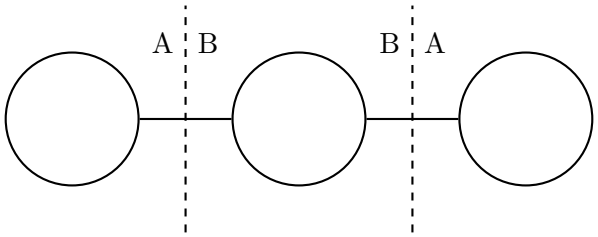
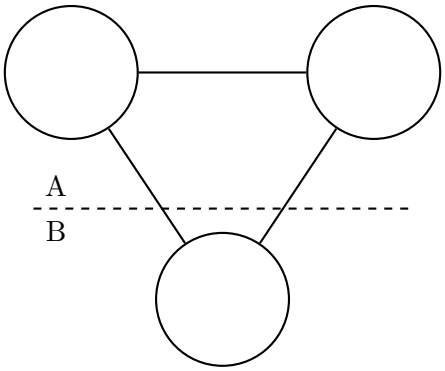
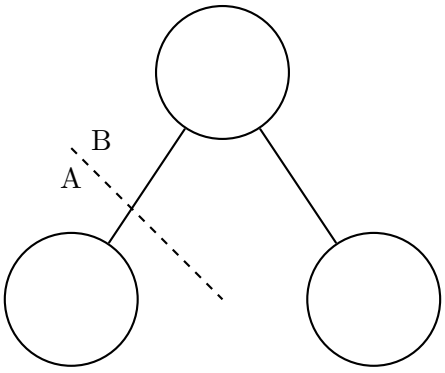
Universiteit Utrecht

Computation of Normalized Cuts of Graphs







Master Thesis
Suzanne Vincken
ICA-4273338

Under supervision of:
Dr. E.J. van Leeuwen,
Prof. dr. H.L. Bodlaender

Graduate School of Natural Sciences
Computing Science

05-07-2021

Abstract

The Normalized Cut problem was introduced by Shi and Malik in 1997 [25, 26]. In a graph, a normalized cut is a partition of the vertices into two sets, where we minimize the cut value over the volume of the separated vertex sets, in an attempt to find a balanced cut. By minimizing the normalized cut, the association between the vertices within the two vertex sets is maximized and the disassociation between the vertices in the two vertex sets is minimized [26]. Next to splitting the graph into two pieces, we can also split the graph in k pieces (a k-cut). There are two common ways to calculate the normalized cut value of a k-cut: we can either minimize the total sum of all the cuts (Normalized k-Cut) or we can minimize the maximum value of all the cuts (Max Normalized k-Cut). These forms of cutting a graph are useful in several fields, for instance, oceanography, discrete mathematics and theoretical computing science. In oceanography the Normalized Cut is widely used. Shi and Malik presented an algorithm using eigenvectors and they proved the Normalized Cut problem to be NP-complete. However, there is a distinct lack of further results on the hardness and fixed parameter tractability of this problem, which motivates our main research question of this thesis: *What is the computational complexity of the* Normalized Cut *problem?*

In this thesis we prove the Normalized Cut to be NP-complete for the vertex cover number equal to 2. We prove the Normalized k-Cut to be NP-complete for the vertex cover number equal to 3 and 4 and $k = 3$ or $k = 4$ respectively. Furthermore, we prove the Max Normalized k-Cut to be NP-complete when k equals the vertex cover number. These hardness results follow from a reduction from the Partition problem. The reductions from Partition hint that the weights of the edges are of influence on the difficulty of these problems.

For the Normalized Cut, Max Normalized 2-Cut, Normalized k-Cut and the Max Normalized k-Cut problems we present algorithms parameterized by treewidth or the vertex cover number. It stands out that we have the total sum of the edge-weights as extra parameter in the running times of these algorithms. For the Unweighted Normalized k-Cut and the Unweighted Max Normalized k-Cut problems we present algorithms parameterized by treewidth or the vertex cover number in which the total number of edges is part of the running time.

Contents

1 Introduction **1**

2 Preliminaries **2**

3 Literature Overview **7**
3.1 Normalized Cut and Conductance 7
3.2 Sparsest Cut 7
3.3 k-Small-Set Expansion 7
3.4 Density Cut 7

4 Two-Cuts **12**
4.1 Hardness Results 12
4.1.1 Normalized Cut 12
4.1.2 Max Normalized 2-Cut 14
4.2 Vertex Cover 16
4.2.1 Normalized Cut 16
4.2.2 Max Normalized 2-Cut 18
4.3 Treewidth 19
4.3.1 Normalized Cut 20
4.3.2 Max Normalized 2-Cut 22

5 Unweighted k-Cuts **22**
5.1 Treewidth 22
5.1.1 Unweighted Normalized k-Cut 23
5.1.2 Unweighted Max Normalized k-Cut 25
5.2 Vertex Cover 25
5.2.1 Unweighted Max Normalized k-Cut 26
5.2.2 Unweighted Normalized k-Cut 27

6 k-Cuts **28**
6.1 Hardness Results 28
6.1.1 Normalized 3-Cut 28
6.1.2 Normalized 4-Cut 31
6.1.3 Normalized k-Cut 33
6.1.4 Max Normalized k-Cut 34
6.2 Treewidth 35
6.2.1 Normalized k-Cut 35
6.2.2 Max Normalized k-Cut 35
6.3 Vertex Cover 36
6.3.1 Max Normalized k-Cut 36
6.3.2 Normalized k-Cut 38

7 Conclusion **38**

References **40**

A Calculations of the counter example **I**

1 Introduction

Several applications use the notion of grouping. Take, for instance, image segmentation. In general, image segmentation is used for perceptual grouping and organization of low-level image features [27]. For example, in the field of oceanography image segmentation is used for synthetic aperture radar images to find the borders of land and sea [10].

We can find another example of grouping in the field of oceanography. Part of the research in this field is about the dynamics of oceans. By doing measurements of the flows of the water, models and simulations of the ocean flows can be made. For some visualizations, see the website of the National Oceanic and Atmospheric Administration and Geophysical Fluid Dynamics Laboratory [19] or the YouTube channel of NASA Scientific Visualization Studio [8]. Furthermore, grouping is used in Lagrangian ocean analysis. In Lagrangian ocean analysis, ocean flows are simulated in ocean circulation models and the output of these models is analysed [28]. This can be used to represent transport between different regions of a fluid domain [24] or to structure marine populations and design marine protected areas [23].

The general idea of graph partitioning is not only useful for oceanography, but also for other fields. The partitioning of graphs is also researched in the fields of discrete mathematics and theoretical computing science [7]. We are interested in the results about graph partitioning in the field of computing science, which may have impactful consequences for the fields mentioned.

There are many ways to partition a graph. One example is to take the *minimum cut*. In a weighted graph the value of a cut is the sum of the weights of the edges in the cut. Since the cut value is the sum of weights, the minimum cut is likely to find a cut that simply cuts off one single vertex from a graph instead of finding a cut that splits this graph in two more equally divided parts [26]. To avoid this problem, we can use one of the many other ways to cut a graph. For instance, the Sparsest Cut problem is a well studied problem where we divide the value of the cut by the number of vertices in the smallest of the two vertex sets. It follows that cutting off one vertex gives us quite a high sparsest cut value, since we only divide by 1. Therefore, cutting off one vertex will probably not be a minimum sparsest cut. Another example of a way to cut a graph is the Conductance. Instead of dividing by the number of vertices in the smallest vertex set, we divide by the volume of the smallest vertex set. The volume is the sum of the weights of all the edges in the vertex set, including the edges in the cut. Another problem that looks similar to the Conductance problem is the Normalized Cut problem. For the calculation of the normalized cut, we do not take the volume of one of the vertex sets, but of both the vertex sets. Now, the volume of the whole graph is taken into account.

The problem that we are most interested in is the Normalized Cut problem, which is introduced by Shi and Malik in 1997 [25, 26]. For the formal definition of the Normalized Cut problem, see Section 2. We will now give an idea of the Normalized Cut. For a weighted graph $G = (V, E)$, let $S, \bar{S} \subset V$ be a partition of V. For a cut $\{S, \bar{S}\}$ we denote the value by $\text{cut}\left(S, \bar{S}\right)$. The volume $\text{vol}\,(S, V)$ equals the sum of all the edges from a vertex in S to a vertex in V. Then the *normalized cut* of a graph G is defined as the minimum over all cuts:

$$\text{Ncut}\left(S, \bar{S}\right) = \frac{\text{cut}\left(S, \bar{S}\right)}{\text{vol}\,(S, V)} + \frac{\text{cut}\left(S, \bar{S}\right)}{\text{vol}\left(\bar{S}, V\right)}. \tag{1}$$

The beauty of the formulation of this problem is that it optimizes the partition of the graph in the following way. By minimizing the normalized cut as in (1) the association between the vertices within the two vertex sets is maximized and the disassociation between

the vertices in the two vertex sets is minimized [26]. For problems like finding a graph partitioning of a flow graph of the ocean the Normalized Cut problem can be very useful [29, 2, 13, 20]. Furthermore, the popularity of the results in the paper "Normalized Cuts and Image Segmentation" of Shi and Malik [26] indicates that this is a very useful idea. Unfortunately, there is not yet much known about the Normalized Cut problem. Shi and Malik [26] showed that this problem is NP-complete and they gave a recursive algorithm for normalized k-cut on image segmentation using eigenvectors, where they cut the image into k different parts. To the best of our knowledge, there are only two NP-completeness results and no further algorithms for the Normalized Cut problem. See Section 3 for an overview of all the results.

Next to splitting the graph into two pieces, we can also split the graph in k pieces (a k-cut). There are two common ways to calculate the normalized cut of a k-cut: we can either minimize the total sum of all the cuts (Normalized k-Cut) or we can minimize the maximum value of all the cuts (Max Normalized k-Cut). For the formal definitions see Section 2. In the overview in Section 3 we find that only four algorithms are known, to solve these two variants, which are originally designed for the vertex-weighted variant of the same problem. No complexity results from the literature are known to us about the two k-cut variants of the Normalized Cut problem.

Since the paper of Shi and Malik [26] is cited so many times, we would expect more results about this problem. Moreover, the Sparsest Cut problem, a similar looking cut problem, does have many complexity results. We are particularly interested in NP-completeness and Fixed Parameter Tractability (FPT) results. For FPT we ask ourselves if there is an algorithm possible in $f(p) \cdot n^{O(1)}$ time, where p is some small parameter of the input graph. Since the Sparsest Cut problem is a similar looking problem we asked ourselves if those complexity and FPT results also hold for the Normalized Cut problem. This motivates our main research question of this thesis: *What is the computational complexity of the* Normalized Cut *problem?*

In this thesis we prove the Normalized Cut to be NP-complete for $\tau = 2$, where τ represents the vertex cover number. Furthermore, we prove the Normalized k-Cut to be NP-complete for $k = \tau = 3$ and $k = \tau = 4$. Also, the Max variant is NP-complete for any fixed $k = \tau$. For the Normalized Cut and both the k-Cut variants we give algorithms parameterized by treewidth or the vertex cover number. For both the k-Cut variants we give algorithms for the unweighted variant. It stands out that for the weighted variant we have the total sum of the edge-weights as extra parameter in the running times of these algorithms.

In Section 2 we first give the definitions of the Normalized Cut problem and other cut problems. The overview of results of these cut problems can be found in Section 3. Thereafter, in Sections 4, 5 and 6 we give our results for the (Max) Normalized (2-)Cut, the Unweighted (Max) Normalized k-Cut and the (Max) Normalized k-Cut respectively. The majority of these results are based on the results for the Sparsest Cut problem, given by Javadi and Nikabadi [15].

2 Preliminaries

When we talk about graph partitioning problems, we have the 2-cut variant and the k-cut variant. We will use a different notation for these two variants to improve the readability. Let $G = (V, E)$ be an undirected graph with edge weights where $n = |V|$ and $m = |E|$. An edge $(i, j) \in E$ between vertices i and j has weight w_{ij}. In this thesis we assume that we have integer edge weights. Note that in an unweighted graph all the weights are equal to 1.

We use the notation $w(A, B)$ to sum all the edge weights of the edges with one endpoint in vertex-set A and one endpoint in vertex-set B. So, we get:

$$w(A, B) = \sum_{\substack{(i,j)\in E,\\ i\in A,\ j\in B}} w_{ij}.$$

For ease of exposition we write $w(v, B) = w(\{v\}, B)$. Note that in $w(A, A)$ we count all the edges twice.

When we talk about a 2-cut or k-cut, we mean that we cut the graph into two or k pieces respectively. We use the following notation:

- 2-cut: let $\left\{S, \bar{S}\right\}$ define the cut, where $S, \bar{S} \subset V$ and $\bar{S} = V \setminus S$ such that $S \neq \emptyset$ and $S \neq V$.
- k-cut: let $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ define the cuts, where $S_i \subset V$ for all $1 \leq i \leq k$ such that
  - $S_i \neq \emptyset$ for all $1 \leq i \leq k$,
  - $S_i \cap S_j = \emptyset$ for all $1 \leq i, j \leq k$ and $i \neq j$, and
  - $\bigcup_{1\leq i\leq k} S_i = V$.

We use the notation $\bar{S}_j = V \setminus S_j$. In the following definitions, let S and $\mathcal{S}$ define a 2-cut and a k-cut, respectively.

In the calculation of the normalized cut we will use the functions cut and vol to calculate the value of the cut and the volume, respectively. Let $K, L \subset V$ such that $K \cap L = \emptyset$. Then the cut value of the edges between K and L is calculated as follows:

$$\operatorname{cut}(K, L) = w(K, L).$$

For the volume, let $P \subseteq Q \subseteq V$. Then the volume of P can be calculated as follows:

$$\operatorname{vol}(P, Q) = w(P, Q).$$

Note that in the volume the edges inside P are counted twice and the edges with one side in P and one side in Q are counted once.

Now, we can formally define the Normalized Cut problem where we make a 2-cut.

Definition 2.1 (Normalized Cut [26])**.** The *normalized cut value* of a cut $\left\{S, \bar{S}\right\}$ in a graph is defined as:

$$\operatorname{Ncut}\left(S, \bar{S}\right) = \frac{\operatorname{cut}\left(S, \bar{S}\right)}{\operatorname{vol}(S, V)} + \frac{\operatorname{cut}\left(S, \bar{S}\right)}{\operatorname{vol}\left(\bar{S}, V\right)}.$$

Now we define the *normalized cut of a graph* as:

$$\operatorname{Ncut}(G) = \min_{S\subsetneq V, S\neq\emptyset} \operatorname{Ncut}\left(S, \bar{S}\right).$$

The problem of computing the normalized cut of a graph is called Normalized Cut. ⊠

For the k-cut we define the Normalized k-Cut problem below. In a k-cut we make $k-1$ cuts simultaneously. We observe that this is not necessarily the same as making $k-1$ 2-cuts recursively. We calculate the normalized k-cut either by minimizing the sum over all the cuts or by minimizing the maximum of all the cuts.

