44} = 0.27197$
$k_{45} = 7.71429$	$\rho_{45} = 0.00094$	$\sigma_{45} = 0.27205$
$k_{46} = 7.79382$	$\rho_{46} = 0.00092$	$\sigma_{46} = 0.27212$
$k_{47} = 7.875$	$\rho_{47} = 0.0009$	$\sigma_{47} = 0.2722$
$k_{48} = 8$	$\rho_{48} = 0.00088$	$\sigma_{48} = 0.27227$
$k_{49} = \infty$	$\rho_{49} = 0$	$\sigma_{49} = 0.27577$

Table D.2: A pre-potential function on bipartite graphs for  $\lambda = 2$ .