Definition 2.2 (Normalized k-Cut [12, 22, 30])**.** The *normalized k-cut value* of a k-cut $\mathcal{S} = \{S_1, \ldots, S_k\}$ is defined as:

$$\mathrm{Ncut}_k(\mathcal{S}) = \sum_{1 \leq i \leq k} \frac{\mathrm{cut}\left(S_i, \bar{S}_i\right)}{\mathrm{vol}\left(S_i, V\right)}.$$

Now we define the *normalized k-cut of a graph* as:

$$\mathrm{Ncut}_k(G) = \min_{\mathcal{S}=\{S_1,\ldots,S_k\}} \mathrm{Ncut}_k(\mathcal{S}).$$

The problem of computing the normalized k-cut of a graph is called Normalized k-Cut. ⊠

Definition 2.3 (Max Normalized k-Cut [9])**.** The *max normalized k-cut value* of a k-cut $\mathcal{S} = \{S_1, \ldots, S_k\}$ is defined as:

$$\mathrm{Ncut}_k^{\max}(\mathcal{S}) = \max_{1 \leq i \leq k} \frac{\mathrm{cut}\left(S_i, \bar{S}_i\right)}{\mathrm{vol}\left(S_i, V\right)}.$$

Now we define the *max normalized k-cut of a graph* as:

$$\mathrm{Ncut}_k^{\max}(G) = \min_{\mathcal{S}=\{S_1,\ldots,S_k\}} \mathrm{Ncut}_k^{\max}(\mathcal{S}).$$

The problem of computing the max normalized k-cut of a graph is called Max Normalized k-Cut. ⊠

In the calculation of the normalized cut, the volumes of both vertex sets are taken into account. The conductance of a graph only takes into account the smallest volume of the two vertex sets. We define the Conductance problem below. Note that the Conductance is the special case of the Max Normalized k-Cut where $k = 2$.

Definition 2.4 (Conductance [17])**.** The *conductance of a cut* $\left\{S, \bar{S}\right\}$ is defined as:

$$\varphi\left(S, \bar{S}\right) = \frac{\mathrm{cut}\left(S, \bar{S}\right)}{\min\left(\mathrm{vol}\left(S, V\right), \mathrm{vol}\left(\bar{S}, V\right)\right)}.$$

The *conductance of a graph* is now defined as:

$$\varphi(G) = \min_{S \subsetneq V, S \neq \emptyset} \varphi\left(S, \bar{S}\right).$$

The problem of computing the conductance of a graph is called Conductance. ⊠

Since the normalized cut (Definition 2.1) and the conductance (Definition 2.4) only differ by one fraction, we can ask ourselves how they are related. Notice that for a given G we have $0 \leq \mathrm{Ncut}(G) \leq 2$ and $0 \leq \varphi(G) \leq 1$. Let $\left\{S, \bar{S}\right\}$ define a cut in a given graph G. It is easy to see that $\varphi\left(S, \bar{S}\right) \leq \mathrm{Ncut}\left(S, \bar{S}\right)$, since both fractions of the normalized cut are positive, and one of the fraction equals the fraction of the conductance. Furthermore, from the following equations it follows that $\varphi\left(S, \bar{S}\right) \leq \mathrm{Ncut}\left(S, \bar{S}\right) \leq 2 \cdot \varphi\left(S, \bar{S}\right)$. Say without loss of generality that $\mathrm{vol}\left(S, V\right) \leq \mathrm{vol}\left(\bar{S}, V\right)$. Then it follows that:

$$\frac{\mathrm{cut}\left(S, \bar{S}\right)}{\mathrm{vol}\left(S, V\right)} \geq \frac{\mathrm{cut}\left(S, \bar{S}\right)}{\mathrm{vol}\left(\bar{S}, V\right)}.$$

With this, we have:

$$\mathrm{Ncut}\left(S, \bar{S}\right) = \frac{\mathrm{cut}\left(S, \bar{S}\right)}{\mathrm{vol}\left(S, V\right)} + \frac{\mathrm{cut}\left(S, \bar{S}\right)}{\mathrm{vol}\left(\bar{S}, V\right)} \leq \frac{\mathrm{cut}\left(S, \bar{S}\right)}{\mathrm{vol}\left(S, V\right)} + \frac{\mathrm{cut}\left(S, \bar{S}\right)}{\mathrm{vol}\left(S, V\right)} = 2 \cdot \varphi\left(S, \bar{S}\right).$$

From this, it follows that $\varphi(G) \leq \text{Ncut}(G) \leq 2 \cdot \varphi(G)$.

It is also interesting to mention that the NORMALIZED CUT problem and the CONDUCTANCE problem do not always find the same cut. For an example, see Figure 1. For the cut $\{S, \bar{S}\} = \{\{a, b\}, \{c, d, e, f\}\}$ we have a normalized cut value of 0.223 and a conductance value of 0.200. For cut $\{S, \bar{S}\} = \{\{a, b\}, \{c, d, e, f\}\}$ we have a normalized cut value of 0.254 and a conductance value of 0.143. The first is the optimal cut for the NORMALIZED CUT problem and the second cut is optimal for the CONDUCTANCE problem. All the other cuts have a higher value. The calculations of all the possible cuts and their values can be found in Appendix A.

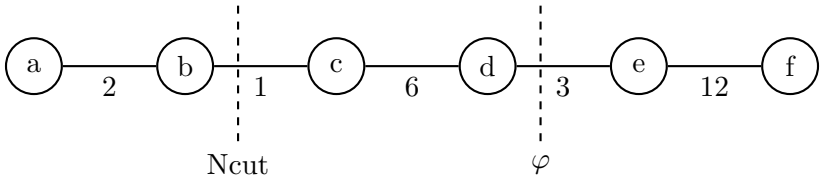


Figure 1: An example of a graph where the NORMALIZED CUT and the CONDUCTANCE do not find the same optimal cut. For the calculations, see Appendix A.

Now, we can conclude that 2φ gives us an upper bound on the value of the normalized cut and it follows that exact algorithms for the CONDUCTANCE problem do not directly give us exact algorithms for the NORMALIZED CUT problem.

As introduced in Section 1 there is also the SPARSEST CUT problem. Instead of using the volume of a vertex set, the sparsest cut uses the amount of vertices in a vertex set. For the SPARSEST CUT problem we also know a 2-cut and a k-cut variant.

Definition 2.5 (Sparsest Cut [15])**.** The *edge expansion* of a cut $\{S, \bar{S}\}$ is defined as:

$$\phi\left(S, \bar{S}\right) = \frac{\text{cut}\left(S, \bar{S}\right)}{\min\left(|S|, |\bar{S}|\right)}.$$

The *sparsest cut* of a graph is now defined as:

$$\phi(G) = \min_{S \subsetneq V, S \neq \emptyset} \phi\left(S, \bar{S}\right).$$

The problem of computing the sparsest cut of a graph is called SPARSEST CUT. ⊠

Definition 2.6 (k-Sparsest Cut [15])**.** The *k-sparsest cut* of a graph is defined as:

$$\phi_k(G) = \min_{\mathcal{S}=\{S_1,\ldots,S_k\}} \max_{1 \leq i \leq k} \phi\left(S_i, \bar{S}_i\right).$$

The problem of computing the k-sparsest cut of a graph is called k-SPARSEST CUT. ⊠

Note that for $k = 2$ these two definitions give us the same result.

A variant of the SPARSEST CUT problem, called the k-SMALL-SET EXPANSION problem, is a problem in which we make one cut in a graph, where one of the two vertex sets is of size at most k. The definition is stated below.

Definition 2.7 (k-Small-Set Expansion [15])**.** We seek for a subset $S \subset V$ of size at most k. The *k-small-set expansion* of a graph is defined as:

$$\psi_k(G) = \min_{S \subset V, |S| \leq k} \phi\left(S, \bar{S}\right),$$

where $\phi\left(S, \bar{S}\right)$ is defined in Definition 2.5.

The problem of computing the k-small-set expansion is called k-Small-Set Expansion. ⊠

Another variant of the Sparsest Cut problem, is referred to by two names. It is either called the Density Cut problem as defined by Bonsma *et al.* [7] or the Mean Sparsest Cut as defined by Javadi and Nikabadi [15]. We will use the term *density cut*. This problem is defined below.

Definition 2.8 (Density Cut [7], Mean Sparsest Cut [15])**.** The *density cut value* of a cut $\left\{S, \bar{S}\right\}$ is defined as:

$$d\left(S, \bar{S}\right) = \frac{\text{cut}\left(S, \bar{S}\right)}{|S| \cdot |\bar{S}|}.$$

The *density cut of a graph* is now defined as:

$$d\left(G\right) = \min_{S \subsetneq V, S \neq \emptyset} d\left(S, \bar{S}\right).$$

The problem of calculating the density cut of a graph is called Density Cut. ⊠

Some of the results in our literature study follow from a vertex weighted variant. In this case the edges and all the vertices have weights. We write $w(u)$ for the weight of vertex $u \in V$. Recall that the edge weights are denoted by w_{ij} for the edge between vertex i and j, for which $(i, j) \in E$ holds. To sum all the vertex weights in a set $S \subseteq V$ we write

$$w\left(S\right) = \sum_{u \in S} w(u).$$

The vertex weighted variants of both the Mean Normalized k-Cut and the Max Normalized k-Cut problems are described by Daneshgar and Javadi [9].

Definition 2.9 (Mean and Max Vertex-Weighted Normalized k-Cut [9])**.** The *mean* and *max vertex-weighted normalized k-cut* of a graph are defined as:

$$\iota_k^{\text{mean}}\left(G\right) = \min_{\mathcal{S}=\{S_1,\ldots,S_k\}} \frac{1}{k} \left(\sum_{1 \leq i \leq k} \frac{\text{cut}\left(S_i, \bar{S}_i\right)}{w\left(S_i\right)} \right)$$

and

$$\iota_k^{\max}\left(G\right) = \min_{\mathcal{S}=\{S_1,\ldots,S_k\}} \max_{1 \leq i \leq k} \frac{\text{cut}\left(S_i, \bar{S}_i\right)}{w\left(S_i\right)}.$$

The problems of computing the mean and max vertex-weighted normalized k-cut are called Mean - and Max Vertex-Weighted Normalized k-Cut. ⊠

The vertex weighted variant of the Sparsest Cut problem is also known as the Quotient Cut problem, described by Park and Phillips [21].

Definition 2.10 (Quotient Cut [21])**.** The *quotient cut* of a graph is defined as:

$$q\left(G\right) = \min_{S \subsetneq V, S \neq \emptyset} \frac{\text{cut}\left(S, \bar{S}\right)}{\min\left(w\left(S\right), w\left(\bar{S}\right)\right)}.$$

The problem of computing the quotient cut of a graph is called Quotient Cut. ⊠

When a result found in the literature study is of the vertex-weighted variant, this is stated by the result.

Remark 2.11. Note that when $w(u)$ equals the sum of its edges for all $u \in V$, the mean vertex-weighted normalized k-cut and the max vertex-weighted normalized k-cut equal the normalized k-cut and the max normalized k-cut, respectively. Likewise, the quotient cut then equals the conductance. Note that the quotient cut does not equal the sparsest cut, since we divide by the volume of S and not the amount of vertices in S.

When all the vertex weights are 1, so $w(u) = 1$ for all $u \in V$, then we are no longer dividing by volume, but by the number of vertices. So it follows that the max unweighted-vertex normalized k-cut translates now to the unweighted k-sparsest cut. We have no similar looking problem for the mean unweighted-vertex normalized k-cut. Moreover, the unweighted-vertex quotient cut translates to the sparsest cut.

3 Literature Overview

Below we will give an overview of the results in the field of Normalized Cut, Conductance, Sparsest Cut, Small-Set Expansion and Density Cut. The lists of results are complete, but not exhausting. Only results about complexity and fixed parameter tractability are taken into account.

3.1 Normalized Cut and Conductance

Table 1 and 2 show the results of the Normalized Cut and the Conductance. Daneshgar and Javadi [9] have proved some interesting theorems about the Mean- and Max Vertex-Weighted Normalized Cut problems. In Corollary 2 in [9] they state that the Max Normalized Cut problem is NP-complete for unweighted trees. Now we recall Remark 2.11 and we realize that this result fits under our definition of the k-Sparsest Cut problem.

3.2 Sparsest Cut

For the Sparsest Cut, there are more results known from existing literature. The results are shown in Table 3. Mohar [18] has shown some useful results. In the paper of Mohar [18], the problem is called *isoperimetric number* of a given graph G. However, that definition is the same as Definition 2.5 of the Sparsest Cut problem that we use.

3.3 k-Small-Set Expansion

The results for the k-Small-Set Expansion are shown in Table 4.

3.4 Density Cut

The results for the Density Cut problem are shown in Table 5.

Table 1: The overview of results in the field of the normalized cut (for the max variant see Table 2). With parameters: τ - the vertex cover number, tw - the treewidth of the input graph, and W - the total edge-weight of the graph.

Problem	Parameter	Results
Normalized Cut, Definition 2.1	-	NP-complete for regular grids [26] NP-complete for bipartite planar weighted graphs [9, 26]
	(τ, W)	NP-complete for graphs with $\tau = 2$ [Theorem 4.3] $O(2^{\tau} nW)$ time algorithm with $O(nW)$ space [Theorem 4.6]
	(tw, W)	$O\left(2^{tw} nW^2\right)$ time algorithm with $O\left(2^{tw} nW\right)$ space [Theorem 4.10]
Normalized k-Cut, Definition 2.2	τ	NP-complete for $k = \tau = 3$ and $k = \tau = 4$ [Theorems 6.2, 6.3] [Conjecture] NP-complete for $k = \tau$ [Conjecture 6.4]
	(τ, W)	$O\left(k^{\tau+1} n^{k+1} 6^k W^{3k}\right)$ time algorithm with $O\left(n^{k+1}(3W)^k\right)$ space for $k \geq 2$ [Theorem 6.9]
	(tw, W)	$O\left(k^{tw} 2^{3k} W^{4k} n\right)$ time algorithm with $O\left(k^{tw} 2^k W^{2k} n\right)$ space for $k \geq 2$ [Theorem 6.6]
Unweighted Normalized k-Cut, Definition 2.2	τ	$O\left(k^{\tau+1} n^{k+1} 6^k m^{3k}\right)$ time algorithm with $O\left(n^{k+1}(3m)^k\right)$ space for $k \geq 2$ [Theorem 5.5]
	tw	$O\left(k^{tw} 2^k m^{4k} n\right)$ time algorithm with $O\left(k^{tw} 2^k m^{2k} n\right)$ space for $k \geq 2$ [Theorem 5.1]

Table 2: The overview of results in the field of the max normalized cut and the conductance (for the results of the normalized cut, see Table 1). With parameters: τ - the vertex cover number, tw - the treewidth of the input graph, W - the total edge-weight of the graph, and ω - the total vertex-weight of the graph.

Problem	Parameter	Results
Max Normalized 2-Cut, Definition 2.3 and Conductance, Definition 2.4	τ	NP-complete for graphs with $\tau = 2$ [Theorem 4.4]
	(τ, W)	$O\left(2^{\tau} nW\right)$ time algorithm with $O\left(nW\right)$ space [Theorem 4.7]
	(tw, W)	$O\left(2^{tw} nW^2\right)$ time algorithm with $O\left(2^{tw} nW\right)$ space [Theorem 4.11]
	ω	$O\left(n^3\omega\right)$ time algorithm for planar graphs *(for vertex-weighted variant)* [21] $O\left(n^2\omega \log(n\omega)\right)$ time algorithm for planar graphs *(for vertex-weighted variant)* [21]
Max Normalized k-Cut, Definition 2.3	τ	NP-complete for $\tau = k$ [Theorem 6.5]
	(τ, W)	$O\left(k^{\tau+1} n^{k+1} (3W)^k\right)$ time algorithm with $O\left(n^{k+1} (3W)^k\right)$ space for $k \geq 2$ [Theorem 6.8]
	(tw, W)	$O\left(k^{tw} 2^{3k} W^{4k} n\right)$ time algorithm with $O\left(k^{tw} 2^k W^{2k} n\right)$ space for $k \geq 2$ [Theorem 6.7]
	-	$O\left(n^2\right)$ time algorithm for weighted trees for $k = 3$ *(for vertex-weighted variant)* [9] $O\left(n^{2k^2-6k-3}\right)$ time algorithm for weighted trees for every $k \geq 4$ *(for vertex-weighted variant)* [9]
Unweighted Max Normalized k-Cut, Definition 2.3	τ	$O\left(k^{\tau+1} n^{k+1} (3m)^k\right)$ time algorithm with $O\left(n^{k+1} (3m)^k\right)$ space for $k \geq 2$ [Theorem 5.3]
	tw	$O\left(k^{tw} 2^k m^{4k} n\right)$ time algorithm with $O\left(k^{tw} 2^k m^{2k} n\right)$ space for $k \geq 2$ [Theorem 5.2]

Table 3: The overview of results in the field of the sparsest cut. With parameters: Δ - the maximum degree, d - the degeneracy, τ - the vertex cover number, tw - the treewidth of the input graph, and ω - the total vertex weight of the graph.

Problem	Parameter	Results
Sparsest Cut, Definition 2.5 *equal to edge-weighted* 2-Sparsest Cut	-	NP-complete for (unweighted) graphs with multiple edges [18]
	τ	FPT [15]
	tw	FPT [15]
	ω	$O\left(n^3\omega\right)$ time algorithm for planar graphs *(for vertex-weighted variant)* [21] $O\left(n^2\omega\log(n\omega)\right)$ time algorithm for planar graphs *(for vertex-weighted variant)* [21]
Unweighted Sparsest Cut, Definition 2.5 *equal to* Unweighted 2-Sparsest Cut	-	NP-complete for graphs with multiple edges [18] $O\left(n\right)$ time algorithm for arbitrary trees [18]
k-Sparsest Cut, Definition 2.6	τ	NP-hard for every $k \geq 3$ and $\tau \geq 3$ [15]
	tw	NP-hard for every $k \geq 3$ and $tw \geq 3$ [15]
	Δ	NP-hard for every $k \geq 2$ and $\Delta \geq 3$ [15]
	d	NP-hard for every $k \geq 2$ and $d \geq 2$ [15]
	-	$O\left(n^2\right)$ time algorithm for weighted trees for $k = 3$ *(for vertex-weighted variant)* [9] $O\left(n^{2k^2-6k-3}\right)$ time algorithm for weighted trees for every $k \geq 4$ *(for vertex-weighted variant)* [9] $O\left(k^2n^3\right)$ time algorithm using dynamic programming on trees *(for which the k vertex sets of the cut induce connected subtrees)* [14]
Unweighted k-Sparsest Cut, Definition 2.6	(k, tw)	W[1]-hard [15]
	tw	FPT for fixed $k \geq 2$ [15]
	τ	FPT for fixed $k \geq 2$ [15]
	-	NP-complete for unweighted (simple) graphs, for $k \geq 2$ *(via vertex-weighted variant)* [9] NP-complete for unweighted trees, where k is part of the input *(via vertex-weighted variant)* [9]

Table 4: The overview of results in the field of the k-small-set expansion. With parameters: Δ - the maximum degree, τ - the vertex cover number, and tw - the treewidth of the input graph.

Problem	Parameter	Results
k-Small-Set Expansion, Definition 2.7	k	W[1]-hard even for unweighted [15]
	Δ	NP-hard for every $\Delta \geq 3$ [15]
	(k, Δ)	FPT [15]
	tw	FPT [15]
	τ	FPT [15]

Table 5: The overview of results in the field of the density cut. With parameters: tw - the treewidth of the input graph, and c - the clique width.

Problem	Parameter	Results
Density Cut, Definition 2.8	-	NP-complete [6]
		$O\left(n^2\right)$ time algorithm for outerplanar graphs [7]
	tw	$O^*\left(2^{tw}\right)$ time algorithm using dynamic programming *(The O^* notation omits factors that are polynomial in the input.)* [6, 7]
Unweighted Density Cut, Definition 2.8	-	NP-complete [6, 7]
		$O\left(n\right)$ time algorithm for unit interval graphs [7]
		$O\left(n\right)$ time algorithm for cactus graphs [7]
	c	$O\left(n^{2c}l\right)$ time algorithm where $1 \leq c \leq n$ with a c-expression of length l for constructing G [7]
		$f(c)n^{O(c)}$ time algorithm not possible unless the Exponential Time Hypothesis fails [7]

4 Two-Cuts

In this section we will prove two hardness results and give some algorithms for the Normalized Cut problem and the Max Normalized 2-Cut problem. These algorithms are parametrized by the vertex cover number, treewidth and/or the total sum of the edge-weights. The proofs and algorithms that we present are based on results of the Sparsest Cut problem by Javadi and Nikabadi [15]. Yet, the results for the Normalized Cut problem need some extra work in comparison with the Sparsest Cut problem, because the normalized cut uses volume in the denominator, where the sparsest cut uses only a counter in the denominator. So for the normalized cut, the denominator is not independent from the numerator where this is the case by the sparsest cut.

4.1 Hardness Results

We will prove the Normalized Cut problem and the Max Normalized 2-Cut problem to be NP-complete, even for vertex cover number equal to 2. To prove this, we will use the a *diamond graph*, which we define below.

Definition 4.1 (Diamond Graph)**.** The *diamond graph* is shown in Figure 2. We have vertex a and vertex b on the left and the right, respectively. In the middle we have the vertices $v_1, \ldots, v_p$ which have an edge to a with weight s_i and an edge to b with weight t_i, for $1 \leq i \leq p$. The vertices v_i have no edges to each other.

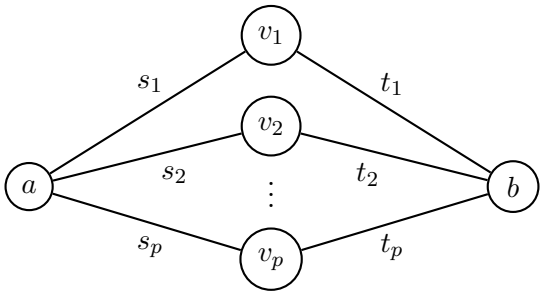


Figure 2: A diamond graph.

⊠

We will prove the Normalized Cut problem to be NP-complete in Theorem 4.3 and the Max Normalized 2-Cut problem to be NP-complete in Theorem 4.4. For these proofs we use the diamond graph and the Partition problem, which is defined below.

Definition 4.2 (Partition)**.** We are given a multiset $\mathcal{C} = \{c_1, c_2, \ldots, c_p\}$ where $c_i \in \mathbb{N}$ for $1 \leq i \leq p$. We make a partition of $\mathcal{C}$ into the multisets $\mathcal{A}$ and $\mathcal{B}$. Say that the sum of the elements in $\mathcal{C}$ is equal to $2D$. We want to answer the question:

$$\text{Is it possible to make a partition such that: } \sum_{a_i \in \mathcal{A}} a_i = \sum_{b_i \in \mathcal{B}} b_i = D \text{ ?}$$

⊠

4.1.1 Normalized Cut

In Theorem 4.3 we will prove that the Normalized Cut problem is NP-complete, even for vertex cover number equal to 2. In this proof we use another notation of the normalized cut,

see (2). Note that the term between the block brackets is constant for a given graph.

$$\begin{aligned}\operatorname{Ncut}(A,B) &= \frac{\operatorname{cut}(A,B)}{\operatorname{vol}(A,V)} + \frac{\operatorname{cut}(A,B)}{\operatorname{vol}(B,V)} \\ &= \frac{\operatorname{cut}(A,B)\operatorname{vol}(B,V)}{\operatorname{vol}(A,V)\operatorname{vol}(B,V)} + \frac{\operatorname{cut}(A,B)\operatorname{vol}(A,V)}{\operatorname{vol}(A,V)\operatorname{vol}(B,V)} \\ &= \frac{\operatorname{cut}(A,B)[\operatorname{vol}(A,V) + \operatorname{vol}(B,V)]}{\operatorname{vol}(A,V)\operatorname{vol}(B,V)}\end{aligned} \tag{2}$$

Theorem 4.3. *The* Normalized Cut *problem is NP-complete for graphs with vertex cover number equal to 2.*

Proof. For this proof we will look at the decision variant of the Normalized Cut problem. We ask ourselves if we can find a normalized cut with a value less than or equal to N.

First, note that the Normalized Cut problem is in NP. We prove the Normalized Cut problem to be NP-complete using a diamond graph as in Figure 2 and a reduction from Partition, see Definition 4.2. We make an instance of the diamond graph. Let $v_1, \ldots, v_p$ be the vertices in the middle and add two vertices a and b. Now let $s_i = c_i$ be the weight of the edge between a and v_i and let $t_i = c_i$ be the weight of the edge between v_i and b, for $1 \leq i \leq p$. It follows that the total sum of the edge weights is $W = \sum_{i=1}^{p}(s_i + t_i) = 4D$. Let $A, B \subset V$ be the two sets that will determine the cut. We choose $N = 1$.

We can translate a solution for a diamond graph to a solution for Partition. Let $\{A, B\}$ be a solution for a given diamond graph. Then, the partition we have equals $\mathcal{A} = \{s_i \mid v_i \in A\}$ and $\mathcal{B} = \{t_i \mid v_i \in B\}$.

The other way around, we can translate a solution for Partition to a solution for the corresponding diamond graph. Let $\{\mathcal{A}, \mathcal{B}\}$ be a solution for Partition. Then, the cut $\{A, B\}$ consists of the following two sets: $A = \{v_i \mid s_i = c_i \in \mathcal{A}\} \cup \{a\}$ and $B = \{v_i \mid t_i = c_i \in \mathcal{B}\} \cup \{b\}$.

If we have a yes-instance for Partition, then we can find a normalized cut smaller than or equal to N. We make a cut $\{A, B\}$ such that $a \in A$ and $b \in B$. Let $I_A \subseteq \{1, \ldots, p\}$ and let $I_B = \{1, \ldots, p\} \setminus I_A$. Now the normalized cut value is the following:

$$\operatorname{Ncut}(A,B) = \frac{\left(\sum_{i\in I_A} t_i + \sum_{i\in I_B} s_i\right) \cdot 2W}{\left(\sum_{i\in I_A}(2s_i + t_i) + \sum_{i\in I_B} s_i\right)\left(\sum_{i\in I_A} t_i + \sum_{i\in I_B}(s_i + 2t_i)\right)}.$$

Since $\sum_{i\in I_A} s_i + \sum_{i\in I_B} t_i = \frac{1}{2}W$,

$$\operatorname{Ncut}(A,B) = \frac{\frac{1}{2}W \cdot 2W}{\left(\frac{1}{2}W + \sum_{i\in I_A} 2s_i\right)\left(\frac{1}{2}W + \sum_{i\in I_B} 2t_i\right)}. \tag{3}$$

If $\{A, B\}$ forms a yes-instance for a partition input, then $\sum_{i\in I_A} t_i = \sum_{i\in I_B} s_i = D = \frac{1}{4}W$. Given this fact and (3) we get:

$$\operatorname{Ncut}(A,B) = \frac{W^2}{\left(\frac{1}{2}W + 2\cdot\frac{1}{4}W\right)\left(\frac{1}{2}W + 2\cdot\frac{1}{4}W\right)} = 1 \leq N.$$

Now we show that if we have a solution for the normalized cut at most $N = 1$, then we have a yes-instance for Partition. We have two possible cuts: either a and b are in the same vertex set or they are in different vertex sets.

First, let a and b be in the same vertex set. Say $a, b \in A$. Then it follows that B only consists of vertices v_i. Let $I \subseteq \{1, \ldots, p\}$ and define $B = V(I) = \{v_i \mid i \in I\}$. Note that $I \neq \emptyset$ must hold, otherwise we would not have a 2-cut. We will show that $\text{Ncut}\,(V(I), V \setminus V(I)) > N = 1$.

$$\begin{aligned}\text{Ncut}\,(V(I), V \setminus V(I)) &= \frac{\left(\sum_{i\in I}(s_i + t_i)\right) \cdot 2W}{\left(\sum_{i\in I}(s_i + t_i)\right)\left(\sum_{i\in I}(s_i + t_i) + 2\sum_{i\notin I}(s_i + t_i)\right)} \\ &= \frac{2W}{\sum_{i\in I}(s_i + t_i) + 2\sum_{i\notin I}(s_i + t_i)}.\end{aligned}$$

This fraction is minimized if the second term in the denominator is maximized. Let $\mu = \min_{1\leq i\leq p} s_i$. Now, we have:

$$\begin{aligned}\text{Ncut}\,(V(I), V \setminus V(I)) &= \frac{2W}{(2\mu) + (2W - 4\mu)} \\ &= \frac{2W}{2W - 2\mu} > 1 = N.\end{aligned}$$

Secondly, we consider the cut $\{A, B\}$, with $a \in A$ and $b \in B$. We take the cut value as written in (3).

$$\text{Ncut}\,(A, B) = \frac{\frac{1}{2}W \cdot 2W}{\left(\frac{1}{2}W + \sum_{i\in I_A} 2s_i\right)\left(\frac{1}{2}W + \sum_{i\in I_B} 2t_i\right)} \leq 1$$

Now let $x = \sum_{i\in I_A} 2s_i$ and $y = \sum_{i\in I_B} 2t_i$. We know that $x + y = W$.

$$\begin{aligned}&\Leftrightarrow W^2 < \left(\frac{1}{2}W + x\right)\left(\frac{1}{2}W + y\right) \\ &\Leftrightarrow W^2 < \frac{1}{4}W^2 + \frac{1}{2}W(x + y) + xy \\ &\Leftrightarrow W^2 < \frac{1}{4}W^2 + \frac{1}{2}W \cdot W + xy \\ &\Leftrightarrow \frac{1}{4}W^2 - xy < 0 \\ &\Leftrightarrow \frac{1}{4}W^2 - (W - y)y < 0 \\ &\Leftrightarrow \frac{1}{4}W^2 - Wy + y^2 < 0\end{aligned}$$

The discriminant of this function is equal to $W^2 - 4 \cdot 1 \cdot \frac{1}{4}W^2 = 0$. So the only solution to $\frac{1}{4}W^2 - Wy + y^2 = 0$ is $y = \frac{1}{2}W$. It follows that $x = y = \frac{1}{2}W$ is a solution for the normalized cut and we have a yes-instance for Partition.

Since we now have a solution for Partition if and only if we have a solution for the Normalized Cut problem smaller than or equal to N, this proof is complete. ■

4.1.2 Max Normalized 2-Cut

Below we will prove that the Max Normalized 2-Cut is NP-complete. This proof follows the same steps as the proof for the Normalized Cut problem in Section 4.1.1.

Theorem 4.4. *The* Max Normalized 2-Cut *problem is NP-complete for graphs with vertex cover number equal to 2.*

Proof. For this proof we will look at the decision variant of the MAX NORMALIZED 2-CUT problem. We ask ourselves if we can find a max normalized 2-cut with a value less than or equal to N. Note that the MAX NORMALIZED 2-CUT problem is in NP.

We prove the MAX NORMALIZED 2-CUT problem to be NP-complete using a diamond graph as in Figure 2 and using a reduction from PARTITION, see Definition 4.2. We make an instance of a diamond graph. Let $v_1, \ldots, v_p$ be the vertices in the middle and add two vertices a and b. Now let $s_i = c_i$ be the weight of the edge between a and v_i and let $t_i = c_i$ be the weight of the edge between v_i and b, for $1 \leq i \leq p$. It follows that the total sum of the edge weights is $W = \sum_{i=1}^{p} (s_i + t_i) = 4D$. Let $A, B \subset V$ define the cut. We choose $N = \frac{1}{2}$.

We can translate a solution for the MAX NORMALIZED 2-CUT problem into a solution for the PARTITION problem and the other way around. This is described in the proof of Theorem 4.3.

If we have a yes-instance for PARTITION, then we can find a solution for the MAX NORMALIZED 2-CUT problem smaller than or equal to N. We make a cut $\{A, B\}$ such that $a \in A$ and $b \in B$. Let $I_A \subseteq \{1, \ldots, p\}$ and let $I_B = \{1, \ldots, p\} \setminus I_A$. Now the max normalized 2-cut value is the following:

$$\mathrm{Ncut}_2^{\max}(A, B) = \max\left\{ \frac{\sum_{i \in I_A} t_i + \sum_{i \in I_B} s_i}{\sum_{i \in I_A} (2s_i + t_i) + \sum_{i \in I_B} s_i}, \ \frac{\sum_{i \in I_A} t_i + \sum_{i \in I_B} s_i}{\sum_{i \in I_A} t_i + \sum_{i \in I_B} (s_i + 2t_i)} \right\}.$$

Since $\sum_{i \in I_A} t_i + \sum_{i \in I_B} s_i = \frac{1}{2}W$,

$$\mathrm{Ncut}_2^{\max}(A, B) = \max\left\{ \frac{\frac{1}{2}W}{\sum_{i \in I_A} 2s_i + \frac{1}{2}W}, \ \frac{\frac{1}{2}W}{\frac{1}{2}W + \sum_{i \in I_B} 2t_i} \right\}. \tag{4}$$

If $\{A, B\}$ forms a yes-instance for a partition input, then $\sum_{i \in I_A} t_i = \sum_{i \in I_B} s_i = D = \frac{1}{4}W$. So we get the following:

$$\mathrm{Ncut}_2^{\max}(A, B) = \max\left\{ \frac{\frac{1}{2}W}{\frac{1}{2}W + \frac{1}{2}W}, \ \frac{\frac{1}{2}W}{\frac{1}{2}W + \frac{1}{2}W} \right\} = \frac{1}{2} \leq N.$$

Now we show that if we have a solution for the max normalized 2-cut at most $N = \frac{1}{2}$, then we have a yes-instance for PARTITION. We have two possible cuts: either a and b are in the same vertex set or a and b are separated.

First, let a and b be in the same vertex set, so $a, b \in A$. Now B only consists of a number of v_i's. Let $I \subseteq \{1, \ldots, p\}$ define the set of v_i's that are in B. Note that $I \neq \emptyset$, because we need to make a 2-cut. We write $V(I) = \{v_i \mid i \in I\}$. We will show that $\mathrm{Ncut}\,(V(I), V \setminus V(I)) > N = \frac{1}{2}$.

$$\begin{aligned} \mathrm{Ncut}_2^{\max}(V(I), V \setminus V(I)) &= \max\left\{ \frac{\sum_{i \in I} (s_i + t_i)}{\sum_{i \in I} (s_i + t_i)}, \ \frac{\sum_{i \in I} (s_i + t_i)}{\sum_{i \in I} (s_i + t_i) + 2\sum_{i \notin I} (s_i + t_i)} \right\} \\ &= 1 > N. \end{aligned}$$

Secondly, consider the cut $\{A, B\}$, with $a \in A$ and $b \in B$. We take the cut value as written in (4).

$$\mathrm{Ncut}_2^{\max}(A, B) = \max\left\{ \frac{\frac{1}{2}W}{\sum_{i \in I_A} 2s_i + \frac{1}{2}W}, \ \frac{\frac{1}{2}W}{\frac{1}{2}W + \sum_{i \in I_B} 2t_i} \right\}.$$

We know that $\sum_{i\in I_A} 2s_i + \sum_{i\in I_B} 2t_i = W$. Let $x = \sum_{i\in I_A} 2s_i$ and $y = \sum_{i\in I_B} 2t_i$. It follows that this cut is optimized when $x = y = \frac{1}{2}W$. Then we get:

$$\text{Ncut}_2^{\max}(A,B) = \max\left\{\frac{\frac{1}{2}W}{\frac{1}{2}W+\frac{1}{2}W},\ \frac{\frac{1}{2}W}{\frac{1}{2}W+\frac{1}{2}W}\right\} = \frac{\frac{1}{2}W}{W} = \frac{1}{2} \leq N.$$

We have a solution for the Max Normalized 2-Cut problem and we have a yes-instance for Partition.

Since we now have a solution for Partition if and only if we have a solution for the Max Normalized 2-Cut problem smaller than or equal to N, this proof is complete. ■

4.2 Vertex Cover

In Section 4.1 we have proved that the Normalized Cut problem and the Max Normalized 2-Cut problem are hard even for vertex cover number equal to 2. Since we have seen their relation with Partition we will give algorithms for both cut problems parametrized by the vertex cover number and the total sum of the edge-weights.

4.2.1 Normalized Cut

Before we give an algorithm for the Normalized Cut problem, we will give an algorithm to find the most optimal cut in a given diamond graph, such that $a \in A$ and $b \in B$ for a cut $\{A, B\}$. We will call this cut: *a cut in between a and b*. Note that this algorithm does not necessarily find the most optimum cut for a diamond graph, but we need the cut to be in between a and b for further use.

Theorem 4.5. *Using the normalized cut, we can find an optimal cut in between a and b in a diamond graph in $O(nW)$ time and in $O(nW)$ space, where W is the total sum of the edge-weights.*

Proof. This proof is based on the proof given by Javadi and Nikabadi [15] for the Sparsest Cut problem and on the dynamic program of the Knapsack problem.

Let $G = (V, E)$ be our diamond-input graph with weighted edges, see Figure 2. We have the vertices $v_1, \ldots, v_p$ and a and b. For every v_i, $1 \leq i \leq p$, we have the weight s_i for the edge to a and the weight t_i for the edge to b. If an edge does not exist, its weight equals 0. Let W be the total sum of the edge-weights. Let $A, B \subset V$.

Step I: Define $A = \{a, v_1, v_2, \ldots, v_p\}$ and $B = \{b\}$ to be the two potential sets for the cut. The value $s_i - t_i$ is the cost of moving a vertex v_i in A to B.

The idea is to check for every possible volume $0 \leq j \leq 2W$ for B what the smallest possible cut value is, using a dynamic program. We write $\text{cut}[i, j]$ for a cell in the table which saves the smallest possible cut value for the first i v's and volume j for set B. A cell in the table has the following value:

$$\text{cut}[i,j] = \min_{\substack{B\subseteq\{b,v_1,\ldots,v_i\},\\ b\in B,\ A=V\setminus B,\\ \text{vol}(B,V)=j}} \text{cut}(A,B).$$

For the first i v's we check if the cut value lowers when we switch some of the v's to B. When the table is filled, the value of $\text{cut}[i, j]$ is the lowest possible cut value for this specific i and j.

Step II: The recurrence relation is:

$$\text{cut}\,[i,j] = \min\{\text{cut}[i-1,j],\ \text{cut}[i-1,j-s_i-t_i]+s_i-t_i\}$$

where the first term is for v_i staying in A and the second term for v_i switching to B. The base case of this recurrence relation is:

$$\text{cut}\,[0,j] = \begin{cases} \sum_{1\le i\le p} t_i & \text{for } j = \sum_{1\le i\le p} t_i, \\ +\infty & \text{otherwise.} \end{cases}$$

Since B only consists of one vertex, which is b, we have the volume of B equal to the cut value. All the edges t_i will be in the cut and in the volume of B.

We will prove this recurrence relation to be correct. Let $\{A, B\}$ be an optimal solution for i and j. Then vertex i can either be in A or in B.

If vertex i is in A, the volume of B will not change and the cut value will not change. Therefore, we can just take the cut value as for the first $i-1$ vertices. Thus, $\text{cut}\,[i,j] = \text{cut}\,[i-1,j]$.

If vertex i is in B, then the volume of B and the cut value change. First, we look at the volume of B. When vertex i was in set A, the edge t_i was part of the cut. So t_i counted once for the volume of B. If i is in B, then s_i counts once for the volume of B (since this edge is in the cut) and the edge t_i counts twice for the volume of B. If we look back in the table we have to subtract one time s_i and one time t_i. Then, for the cut value the following holds. First t_i was part of the cut, now s_i is part of the cut. So we add s_i to the cut value and we subtract t_i from the cut value. It follows that if i is in B, then $\text{cut}\,[i,j] = \text{cut}\,[i-1,j-s_i-t_i]+s_i-t_i$.

Step III: If the table by this recurrence relation is filled, we can find the normalized cut of G (in between a and b) as follows:

$$\text{Ncut}\,(G) = \min_{0\le j\le 2W} \frac{\text{cut}\,[p,j]\cdot 2W}{(2W-j)\cdot j}.$$

Note that this algorithm always finds a cut in between a and b, since a and b never switch from A to B or the other way around. Step I costs $O\,(n\log(n))$ time, step II costs $O\,(nW)$ time and step III costs $O(W)$ time. Thus, the dynamic program runs in $O\,(nW)$ time. The table uses $O\,(nW)$ space. ■

Now that we have an algorithm for a diamond graph, we will give an algorithm for the Normalized Cut problem. We extend the diamond graph such that it corresponds to the given graph. The algorithm in the proof of Theorem 4.6 guesses a partition of the vertex cover and then applies the algorithm in the proof of Theorem 4.5.

Theorem 4.6. *The* Normalized Cut *problem can be solved in* $O\,(2^\tau nW)$ *time and* $O\,(nW)$ *space, where* τ *is the minimum vertex cover and* W *is the total sum of the edge-weights.*

Proof. This proof is based on the proof of Theorem 4.5. Let $G = (V, E)$ be our input graph with weighted edges. Let W be the total sum of the edge-weights. Let C be a vertex cover of G with $|C| = \tau$ and let $I = V(G) \setminus C$ be the independent set. If a vertex cover is not given, we can find an optimal vertex cover in $O\,(2^\tau n)$ time [11]. Let $\tilde{C} \subseteq C$ be a fixed subset.

We can find a diamond look-alike in the graph that we have now. Call the vertices of the independent set $v_1, v_2, \ldots, v_p$ with $p = |I|$. Now let vertex a of the diamond graph correspond with the set $\tilde{C}$ and let b correspond with $C \setminus \tilde{C}$. It follows that for every $1 \le i \le p$ we have $s_i = w(v_i, \tilde{C})$ and $t_i = w(v_i, C \setminus \tilde{C})$. If a certain s_i or t_i does not exist (because there are no edges from v_i to $\tilde{C}$ or $C \setminus \tilde{C}$), then we define this to be 0. Note that the structure that we have now looks similar to that of a diamond graph. Two important differences are

that a and b are now a set of vertices with possible inner-edges and there are possible edges between a and b. Now let $A = \tilde{C} \cup I$ and $B = C \setminus \tilde{C}$ be a partition of V.

We will run the algorithm from the proof of Theorem 4.5 for every $\tilde{C} \subseteq C$ including two small changes. For a specific $\tilde{C} \subseteq C$ we write $\text{cut}_{\tilde{C}}[i, j]$ for the cut value. Furthermore, we have to take possible edges between and inside $\tilde{C}$ and $C \setminus \tilde{C}$ into account. In Step II we save the smallest cut in a table for a specific $\tilde{C}$. In Step III we calculate the lowest cut value for this specific $\tilde{C}$. The minimum normalized cut of G is

$$\text{Ncut}(G) = \min_{\tilde{C} \subseteq C} \text{Ncut}_{\tilde{C}}(G).$$

We remember the $\tilde{C}$ with the lowest cut value to keep track of the normalized cut of G.

Step I: We have $A = \tilde{C} \cup I$ and $B = C \setminus \tilde{C}$, as defined above. The value $s_i - t_i$ is the cost to switch v_i from A to B.

Step II: In $\text{cut}_{\tilde{C}}[i, j]$ we have i for the first i vertices $v_1, \ldots, v_i$ and j for the volume of B. The recurrence relation looks the same as for the diamond, as described in the proof of Theorem 4.5. It follows that the recurrence relation equals:

$$\text{cut}_{\tilde{C}}[i, j] = \min\left\{\text{cut}_{\tilde{C}}[i-1, j],\ \text{cut}_{\tilde{C}}[i-1, j - s_i - t_i] + s_i - t_i\right\},$$

where the first term is for i staying in A and the second for i switching to B.

The base case is different, since we have possible edges between $\tilde{C}$ and $C \setminus \tilde{C}$ and inside of $C \setminus \tilde{C}$ that influence the cut value and the volume j. The cut not only consists of edges from $C \setminus \tilde{S}$ to v_i for $1 \leq i \leq p$, but also from $C \setminus \tilde{C}$ to $\tilde{C}$. In the volume of B we have also inner-edges to take into account. This gives us the following base case:

$$\text{cut}_{\tilde{C}}[0, j] = \begin{cases} R & \text{for } j = R + w(C \setminus \tilde{C}, C \setminus \tilde{C}), \\ +\infty & \text{otherwise.} \end{cases}$$

where $R = w(C \setminus \tilde{C}, \tilde{C}) + \sum_{1 \leq i \leq p} t_i$.

Step III: Now the normalized cut of G, using this specific $\tilde{C}$, is equal to

$$\text{Ncut}_{\tilde{C}}(G) = \min_{0 \leq j \leq 2W} \frac{\text{cut}_{\tilde{C}}[p, j] \cdot 2W}{(2W - j) \cdot j}.$$

Since we use the algorithm of Theorem 4.5 for every $\tilde{C} \subseteq C$, this dynamic program runs in $O(2^\tau nW)$ time. The space usage is still $O(nW)$. ■

4.2.2 Max Normalized 2-Cut

The Max Normalized 2-Cut problem can be solved using a very similar algorithm as the one for solving the Normalized Cut problem in Theorem 4.6.

Theorem 4.7. *The* Max Normalized 2-Cut *problem can be solved in $O(2^\tau nW)$ time and in $O(nW)$ space, where τ is the minimum vertex cover and W is the total sum of the edge-weights.*

Proof. This proof follows the algorithm given in the proof of Theorem 4.6. Let $G = (V, E)$ be our input graph with weighted edges. Let W be the total sum of the edge-weights. Let C be a vertex cover of G with $|C| = \tau$ and let $I = V \setminus C$ be the independent set. Let $\tilde{C} \subseteq C$ be a fixed subset. If a vertex cover is not given, we can find an optimal vertex cover in $O(2^\tau n)$ time [11].

Call the vertices of the independent set $v_1, v_2, \ldots, v_p$ with $p = |I|$. For every $1 \leq i \leq p$ we have $s_i = w(v_i, \tilde{C})$ and $t_i = w(v_i, C \setminus \tilde{C})$. If a certain s_i or t_i does not exists, let this be 0. Let $A = \tilde{C} \cup I$ and $B = C \setminus \tilde{C}$ be a partition of V.

We want to minimize

$$\max \left\{ \frac{\operatorname{cut}(A,B)}{\operatorname{vol}(A,V)}, \frac{\operatorname{cut}(A,B)}{\operatorname{vol}(B,V)} \right\}.$$

Note that for a chosen $\operatorname{vol}(B,V)$, we know that $\operatorname{vol}(A,V) = 2W - \operatorname{vol}(B,V)$. So to minimize the maximum normalized 2-cut for a specific volume of B we have to find the lowest cut value. So we could fill a table for every $\tilde{C}$ where we save for the first i vertices v_i and for volume $0 \leq j \leq 2W$ what the minimum cut value is. Now remark that in the algorithm described in Theorem 4.6 we fill a table that looks exactly like this. It follows that we can use the dynamic program that is described in Theorem 4.6.

To make this algorithm suitable for the max normalized 2-cut, we only have to change the formula in **Step III** and the calculation of the minimum max normalized 2-cut. The latter equals:

$$\operatorname{Ncut}_2^{\max}(G) = \min_{\tilde{C} \subseteq C} \operatorname{Ncut}_{2\,\tilde{C}}^{\max}(G).$$

Step III: The max normalized 2-cut of G for a specific $\tilde{C}$, is equal to

$$\operatorname{Ncut}_{2\,\tilde{C}}^{\max}(G) = \min_{0 \leq j \leq 2W} \max \left\{ \frac{\operatorname{cut}_{\tilde{C}}[p,j]}{2W - j}, \frac{\operatorname{cut}_{\tilde{C}}[p,j]}{j} \right\}.$$

It follows that this algorithm runs in $O(2^\tau nW)$ time and it uses $O(nW)$ space ■

4.3 Treewidth

Shi and Malik [26] have proven that the Normalized Cut problem is NP-complete. From this reduction it also follows that the Normalized Cut problem is NP-complete for graphs with treewidth equal to 3.

In Theorem 4.10 we prove that the Normalized Cut problem is fixed parameter tractable for treewidth combined with the total sum of the edge-weights. In Theorem 4.11 we prove this for the Max Normalized 2-Cut problem. First, we will recall the definition of a *tree decomposition* and *treewidth*.

Definition 4.8 (Tree Decomposition and Treewidth [3, 15])**.** A tree decomposition of a graph $G = (V, E)$ is a pair $\left(T, \{X_t\}_{t \in V(T)}\right)$ where T is a tree for which every node t is assigned a vertex subset $X_t \subseteq V(G)$, called a *bag*, such that the following three conditions hold:

1. $\bigcup_{t \in V(T)} X_t = V(G)$, which means that every vertex of G is at least in one bag.
2. For every edge $(u, v) \in E(G)$, there exists a node $t \in T$ such that the bag X_t contains both u and v.
3. For every $u \in V(G)$, the set $T_u = \{t \in V(T) \mid u \in X_t\}$, i.e., the set of nodes whose corresponding bags contain u, induces a connected subtree of T.

The treewidth of a tree decomposition $\left(T, \{X_t\}_{t \in V(T)}\right)$ is $\max_{t \in V(T)} |X_t| - 1$. The treewidth of a graph G is the minimum treewidth over all possible tree decompositions of G.

Furthermore, we will use the following notation: For every node $i \in V(T)$ we have subtree T_i rooted at node i. Let $G_i = (V_i, E_i)$ be the induced subgraph of G induced by the vertices in $\bigcup_{j \in V(T_i)} X_j$. ⊠

From a tree decomposition we can make a *nice tree decomposition*, first introduced by Kloks [16], which we will use in our proofs.

Definition 4.9 (Nice Tree Decomposition [4, 15])**.** Let $G = (V, E)$ be a given graph. We call a tree decomposition $\left(T, \{X_t\}_{t \in V(T)}\right)$ of G a nice tree decomposition if T is a binary tree rooted at node r, where every node is of one of the following four nodes.

1. **Leaf node:** If node i is a leaf of T and $|X_i| = 1$.
2. **Introduce node:** If node i has exactly one child j such that $X_i = X_j \cup \{v\}$ for some vertex $v \notin X_j$.
3. **Forget node:** If node i has exactly one child j such that $X_i = X_j \setminus \{v\}$ for some vertex $v \in X_j$.
4. **Join node:** If node i has exactly two children j and k such that $X_i = X_j = X_k$.

⊠

A tree decomposition can be transformed into a nice tree decomposition using a linear time algorithm [15, 16].

4.3.1 Normalized Cut

In the following theorem we will prove that the Normalized Cut problem is fixed parameter tractable for treewidth combined with the total sum of the edge-weights.

Theorem 4.10. *The* Normalized Cut *problem can be solved in* $O\left(2^{tw} n W^2\right)$ *time when a (nice) tree decomposition is given and in* $O\left(2^{tw} n W\right)$ *space, where* tw *is the treewidth and* W *is the total sum of the edge-weights.*

Proof. This proof is based on the proof given by Javadi and Nikabadi [15] for the Sparsest Cut problem. We will use a dynamic program to solve this problem. Let tw be the treewidth of input graph G. Let $\left(T, \{X_t\}_{t \in V(T)}\right)$ be a nice tree decomposition of G. Define A and B as (possibly empty) subsets of $V(G)$.

We use the following notation. We write $\mathrm{cut}_i[C, f]$ for the cut value of node i, where $C \subseteq X_i$ and $f = \mathrm{vol}(A, V_i)$. We save all those values in a table of which the rows represent (i, C) where $i \in V(T)$ and $C \subseteq X_i$ and the columns represent $0 \leq f \leq 2W$. A cell of the table has the following value:

$$\mathrm{cut}_i[C, f] = \min_{\substack{A \subseteq V_i,\ B = V_i \setminus A,\\ C = X_i \cap A,\\ f = \mathrm{vol}(A, V_i)}} \mathrm{cut}(A, B).$$

When there is no feasible solution, the value of $\mathrm{cut}_i[C, f]$ is equal to $+\infty$.

We will fill the table in postorder, i.e. bottom-up, using the following recursive relations.

Leaf node: For a single leaf node i, its bag X_i contains only one vertex and no edges. It follows that the cut value is zero when the volume is zero:

$$\mathrm{cut}_i[C, f] = \begin{cases} 0 & \text{if } f = 0, \\ +\infty & \text{otherwise.} \end{cases}$$

Introduce node: For an introduce node i with child node j and $X_i = X_j \cup \{v\}$ we have either the choice to put v into set C (and thus A) or not:

$$\text{cut}_i\,[C, f] = \begin{cases} \text{cut}_j\,[C, f - w(v, C)] + w(v, C) & \text{if } v \notin C, \\ \text{cut}_j\,[C \setminus \{v\}, f - w(v, X_i \setminus C) - 2w(v, C)] + w(v, X_i \setminus C) & \text{if } v \in C. \end{cases}$$

If $v \notin C$, then the edge(s) from v to C is/are in the cut. So the cut value increases with the value $w(v, C)$. Since the edges that are in the cut are also part of the volume of A, we have to subtract this from f to search back in the table for the optimum cut value where $v \notin C$.

If $v \in C$, and thus $v \in A$, then the cut only increases with the value of the edges out of C, i.e. $w(v, X_i \setminus C)$. The rest of the edges are inner-edges of C, so the value counts double for the volume, which is represented by $2w(v, C)$. We have to subtract the inner-edges as well as the edges in the cut, when we look back in the table.

Forget node: For a forget node i with child node j and $X_i = X_j \setminus \{v\}$ we can take the minimum value of the solutions either including v or excluding v:

$$\text{cut}_i\,[C, f] = \min\left\{\text{cut}_j\,[C \cup \{v\}, f]\,,\ \text{cut}_j\,[C, f]\right\}.$$

Join node: For a join node i with children j and k, where $X_i = X_j = X_k$, we can take the combined optimum solution of its children:

$$\text{cut}_i\,[C, f] = \min_{f_1 + f_2 = f + \text{vol}(C, X_i)} \left\{\text{cut}_j\,[C, f_1] + \text{cut}_k\,[C, f_2] - \text{cut}\,(C, X_i \setminus C)\right\}.$$

To make sure that the saved cut value in the table is correct, we take the sum of the cut values of j and k and we subtract $\text{cut}\,(C, X_i \setminus C)$. Since $X_i = X_j = X_k$, the cut value of j and the cut value of k include the value $\text{cut}\,(C, X_i \setminus C)$. To save a correct cut value in the table we have to subtract this double-counted part once.

We show that, given a C and an f, if we have an optimal cut for j and k, then we can form an optimal cut for i. By Definition 4.8 point 3 we know that the vertices in the childnodes of X_j and X_k, excluding the vertices that are in X_j and X_k, are independent from each other, otherwise we would not have an induced connected subtree for some vertex in T_j or T_k. So we can take an optimum cut value for some volume f_1 in X_j and f_2 in X_k such that the volumes add up to the given f. Now remark that, since $X_i = X_j = X_k$, in f_1 and f_2 we count $\text{vol}\,(C, X_i)$ twice. So it must hold that $f_1 + f_2 = f + \text{vol}\,(C, X_i)$. Then we have found a cut for i for the given C and f.

It rests us to show that the cut for i, which is the minimum cut over all the possible cuts, is also optimal. Suppose that we can find a better cut for i, for a certain C and f. Let $\text{cut}\,(A, B) < \infty$ be this cut. Then it follows that $A \subseteq V_i$. Let $A_j = A \cap V_j$ and $A_k = A \cap V_k$. Now, define $f_1 = \text{vol}\,(A_j, V_j)$ and $f_2 = \text{vol}\,(A_k, V_k)$. Since $X_i = X_j = X_k$ the term $\text{vol}\,(C, X_i)$ is counted in both f_1 and f_2. It follows that $f_1 + f_2 = f + \text{vol}\,(C, X_i)$ holds and we have found a better cut for either j or k or both. Since we concluded that the cut for j and k are optimal, we have a contradiction. It follows that the minimum cut for i is optimal.

We can find the minimum normalized cut of the given graph by calculating:

$$\text{Ncut}(G) = \min_{\substack{C \subseteq X_r, \\ 0 \leq f \leq 2W}} \frac{\text{cut}\,[C, f] \cdot 2W}{f \cdot (2W - f)}.$$

The table has size $n2^{tw} \cdot 2W$. The leaf nodes, introduce nodes and forget nodes can be done in constant time. The join node will take $O(W)$ time. We can find the minimum

normalized cut of the graph in $O\left(2^{|X_r|}W\right)$ time. This results in an algorithm that runs in $O\left(2^{tw}nW^2\right)$. When the tree decomposition is not given, we can find one in $tw^{O(tw^3)}$ time [3] or approximate one in $2^{O(tw)}n$ time [5]. ■

4.3.2 Max Normalized 2-Cut

Below we will give an algorithm that solves the Max Normalized 2-Cut with the parameters treewidth and total sum of the edge-weights.

Theorem 4.11. *The* Max Normalized 2-Cut *problem can be solved in* $O\left(2^{tw}nW^2\right)$ *time when a (nice) tree decomposition is given and in* $O\left(2^{tw}nW\right)$ *space, where* tw *is the treewidth and* W *is the total sum of the edge-weights.*

Proof. This proof follows the algorithm given in the proof of Theorem 4.10. We use a dynamic program to solve this problem. Let tw be the treewidth of input graph G. Let $\left(T, \{X_t\}_{t\in V(T)}\right)$ be a nice tree decomposition of G. Define A and B as (empty) subsets of V.

We use the following notation. We write $\mathrm{cut}_i[C, f]$ for the cut value of node i, where $C \subseteq X_i$ and $f = \mathrm{vol}(A, V_i)$. We will save all those values in a table of which the rows represent (i, C) where $i \in V(T)$ and $C \subseteq X_i$ and the columns represent $0 \leq f \leq 2W$. A cell of the table has the following value:

$$\mathrm{cut}_i[C, f] = \min_{\substack{A\subseteq V_i,\ B=V_i\setminus A\\ C=X_i\cap A\\ f=\mathrm{vol}(A,V_i)}} \mathrm{cut}(A, B).$$

When there is no feasible solution, the value of $\mathrm{cut}_i[C, f]$ is equal to $+\infty$.

Note that we now have the same set-up for a dynamic program as given in the proof of Theorem 4.10 for the Normalized Cut problem. A cell in the table holds the minimum cut value for a specific C and f. When we calculate the max normalized 2-cut, we want the lowest cut value for a chosen volume f. It follows that we can use the dynamic program as described in the proof of Theorem 4.10.

We find the minimum max normalized 2-cut of the given graph by calculating:

$$\mathrm{Ncut}(G) = \min_{\substack{C\subseteq X_r,\\ 0\leq f\leq 2W}} \max\left\{\frac{\mathrm{cut}[C, f]}{f},\ \frac{\mathrm{cut}[C, f]}{2W - f}\right\}.$$

Thus, we have an algorithm that runs in $O\left(2^{tw}nW^2\right)$ time and uses $O\left(2^{tw}nW\right)$ space. ■

5 Unweighted k-Cuts

We will give algorithms for the Unweighted Normalized k-Cut problem and the Unweighted Max Normalized k-Cut problem parameterized by treewidth in Section 5.1 and by the vertex cover number in Section 5.2. The vertex cover algorithm is an improvement compared to the treewidth algorithm in time and space, as explained in Section 5.2.

Recall that we write w_{ij} for the weight of the edge between vertex i and j. Since we work with an unweighted graph, it holds that $w_{ij} = 1$ for every i and j with $(i, j) \in E(G)$.

5.1 Treewidth

For the definition of treewidth and (nice) tree decomposition see Section 4.3. First we look at the Unweighted Normalized k-Cut problem.

5.1.1 Unweighted Normalized k-Cut

Below we give an algorithm for the Unweighted Normalized k-Cut problem parameterized by treewidth.

Theorem 5.1. *For every fixed integer $k \geq 2$, the* Unweighted Normalized k-Cut *problem can be solved in $O\left(k^{tw}2^k m^{4k} n\right)$ time and in $O\left(k^{tw}2^k m^{2k} n\right)$ space, where tw is the treewidth of the input graph.*

Proof. The idea of this proof is based on the proof of Javadi and Nikabadi for the Unweighted k-Sparsest Cut problem [15]. Also, this proof follows the same idea as the proof of Theorem 4.10 which is the algorithm for the Normalized k-Cut problem parameterized by treewidth. We will use a dynamic program which fills a table postorder, i.e. bottom-up. We use a nice tree decomposition. Let tw be the treewidth of input graph $G = (V, E)$ with unweighted edges.

For each $i \in V(T)$ we define the following notation. Define $\mathcal{S} = (S_1, \ldots, S_k)$ to be a k-partition of V_i and let it be a solution for the Unweighted Normalized k-Cut problem on G_i. Let $c \in \{1, \ldots, k\}^{|X_i|}$ be a configuration vector, where $c[u] = j$ if and only if $u \in X_i \cap S_j$. In words, for every vertex $u \in X_i$ the vector c saves which S_j of the solution vertex u is in. To keep track of the number of inner-edges and outgoing edges we introduce the vectors $e_{in} \in \{0, \ldots, 2m\}^k$ and $e_{out} \in \{0, \ldots, m\}^k$, where $e_{in}[j] = w(S_j, S_j)$ and $e_{out}[j] = w(S_j, V_i \setminus S_j)$. Since the inner-edges represent the volume, we count the number of inner-edges twice in e_{in}.

We make a table with $|V(T)|$ rows and at most $k^{tw}(2m)^k m^k = k^{tw}2^k m^{2k}$ columns, where each row represents a node $i \in V(T)$ and every column represents a combination of a configuration vector c and two edge-count vectors e_{in} and e_{out}. A cell in the table $\text{cut}[i, c, e_{in}, e_{out}]$ is equal to 1 if and only if there is a k-partition $(S_1, \ldots, S_k)$ of $V(G_i)$ such that for each $j \in \{1, \ldots, k\}$, it holds that $S_j \cap X_i = \{u \in X_i \mid c[u] = j\}$, $w(S_j, S_j) = e_{in}[j]$ and $w(S_j, \bar{S_j}) = e_{out}[j]$. If no such partition exists, then $\text{cut}[i, c, e_{in}, e_{out}] = 0$.

We will fill the table bottom-up using the following recursive relations.

Leaf node: A single leaf node i has its corresponding bag X_i which contains only one vertex and no edges. It follows that the table contains a 1 if and only if e_{in} and e_{out} are vectors with only zeros.

$$\text{cut}\,[i, c, e_{in}, e_{out}] = \begin{cases} 1 & \text{if } e_{in}[u] = e_{out}[u] = 0 \text{ for all } u \in X_i, \\ 0 & \text{otherwise.} \end{cases}$$

Introduce node: Let i be an introduce node with child l such that $X_i = X_l \cup \{v\}$. Let c, e_{in} and e_{out} be three vectors for node i. Then let $c[v] = j_v$. We introduce the three vectors c^l, e^l_{in} and e^l_{out} for node l. Let c^l be the restriction of configuration vector c on X_l and define e^l_{in} and e^l_{out} as follows:

$$e^l_{in}[j] = \begin{cases} e_{in}[j] & \text{if } j \neq j_v, \\ e_{in}[j] - 2w(v, c^{-1}(j_v)) & \text{if } j = j_v, \end{cases}$$

$$e^l_{out}[j] = \begin{cases} e_{out}[j] - w(v, c^{-1}(j)) & \text{if } j \neq j_v, \\ e_{out}[j] - w(v, X_i \setminus c^{-1}(j_v)) & \text{if } j = j_v, \end{cases}$$

where $c^{-1}(j) = \{u_i \mid u_i \in S_j\}$.

If $j \neq j_v$, then for the total sum of the inner-edges it changes nothing, since the inner-edges of S_j and S_{j_v} are independent from each other. For the outgoing edges it holds that

every edge from v in S_{j_v} to a vertex in S_j is part of the cut. Since vertex v does not exist in bag X_l, we have to subtract these edges from the value of $e_{out}[j]$ to be correct for $e^l_{out}[j]$.

If $j = j_v$, then introducing vertex v has influence on the inner-edges of S_{j_v}. So for e^l_{in} to be correct, we have to subtract all the inner-edges connected to v to get the correct value of e^l_{in}. Since we save the volume and $w(v, c^{-1}(j_v))$ counts the edges only once, we subtract this term twice. For the outgoing edges it holds that all the edges from $v \in S_{j_v}$ to vertices in other S_j are part of the cut. So we have to subtract those edges from e_{out} for e^l_{out} to be correct.

Now, for the introduce node, we check for every possible j_v if we we have a solution. We get:

$$\text{cut}\left[i, c, e_{in}, e_{out}\right] = \text{cut}\left[l, c^l, e^l_{in}, e^l_{out}\right].$$

Forget node: Let i be a forget node with child l such that $X_i = X_l \setminus \{v\}$. Let c, e_{in} and e_{out} be the three vectors for node i. For child node l the configuration c of X_i is extended with the choice in which S_j vertex v should be. If we want to know if a certain $\text{cut}[i, c, e_{in}, e_{out}]$ is possible, we need to know if one of the solutions including v is possible. Thus, for every $j \in \{1, \ldots, k\}$ we define configuration vector c_j such that $c_j[v] = j$ and $c_j[u] = c[u]$ for every $u \in X_i$. Now we can check for all these possible configurations if one of those is possible. We get the following:

$$\text{cut}\left[i, c, e_{in}, e_{out}\right] = \max_{1 \le j \le k} \text{cut}\left[l, c_j, e_{in}, e_{out}\right].$$

Join node: Let i be a join node with two children l_1 and l_2 such that $X_i = X_{l_1} = X_{l_2}$. We write e^t_{in} for the inner-edges of child l_t. Let $\mathcal{A}$ be the set of 4-tuples $\left(e^1_{in}, e^2_{in}, e^1_{out}, e^2_{out}\right)$ such that the following two equations hold. For all $j \in \{1, \ldots, k\}$

$$e^1_{in}[j] + e^2_{in}[j] = e_{in}[j] + w(c^{-1}(j), c^{-1}(j)), \tag{5}$$

$$e^1_{out}[j] + e^2_{out}[j] = e_{out}[j] + w(c^{-1}(j), X_i \setminus c^{-1}(j)). \tag{6}$$

Now for join node i we take the combined optimum for its children.

$$\text{cut}\left[i, c, e_{in}, e_{out}\right] = \max_{\left(e^1_{in}, e^2_{in}, e^1_{out}, e^2_{out}\right) \in \mathcal{A}} \min_{t \in \{1,2\}} \text{cut}\left[l_t, c, e^t_{in}, e^t_{out}\right].$$

Note that both e^1_{in} and e^2_{in} count the inner edges of vertices in X_i. It follows that given $e^1_{in}[j]$ and $e^2_{in}[j]$ for a specific S_j, we count the volume of the inner-edges of all the vertices in the collection $c^{-1}(j)$ twice. Therefore, (5) must hold. Note that the last term in (5) already counts the number of inner-edges twice, so it equals the volume of the inner-edges. A similar argument can be made for the edges in the cut, e_{out}. Given an S_j, both $e^1_{out}[j]$ and $e^2_{out}[j]$ count the edges from $c^{-1}(j)$ to $X_i \setminus c^{-1}(j)$. It follows that (6) must hold. Now it holds that we have a valid cut for i with e_{in} and e_{out} if and only if (5) and (6) hold.

We show that if we have two valid cuts for l_1 and l_2, then we have a valid cut for i. By Definition 4.8 point 3 we know that the vertices in the childnodes of X_{l_1} and X_{l_2}, excluding the vertices that are in X_{l_1} and X_{l_2}, are independent from each other, otherwise we would not have an induced connected subtree for some vertex in T_{l_1} or T_{l_2}. Now, if both $\text{cut}\left[l_1, c, e^1_{in}, e^1_{out}\right]$ and $\text{cut}\left[l_2, c, e^2_{in}, e^2_{out}\right]$ are equal to 1, and they follow (5) and (6), then the combined solution is a possible cut for node i. Moreover, the reverse proof is analogous.

When we have the table filled, we can find the minimum unweighted normalized k-cut of the given graph. We check all the table instances for the root r: $\text{cut}\left[r, c, e_{in}, e_{out}\right]$. If the

instance is equal to 0, then we define the unweighted normalized k-cut to be $+\infty$. If the instance is equal to 1, we can calculate the unweighted normalized k-cut as follows:

$$\mathrm{Ncut}_k(\mathcal{S}) = \sum_{j=1}^{k} \frac{e_{out}[j]}{e_{out}[j] + e_{in}[j]}.$$

We can find the minimum unweighted normalized k-cut of the given graph by calculating:

$$\mathrm{Ncut}_k(G) = \min_{\mathcal{S}} \mathrm{Ncut}_k(\mathcal{S}).$$

The table has size $O\left(k^{tw}2^k m^{2k} n\right)$. For a leaf node we need constant time. Both the introduce and forget node cost $O(k)$ time. The join node will take $O\left(m^{2k}\right)$ time. We can find the minimum unweighted normalized k-cut of the graph in $O\left(k^{tw}2^k m^{2k}\right)$ time. This results in an algorithm that runs in $O\left(k^{tw}2^k m^{4k} n\right)$ time. ■

5.1.2 Unweighted Max Normalized k-Cut

Below we give an algorithm for the Unweighted Max Normalized k-Cut parameterized by treewidth. This proof is based on the proof of Theorem 5.1.

Theorem 5.2. *For every fixed integer $k \geq 2$, the* Unweighted Max Normalized k-Cut *problem can be solved in $O\left(k^{tw}2^k m^{4k} n\right)$ time and in $O\left(k^{tw}2^k m^{2k} n\right)$ space, where tw is the treewidth of the input graph.*

Proof. In the proof of Theorem 5.1 we make a table in which we save the possible cuts with value 1 and the cuts that are not possible with value 0. Since, for the max-variant of the same problem only the calculation of the cut is different, we can use the same dynamic program.

When the table is filled, we can find the minimum unweighted max normalized k-cut by checking all the table instances for the root r: $\mathrm{cut}[r, c, e_{in}, e_{out}]$. If the instance is equal to 0, then we define the unweighted max normalized k-cut to be $+\infty$. If the instance is equal to 1, we can calculate the unweighted max normalized k-cut as follows:

$$\mathrm{Ncut}_k^{\max}(\mathcal{S}) = \max_{1 \leq j \leq k} \frac{e_{out}[j]}{e_{out}[j] + e_{in}[j]}.$$

We can find the minimum unweighted max normalized k-cut of the given graph by calculating:

$$\mathrm{Ncut}_k^{\max}(G) = \min_{\mathcal{S}} \mathrm{Ncut}_k^{\max}(\mathcal{S}).$$

Thus, this algorithm still runs in $O\left(k^{tw}2^k m^{4k} n\right)$ time and uses $O\left(k^{tw}2^k m^{2k} n\right)$ space. ■

5.2 Vertex Cover

Since a bounded vertex cover implies bounded treewidth, we can solve the problem for vertex cover with the treewidth algorithm. However, we will improve the running time and the space of the treewidth algorithm. To show this we will use the upper-bound $tw = \tau$.

We have an $O\left(k^{tw}2^k m^{4k} n\right)$ time algorithm for treewidth from Section 5.1 and below we discuss an $O\left(k^{\tau+1} n^{k+1} (3m)^k\right)$ time algorithm for vertex cover. Since we want the vertex cover algorithm to be an improvement of the treewidth algorithm, and we assumed that $k^{tw} = k^{\tau}$, we want $2^k m^{3k} > k \cdot n^k 3^k$ to hold. We assume that we have a connected graph, if not, we can run the algorithms on each connected component. Furthermore, k is a chosen (small) number. With this, we get $m^{3k} > n^k$, and we see that the vertex cover algorithm will indeed be faster. Moreover, the treewidth algorithm uses $O\left(k^{tw}2^k m^{2k} n\right)$ space and the vertex cover algorithm uses $O\left(n^{k+1}(3m)^k\right)$ space, which is clearly an improvement.

5.2.1 Unweighted Max Normalized k-Cut

Below we give an algorithm for the Unweighted Max Normalized k-Cut.

Theorem 5.3. *For every fixed integer $k \geq 2$, the* Unweighted Max Normalized k-Cut *problem can be solved in $O\left(k^{\tau+1}n^{k+1}(3m)^k\right)$ time and in $O\left(n^{k+1}(3m)^k\right)$ space, where τ is the vertex cover number of the input graph.*

Proof. The idea of this proof is based on the proof of Javadi and Nikabadi for the Unweighted k-Sparsest Cut problem [15] and on the algorithm for the 2-cut in Theorem 4.6. We will use a dynamic program to solve the decision variant of the Unweighted Max Normalized k-Cut problem, i.e. for a given graph G and rational number N does $\text{Ncut}_k^{\max}(G) \leq N$ hold?

Let τ be the vertex cover number of input graph $G = (V, E)$ with unweighted edges. Let C be a vertex cover of G with $|C| = \tau$ and let $I = V \setminus C$ be the independent set. If a vertex cover is not given, we can find an optimal vertex cover in $O(2^\tau n)$ time [11]. For every k-partition $(C_1, \ldots, C_k)$ of C we aim to assign the vertices in I to a C_j such that the unweighted max normalized k-cut will be at most N.

Let $(C_1, \ldots, C_k)$ be a fixed k-partition of C. Let $a_1, \ldots, a_k$ be fixed non-negative integers such that $a_1 + \cdots + a_k = |I|$. We will make a k-partition $(A_1, \ldots, A_k)$ of I such that $|A_j| = a_j$ for $1 \leq j \leq k$. We define $(S_1, \ldots, S_k)$ to be the k-partition of G such that $S_j = C_j \cup A_j$, for $1 \leq j \leq k$.

Now we can calculate the cut-value of S_j in a certain k-partition for $1 \leq j \leq k$. In the cut are all the edges between

1. C_j and $C \setminus C_j$,
2. C_j and $I \setminus A_j$, which equals between C_j and I minus the edges between C_j and A_j,
3. A_j and $C \setminus C_j$, which equals between A_j and C minus the edges between A_j and C_j.

Note that $w(C_j, C \setminus C_j) + w(C_j, I) = w(C_j, V \setminus C_j)$. Then, in the volume we count the edges in the cut plus the edges inside C_j, which equals $w(C_j, C_j)$, and from C_j to A_j, which equals $2w(A_j, C_j)$. It follows that the cut-value for S_j can be calculated as:

$$\frac{\text{cut}\left(S_j, \bar{S_j}\right)}{\text{vol}\left(S_j, V\right)} = \frac{w(C_j, V \setminus C_j) + w(A_j, C) - 2w(A_j, C_j)}{w(C_j, V \setminus C_j) + w(C_j, C_j) + w(A_j, C)}.$$

Since we have a fixed k-partition of C, the first term in the numerator and denominator and the second term in the denominator are constant. Define $W_j = w(C_j, V \setminus C_j)$ and $c_j = w(C_j, C_j)$. For the variable terms we define for every vertex $i \in I$: $x_i = w(i, C)$ and $y_{ij} = w(i, C_j)$. Now we can rewrite the cut value as follows:

$$\begin{aligned}
\frac{\text{cut}\left(S_j, \bar{S_j}\right)}{\text{vol}\left(S_j, V\right)} = \frac{W_j + \sum_{i \in A_j} x_i - 2\sum_{i \in A_j} y_{ij}}{W_j + c_j + \sum_{i \in A_j} x_i} &\leq N \\
\Leftrightarrow W_j + \sum_{i \in A_j} x_i - 2\sum_{i \in A_j} y_{ij} &\leq N\left(W_j + c_j + \sum_{i \in A_j} x_i\right) \\
\Leftrightarrow (1 - N)\sum_{i \in A_j} x_i - 2\sum_{i \in A_j} y_{ij} &\leq (N-1)W_j + Nc_j \qquad (7)
\end{aligned}$$

Solving (7) looks like a variant of the bin packing problem. We define $b_j = (N-1)W_j + Nc_j$.

Problem 5.4 (Bin Packing variant)**.**
Input: Integers $x_1, \ldots, x_{|I|}$, $y_{1,1}, \ldots, y_{|I|,k}$, $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$ and rational number N.
Question: Is there a k-partition $(A_1, \ldots, A_k)$ of $\{1, \ldots, |I|\}$ such that for each $j \in \{1, \ldots, k\}$ it holds that $|A_j| = a_j$ and $(1 - N)\sum_{i \in A_j} x_i - 2\sum_{i \in A_j} y_{ij} \leq b_j$?

Below we will write down the algorithm for the UNWEIGHTED MAX NORMALIZED k-CUT problem.

For every k-partition $(C_1, \ldots, C_k)$ of C we will solve the bin packing variant as described in Problem 5.4. When **Step III** returns a 1, there is a possible solution. We can stop the algorithm and we are done. If for all k-partitions of C **Step III** returns a 0, then we cannot find a unweighted max nomalized k-cut with a value at most N.

Step I: We solve the bin packing variant with a dynamic program. Let $a'_1, \ldots, a'_k$ and $b'_1, \ldots, b'_k$ be integers such that $0 \leq a'_j \leq n$ and $0 \leq b'_j \leq b_j$ for every $1 \leq j \leq k$. Note that $\max_{1 \leq i \leq |I|} b_i \leq 3m$. We fill a table with $|I|$ rows and at most $(n \cdot 3m)^k$ columns.

A cell of this table $T(i; a'_1, \ldots, a'_k; b'_1, \ldots, b'_k)$ equals 1 if and only if there exists a k-partition $(A'_1, \ldots, A'_k)$ of $\{1, \ldots, i\}$ such that for each $j \in \{1, \ldots, k\}$ it holds that $|A'_j| = a'_j$ and $(1-N)\sum_{i \in A'_j} x_i - 2\sum_{i \in A'_j} y_{ij} \leq b'_j$. Otherwise, $T(i; a'_1, \ldots, a'_k; b'_1, \ldots, b'_k) = 0$.

Step II: The recurrence relation looks as follows. We have

$$T\left(i; a'_1, \ldots, a'_k; b'_1, \ldots, b'_k\right) = 1 \tag{8}$$

if and only if for a certain $s \in \{1, \ldots, k\}$

$$T\left(i-1; a'_1, \ldots, a'_s - 1, \ldots, a'_k; b'_1, \ldots, b'_s - z, \ldots, b'_k\right) = 1, \tag{9}$$

where $z = (1-N)\, x_i - 2y_{i,s}$.

The base case is: $T(1; a'_1, \ldots, a'_k; b'_1, \ldots, b'_k)$ equals 1 if and only if there is a $s \in \{1, \ldots, k\}$ such that $a'_s = 1$ and $(1-N)x_1 - 2y_{1,s} \leq b'_s$, and for all $j \in \{1, \ldots, k\} \setminus \{s\}$ we have $a'_j = 0$. Otherwise, it is equal to 0.

If, for a certain $s \in \{1, \ldots, k\}$, we know that (9) is true, then it follows that we put i in A'_s. Now the size of $|A'_s|$ equals a'_s, which is one bigger than for vertices $1, \ldots, i-1$. Since we have for $1, \ldots, i-1$ that $(1-N)\sum_{i \in A_s} x_i - 2\sum_{i \in A_s} y_{i,s} \leq b_s - z$ and for i that $(1-N)\, x_s - 2y_{i,s} = z$ it follows for i that $(1-N)\sum_{i \in A_s} x_i - 2\sum_{i \in A_s} y_{i,s} \leq b_s$. Therefore, we have that (8) holds if and only if for a certain $s \in \{1, \ldots, k\}$ we have that (9) holds.

Step III: We return

$$\max_{a_1 + \cdots + a_k = |I|} T(I; a_1, \ldots, a_k; b_1, \ldots, b_k)\,.$$

If this value is equal to 1, then it is possible to find a normalized k-cut of G for the given k-partition of C and some sum $a_1 + \cdots + a_k = |I|$.

We have k^τ possible k-partitions of C. The table we fill in **Step I** is of size $|I|\,(n \cdot 3m)^k$. We need $O(k)$ time to calculate every cell in Step II. In Step III we read $|I|^k$ possible solutions from the table. Since $|I| \leq n$, this gives us a running time of $O\left(k^{\tau+1} n^{k+1} (3m)^k + k^\tau n^k\right)$ which results in an $O\left(k^{\tau+1} n^{k+1} (3m)^k\right)$ time algorithm. Furthermore, this algorithm uses $O\left(n^{k+1}(3m)^k\right)$ space. ■

5.2.2 Unweighted Normalized k-Cut

Below we give an algorithm for the UNWEIGHTED NORMALIZED k-CUT. This algorithm uses $O\left(2^k m^{2k}\right)$ more time than the same algorithm for the max variant. Then, compared to the treewidth algorithm, we get $m^k > n^k$, so the vertex cover algorithm will still be an improvement. Moreover, the space usages is still a big improvement.

Theorem 5.5. *For every fixed integer $k \geq 2$, the* UNWEIGHTED MAX NORMALIZED k-CUT *problem can be solved in $O\left(k^{\tau+1} n^{k+1} 6^k m^{3k}\right)$ time and uses $O\left(n^{k+1}(3m)^k\right)$ space, where τ is the vertex cover number of the input graph.*

Proof. This proof follows the proof of Theorem 5.3 for the UNWEIGHTED MAX NORMALIZED k-CUT problem. We will use the same dynamic program to solve the decision variant of the UNWEIGHTED NORMALIZED k-CUT problem, i.e. for a given graph G and rational number N does $\mathrm{Ncut}_k(G) \leq N$ hold?

We ask ourselves for every $1 \leq j \leq k$ if $\frac{\mathrm{cut}(S_j, \bar{S}_j)}{\mathrm{vol}(S_j, V)} \leq N_j$ such that $N_1 + \cdots + N_k = N$. Then we can follow the same algorithm as for the max variant of this problem. These N_j will be a fraction between 0 and 1. For the numerator, which represents the cut value, we have m options and for the denominator, which represents the volume, we have $2m$ options. In total this gives us $2m^2$ options for every N_j, thus $2^k m^{2k}$ options for the sum up to N.

Together with the $2^k m^{2k}$ options for the sum up to N, and the fact that $2^k 3^k = 6^k$, this algorithm runs in $O\left(k^{\tau+1} n^{k+1} 6^k m^{3k}\right)$ time. This algorithm still uses $O\left(n^{k+1}(3m)^k\right)$ space. ■

6 k-Cuts

In this section we will prove that the NORMALIZED k-CUT is NP-complete for $\tau = k = 3$, $\tau = k = 4$ and we conjecture that this also holds for general $\tau = k$, where τ is the vertex cover number. Furthermore, we prove that the MAX NORMALIZED k-CUT is NP-complete for $k = \tau$. Moreover, we give algorithms for the (MAX) NORMALIZED k-CUT problems based on the algorithms given in Section 5 for the unweighted variant.

6.1 Hardness Results

We have seen that the NORMALIZED CUT problem is NP-complete for vertex cover number equal to 2. The proofs for the (MAX) NORMALIZED 3,4,k-CUT problems look similar. Again, we use a diamond figure. The 3-diamond graph is shown below.

Definition 6.1 (3-Diamond Graph)**.** The *3-diamond graph* is shown in Figure 3. We have vertices u_1, u_2 and u_3 that form a vertex cover of the graph. In the middle we have the vertices $v_1, \ldots, v_{p+1}$ which have an edge to u_1, u_2 and u_3 with weight w_i, for $1 \leq i \leq p+1$. The vertices v_i have no edges to each other.

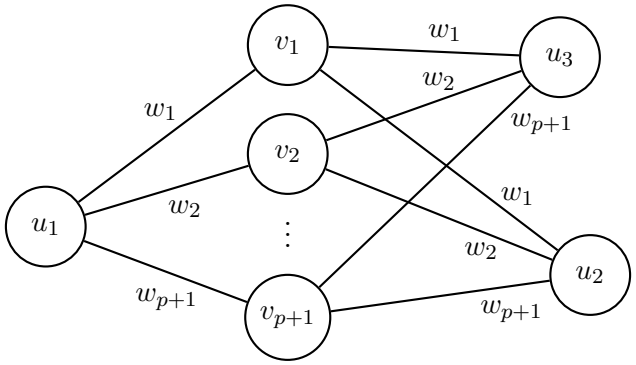


Figure 3: A 3-diamond graph.

⊠

6.1.1 Normalized 3-Cut

Now we will prove that the NORMALIZED 3-CUT problem is NP-complete for vertex cover number equal to 3.

Theorem 6.2. *The* Normalized 3-Cut *problem is NP-complete for graphs with vertex cover number equal to 3.*

Proof. For this proof we will look at the decision variant of the Normalized 3-Cut problem. We ask ourselves if we can find a normalized 3-cut with a value at most N.

First, note that this problem is in NP. We prove NP-completeness using a 3-diamond graph and using a reduction from Partition, see Definition 4.2. We make an instance of a 3-diamond graph. Let $G = (V, E)$ be our 3-diamond graph. Let $v_1, \ldots, v_{p+1}$ be the vertices in the middle and add three vertices u_1, u_2 and u_3. Now let $w_i = c_i$ be the weight of the edges between v_i and u_1, u_2 and u_3 respectively, for $1 \leq i \leq p$. Let $w_{p+1} = D$ be the weight of the edge between v_{p+1} and u_1, u_2 and u_3. It follows that the total sum of the edge weights is $W = \sum_{i=1}^{p+1} w_i = 9D$. We choose $N = 2$.

A solution for the 3-diamond graph translates equally to a solution for Partition as we described in the proof of Theorem 4.3 in Section 4 and the same holds for the other way around. Now we have $w_{p+1} = D$. So, for a 3-cut we will see that $S_3 = \{v_{p+1}, u_3\}$ will be part of the solution. Then S_1 and S_2 will define a partition, as described in the proof of Theorem 4.3.

If we have a yes-instance for Partition, then we can find a solution for the Normalized 3-Cut problem at most N. Let $I_3 = \{p+1\}$ and let $I_1, I_2 \subset \{1, \ldots, p\}$ be a partition such that $I_1 \cup I_2 = \{1, \ldots, p\}$ and $I_1 \cap I_2 = \emptyset$. We make a cut $\mathcal{S} = (S_1, S_2, S_3)$ such that $S_j = \{v_i \mid i \in I_j\} \cup \{u_j\}$ for $j \in \{1, 2, 3\}$. Now the normalized 3-cut can be calculated using the following ideas. Say without loss of generality that $I_1 = \{1, \ldots, r\}$. This means that $S_1 = \{v_1, \ldots, v_r\} \cup \{u_1\}$. Thus, all the edges that we cut in cut $\left(S_1, \bar{S_1}\right)$ and their weights are:

1. $w_1, \ldots, w_r$ from $v_1, \ldots, v_r$ to u_2,
2. $w_1, \ldots, w_r$ from $v_1, \ldots, v_r$ to u_3, and
3. $w_{r+1}, \ldots, w_{p+1}$ from u_1 to $v_{r+1}, \ldots, v_{p+1}$.

Note that with point 1 and 3, all the edge-weights w_1 to w_{p+1} are exactly cut once. This cut has a total value of $3D$. Define $x = \sum_{i \in I_1} w_i$ to be the sum of the weights of the edges of all the v_i that are chosen to be in S_1. Now with point 2 we cut exactly x. Furthermore, we can see that the inner-edges of S_1 now have value x. It follows that:

$$\frac{\text{cut}\left(S_1, \bar{S_1}\right)}{\text{vol}\left(S_1, V\right)} = \frac{3D + x}{3D + x + 2x} = \frac{3D + x}{3D + 3x}. \tag{10}$$

We do the same for S_2 and S_3. Let $y = \sum_{i \in I_2} w_i$ and $z = \sum_{i \in I_3} w_i$. Note that $x + y + z = 3D$ must hold, because one v_i can only be in one S_j. It follows that the normalized 3-cut can be calculated as follows:

$$\text{Ncut}_3\left(\mathcal{S}\right) = \frac{3D + x}{3D + 3x} + \frac{3D + y}{3D + 3y} + \frac{3D + z}{3D + 3z}. \tag{11}$$

Since $I_3 = \{p+1\}$, we have $z = D$. This gives us:

$$\text{Ncut}_3\left(\mathcal{S}\right) = \frac{3D + x}{3D + 3x} + \frac{3D + y}{3D + 3y} + \frac{4D}{6D}.$$

If $\mathcal{S}$ forms a yes-instance for a partition input, then $x = y = D$ holds and we will find a 3-diamond cut at most N:

$$\text{Ncut}_3\left(\mathcal{S}\right) = \frac{3D + D}{3D + 3D} + \frac{3D + D}{3D + 3D} + \frac{4D}{6D} = 3 \cdot \frac{4D}{6D} = 2 \leq N.$$

Now we show that if we have a solution to the NORMALIZED 3-CUT problem at most N, then we have a yes-instance for PARTITION. Let $I_1, I_2, I_3 \subset \{1, \ldots, p+1\} = I$ be a partition such that $I_1 \cup I_2 \cup I_3 = I$ and $I_1 \cap I_2 = \emptyset$, $I_1 \cap I_3 = \emptyset$ and $I_2 \cap I_3 = \emptyset$. Without loss of generality, we have three possible cuts, due to symmetry in the 3-diamond.

- **Option A:** $S_j = \{v_i \mid i \in I_j\} \cup \{u_j\}$ for $j \in \{1, 2, 3\}$.
- **Option B:**
  - $S_1 = \{v_i \mid i \in I_1\}$ and $I_1 \neq \emptyset$,
  - $S_2 = \{v_i \mid i \in I_2\} \cup \{u_1, u_2\}$ and
  - $S_3 = \{v_i \mid i \in I_3\} \cup \{u_3\}$.
- **Option C:**
  - $S_1 = \{v_i \mid i \in I_1\}$ and $I_1 \neq \emptyset$,
  - $S_2 = \{v_i \mid i \in I_2\}$ and $I_2 \neq \emptyset$ and
  - $S_3 = \{v_i \mid i \in I_3\} \cup \{u_1, u_2, u_3\}$.

First, we look at option B. The cut values can be calculated as follows:

- $\frac{\text{cut}(S_1, \bar{S_1})}{\text{vol}(S_1, V)} = \frac{3z}{3z}$, where $z = \sum_{i \in I_1} w_i$. Since S_1 only contains some v_i, we will cut all the three edges from every $v_i \in S_1$. Furthermore, because there are no edges between different v_i, we have no inner-edges, so the volume equals the cut value.
- $\frac{\text{cut}(S_2, \bar{S_2})}{\text{vol}(S_2, V)} = \frac{2(3D-x)+x}{2(3D-x)+x+4x} = \frac{6D-x}{6D+3x}$, where $x = \sum_{i \in I_2} w_i$. Since u_1 and u_2 are in S_2, all the vertices v_i for $i \in I_2$ have an inner edge to both u_1 and u_2. So we cut one times x from all the edges to u_3 and we cut the edges from v_i to u_1 and u_2 for $i \notin I_2$, which gives us $2(3D - x)$. The inner edges have total value $2x$, thus a volume of $4x$.
- $\frac{\text{cut}(S_3, \bar{S_3})}{\text{vol}(S_3, V)} = \frac{3D+y}{3D+3y}$, where $y = \sum_{i \in I_3} w_i$, which follows from (10).

This gives us a the following normalized 3-cut:

$$\text{Ncut}_3(\mathcal{S}) = \frac{6D - x}{6D + 3x} + \frac{3D + y}{3D + 3y} + \frac{3z}{3z}. \tag{12}$$

To find the possible solutions, we solve the following system. We try to find a cut value at most $2 = N$. Note that z should be bigger than zero, because we need to find a 3-cut. Therefore, S_1 must contain at least one v_i.

$$\begin{cases} \frac{6D-x}{6D+3x} + \frac{3D+y}{3D+3y} + \frac{3z}{3z} \leq 2 \\ x + y + z = 3D \\ x, y \geq 0,\ z > 0 \end{cases}$$

If we let WolframAlpha [1] solve this system, we see that no solutions exist. It follows that option B gives us a normalized 3-cut with a value bigger than N.

Secondly, we look at option C. The cut values can be calculated similar as for option B.

- $\frac{\text{cut}(S_1, \bar{S_1})}{\text{vol}(S_1, V)} = \frac{3y}{3y}$, where $y = \sum_{i \in I_1} w_i$.
- $\frac{\text{cut}(S_2, \bar{S_2})}{\text{vol}(S_2, V)} = \frac{3z}{3z}$, where $z = \sum_{i \in I_2} w_i$.
- $\frac{\text{cut}(S_3, \bar{S_3})}{\text{vol}(S_3, V)} = \frac{3(3D-x)}{3(3D-x)+6x} = \frac{9D-3x}{9D+3x}$, where $x = \sum_{i \in I_3} w_i$.

This gives us a the following normalized 3-cut:

$$\text{Ncut}_3(\mathcal{S}) = \frac{9D - 3x}{9D + 3x} + \frac{3y}{3y} + \frac{3z}{3z}. \tag{13}$$

We want to solve the following system. Again we search for a solution where the cut value is at most $2 = N$. Note that y and z should be bigger than zero, because we need to find a 3-cut.

$$\begin{cases} \frac{9D-3x}{9D+3x} + \frac{3y}{3y} + \frac{3z}{3z} \leq 2 \\ x + y + z = 3D \\ x \geq 0,\ y, z > 0 \end{cases}$$

It follows that $\frac{9D-3x}{9D+3x} \leq 0$ and $x < 3D$, since $y, z > 0$. Let $x = 3D - \delta$ with $\delta > 0$. Then

$$\frac{9D - 3x}{9D + 3x} = \frac{9D - 3(3D - \delta)}{9D + 3(3D - \delta)} = \frac{3\delta}{18D - 3\delta} > 0,$$

shows us a contradiction. So, option C gives us a cut with a value bigger than N.

Finally, we have option A. We take the cut value as written in (11) and we solve the following system.

$$\begin{cases} \frac{3D+x}{3D+3x} + \frac{3D+y}{3D+3y} + \frac{3D+z}{3D+3z} \leq 2 \\ x + y + z = 3D \\ x, y, z \geq 0 \end{cases}$$

The solution $x = y = z = D$ gives us a cut value at most N. Note that edge w_{p+1} has edge-weight D. Without loss of generality let $I_3 = \{p + 1\}$. Now we have vertices $v_1, \ldots, v_p$ to divide over I_1 and I_2. Since we have that $x = y = D$, it follows that I_1 and I_2 form a partition for $v_1, \ldots, v_p$. Thus, a solution for the normalized 3-cut implies a yes-instance for the PARTITION problem.

Since we now have a solution for PARTITION if and only if we have a solution for the NORMALIZED 3-CUT problem with value at most N, this proof is complete. ■

6.1.2 Normalized 4-Cut

Similar as for the NORMALIZED 3-CUT we can prove that the NORMALIZED 4-CUT is NP-complete for vertex cover number equal to 4. We use a 4-diamond graph, which is the 3-diamond graph as in Figure 3 expanded with u_4, v_{p+2} and corresponding edges.

Theorem 6.3. *The* NORMALIZED 4-CUT *problem is NP-complete for graphs with vertex cover number equal to 4.*

Proof. For this proof we will look at the decision variant of the NORMALIZED 4-CUT problem. We ask ourselves if we can find a normalized 4-cut with value at most N.

First, note that this problem is in NP. We prove NP-completeness using a 4-diamond graph and a reduction from PARTITION, see Definition 4.2. We make an instance of the 4-diamond graph. Let $G = (V, E)$ be our 4-diamond graph. Let $v_1, \ldots, v_{p+2}$ be the vertices in the middle and add four vertices $u_1, \ldots, u_4$. Now let $w_i = c_i$ be the weight of the edges between v_i and u_j, for $1 \leq i \leq p$ and $1 \leq j \leq 4$. Let $w_{p+1} = w_{p+2} = D$ be the value of the edges between v_i and u_j, for $i = \{p + 1, p + 2\}$ and $1 \leq j \leq 4$. It follows that the total sum of the edge weights is $W = \sum_{i=1}^{p+2} w_i = 16D$. We choose $N = 3$.

If we have a yes-instance for PARTITION, then we can find a solution for the NORMALIZED 4-CUT problem at most N. Let $I_3 = \{p+1\}$, $I_4 = \{p+2\}$ and let $I_1, I_2 \subset \{1, \ldots, p\}$ be a partition such that $I_1 \cup I_2 = \{1, \ldots, p\}$ and $I_1 \cap I_2 = \emptyset$. We make a cut $\mathcal{S} = (S_1, \ldots, S_4)$ such that $S_j = \{v_i \mid i \in I_j\} \cup \{u_j\}$ for $1 \leq j \leq 4$. Now the normalized 4-cut can be calculated using the same ideas as described in the proof of Theorem 6.2 for the normalized 3-cut. Let $x = \sum_{i \in I_1} w_i$ and $y = \sum_{i \in I_2} w_i$. For I_3 and I_4 we have a value of D. We get:

$$\mathrm{Ncut}_4(\mathcal{S}) = \frac{4D+2x}{4D+4x} + \frac{4D+2y}{4D+4y} + \frac{6D}{8D} + \frac{6D}{8D}.$$

If $\mathcal{S}$ forms a yes-instance for a partition input, then $x = y = D$ holds and we will find a normalized 4-cut with value at most N:

$$\mathrm{Ncut}_4(\mathcal{S}) = \frac{4D+2D}{4D+4D} + \frac{4D+2D}{4D+4D} + \frac{6D}{8D} + \frac{6D}{8D} = 4 \cdot \frac{6D}{8D} = 3 \leq N.$$

Now we show that if we have a solution to the NORMALIZED 4-CUT problem, then we have a yes-instance for PARTITION. Let $I_1, \ldots, I_4 \subset \{1, \ldots, p+2\} = I$ be a partition such that $\bigcup_{1 \leq j \leq 4} I_j = I$ and $I_i \cap I_j = \emptyset$ for $1 \leq i, j \leq 4$ and $i \neq j$. We have five possible cuts, due to symmetry in the diamond. Let $\{v_i \mid i \in I_j\} \subseteq S_j$ for $1 \leq j \leq 4$. Now we have to choose in which S_j we put the u_j.

- **Option I:** $u_j \in S_j$ for $1 \leq j \leq 4$.
- **Option II:** $u_1, u_2 \in S_1$, $u_3 \in S_2$, $u_4 \in S_3$ and $I_4 \neq \emptyset$.
- **Option III:** $u_1, u_2 \in S_1$, $u_3, u_4 \in S_2$ and $I_3, I_4 \neq \emptyset$.
- **Option IV:** $u_1, u_2, u_3 \in S_1$, $u_4 \in S_2$ and $I_3, I_4 \neq \emptyset$.
- **Option V:** $u_j \in S_1$ for $1 \leq j \leq 4$ and $I_2, I_3, I_4 \neq \emptyset$.

The cut values for all these options can be composed, following the same reasoning as for the 3-diamond graph. We use the variables x, y, z, w for the sum of the edges of vertices that are in I_1, I_2, I_3, I_4 respectively. For option II-V we have the following systems:

$$II: \begin{cases} \frac{8D}{8D+4x} + \frac{4D+2y}{4D+4y} + \frac{4D+2z}{4D+4z} + \frac{4w}{4w} \leq 3 \\ x+y+z+w = 4D \\ x, y, z \geq 0,\ w > 0 \end{cases} \qquad III: \begin{cases} \frac{8D}{8D+4x} + \frac{8D}{8D+4y} + \frac{4z}{4z} + \frac{4w}{4w} \leq 3 \\ x+y+z+w = 4D \\ x, y \geq 0,\ z, w > 0 \end{cases}$$

$$IV: \begin{cases} \frac{12D-2x}{12D+4x} + \frac{4D+2y}{4D+4y} + \frac{4z}{4z} + \frac{4w}{4w} \leq 3 \\ x+y+z+w = 4D \\ x, y \geq 0,\ z, w > 0 \end{cases} \qquad V: \begin{cases} \frac{16D-4x}{16D+4x} + \frac{4y}{4y} + \frac{4z}{4z} + \frac{4w}{4w} \leq 3 \\ x+y+z+w = 4D \\ x \geq 0,\ y, z, w > 0 \end{cases}$$

If we let WolframAlpha [1] solve these systems, we see that no solutions exist. It follows that these options give us a cut with a value bigger than N.

Finally, we have option I. We solve the following system.

$$\begin{cases} \frac{4D+2x}{4D+4x} + \frac{4D+2y}{4D+4y} + \frac{4D+2z}{4D+4z} + \frac{4D+2w}{4D+4w} \leq 3 \\ x+y+z+w = 4D \\ x, y, z, w \geq 0 \end{cases}$$

The solution $x = y = z = w = D$ gives us a cut value at most N. Since $w_{p+1} = w_{p+2} = D$ we have that the $v_1, \ldots, v_p$ form a partition which both sum up to value D. So, we have a

yes-instance for partition. Thus, a solution for the normalized 4-cut implies a yes-instance for the PARTITION problem.

Since we now have a solution for PARTITION if and only if we have a solution for the NORMALIZED 4-CUT problem with value at most N, this proof is complete. ■

6.1.3 Normalized k-Cut

Since we have seen that the NORMALIZED k-CUT problem is NP-complete for vertex cover k for $k = 2, 3, 4$ (see Theorems 4.3, 6.2 and 6.3) we assume that this will also work for the generalized k-cut. We see that whenever we have a fraction $\frac{ax}{ax}$, with a is a certain integer and x is the sum of the edge weights in one of the S_j's, then we have no solution at most N. We expect this to be the same for a k-cut. However, the number of cases grows rapidly, so making an exhaustive case-based proof is difficult. Therefore, this remains an open problem.

The k-diamond looks like the diamond as shown in Figure 3, but then extended to a graph with vertex cover number equal to k. We now have the vertices $u_1, \ldots, u_k$ connected with the vertices $v_1, \ldots, v_{p+k-2}$ in the middle. Vertex v_i has an edge of weight w_i to every u_j for $1 \leq i \leq p + k - 2$ and $1 \leq j \leq k$, where $w_{p+1} = \cdots = w_{p+k-2} = D$.

Conjecture 6.4. *For any fixed $k \geq 2$, the* NORMALIZED k-CUT *problem is NP-complete for graphs with vertex cover number equal to k.*

We will prove that if we have a yes-instance for PARTITION, that we have a solution for the NORMALIZED k-CUT problem at most N.

We make an instance of the k-diamond graph. Let $v_1, \ldots, v_{p+k-2}$ be the vertices in the middle and add k vertices $u_1, \ldots, u_k$. Now let $w_i = c_i$ be the weight of the edges between v_i and U_j, for $1 \leq i \leq p$ and $1 \leq j \leq k$ and let $w_i = D$ for the weight of the edges between v_i and u_j for $p + 1 \leq i \leq p + k - 2$ and $1 \leq j \leq k$. It follows that the total sum of the edge weights is $W = \sum_{i=1}^{p+k-2} w_i = k^2 D$. We choose $N = k - 1$.

If we have a yes-instance for PARTITION, then we can find a solution for the NORMALIZED k-CUT problem at most N. Let $I_3 = \{p + 1\}, I_4 = \{p + 2\}, \ldots, I_k = \{p + k - 2\}$ and let $I_1, I_2 \subset \{1, \ldots, p\}$ be a partition such that $I_1 \cup I_2 = \{1, \ldots, p\}$ and $I_1 \cap I_2 = \emptyset$. We make a cut $\mathcal{S} = (S_1, \ldots, S_k)$ such that $S_j = \{v_i \mid i \in I_j\} \cup \{U_j\}$ for $1 \leq j \leq k$.

For the NORMALIZED k-CUT we get the following general formula:

$$\mathrm{Ncut}_k(\mathcal{S}) = \sum_{j=1}^{k} \frac{kD + (k-2)x_j}{kD + kx_j}, \tag{14}$$

where $x_j = \sum_{i \in I_j} w_i$, such that $\sum_{j=1}^{k} x_j = kD$. Since we defined $I_3, \ldots, I_k$ such that $\sum_{i \in I_j} w_i = D$, for all $3 \leq j \leq k$, we get:

$$\mathrm{Ncut}_k(\mathcal{S}) = \frac{kD + (k-2)x_1}{kD + kx_1} + \frac{kD + (k-2)x_2}{kD + kx_2} + (k-2)\frac{(2k-2)D}{2kD}.$$

If we have a yes-instance for PARTITION, it holds that $x_1 = x_2 = D$. So we get for the normalized k-cut:

$$\mathrm{Ncut}_k(G) = k \cdot \frac{(2k-2)D}{2kD} = k - 1 \leq N.$$

6.1.4 Max Normalized k-Cut

Below we will prove that the MAX NORMALIZED k-CUT is NP-complete for vertex cover number equal to k. This proof looks similar to the proof for the NORMALIZED 3-CUT problem in Section 6.1.1. Again we use a k-diamond graph, as described in Section 6.1.3.

Theorem 6.5. *For any fixed $k \geq 2$, the* MAX NORMALIZED k-CUT *problem is NP-complete for graphs with vertex cover number equal to k.*

Proof. We will look at the decision variant of the MAX NORMALIZED k-CUT problem. We will ask ourselves if we can find a max normalized k-cut with value at most N.

First, note that this problem is in NP. We will prove NP-completeness using k-diamond graph and a reduction from PARTITION, see Definition 4.2. We make an instance of the k-diamond graph. Let $v_1, \ldots, v_{p+k-2}$ be the vertices in the middle and add k vertices $u_1, \ldots, u_k$. Now let $w_i = c_i$ be the weight of the edges between v_i and u_j, for $1 \leq i \leq p$ and $1 \leq j \leq k$ and let $w_i = D$ for the weight of the edges between v_i and u_j for $p+1 \leq i \leq p+k-2$ and $1 \leq j \leq k$. It follows that the total sum of the edge weights is $W = \sum_{i=1}^{p+k-2} w_i = k^2 D$. We choose $N = \frac{k-1}{k}$.

If we have a yes-instance for PARTITION, then we can find a solution for the MAX NORMALIZED k-CUT problem at most N. Let $I_3 = \{p+1\}, I_4 = \{p+2\}, \ldots, I_k = \{p+k-2\}$ and let $I_1, I_2 \subset \{1, \ldots, p\}$ be a partition such that $I_1 \cup I_2 = \{1, \ldots, p\}$ and $I_1 \cap I_2 = \emptyset$. We make a cut $\mathcal{S} = (S_1, \ldots, S_k)$ such that $S_j = \{v_i \mid i \in I_j\} \cup \{u_j\}$ for $1 \leq j \leq k$. From (14) follows that:

$$\mathrm{Ncut}_k^{\max}(\mathcal{S}) = \max_{1 \leq j \leq k} \frac{kD + (k-2)x_j}{kD + kx_j}, \tag{15}$$

where $x_j = \sum_{i \in I_j} w_i$, such that $\sum_{j=1}^{k} x_j = kD$. Since we defined $I_3, \ldots, I_k$ such that $\sum_{i \in I_j} w_i = D$, for all $3 \leq j \leq k$ we get:

$$\mathrm{Ncut}_k^{\max}(\mathcal{S}) = \max\left\{ \frac{kD + (k-2)x_1}{kD + kx_1}, \ \frac{kD + (k-2)x_2}{kD + kx_2}, \ \frac{(2k-2)D}{2kD} \right\}.$$

Since we have a yes-instance for PARTITION, it holds that $x_1 = x_2 = D$. So we get the following:

$$\mathrm{Ncut}_k^{\max}(G) = \frac{(2k-2)D}{2kD} = \frac{k-1}{k} \leq N. \tag{16}$$

Now we show that if we have a solution for the MAX NORMALIZED k-CUT problem with value at most N, then we have a yes-instance for PARTITION. Let $\{1, \ldots, p+k-2\} = I$ and let $I_1, \ldots, I_k \subset I$ be a partition such that $\bigcup_{1 \leq j \leq k} I_j = I$ and $I_i \cap I_j = \emptyset$ for all $1 \leq i, j \leq k$ and $i \neq j$.

If $S_j = I_j$, for some $1 \leq j \leq k$, then S_j only contains independent vertices. Therefore, we have a volume that equals the cut. It follows that we have a cut-vale of 1. Since $N = \frac{k-1}{k}$ we have no solution to the k-diamond at most N. Thus, we know that it must hold that $u_j \in S_j$ for all $j \in \{1, \ldots, k\}$. So, the solution looks like $S_j = \{v_i \mid i \in I_j\} \cup u_j$ for $1 \leq j \leq k$. The cut value equals (15):

$$\mathrm{Ncut}_k^{\max}(\mathcal{S}) = \max_{1 \leq j \leq k} \frac{kD + (k-2)x_j}{kD + kx_j},$$

where $x_j = \sum_{i \in I_j} w_i$, such that $\sum_{j=1}^{k} x_j = kD$. In (16) we have seen that $x_i = D$ for all $1 \leq i \leq k$ is a solution for this k-diamond. Since $w_{p+1} = \cdots = w_{p+k-2} = D$ we only have to

divide the vertices $v_1, \ldots, v_p$ over two cut-sets. Since the value of these cut-sets equals D, we have found a partition. It follows that we have a yes-instance for PARTITION.

Since we now have a solution for PARTITION if and only if we have a solution for the MAX NORMALIZED k-CUT problem with value at most N, this proof is complete. ■

6.2 Treewidth

In this Section we give algorithms parameterized by treewidth and the total sum of the edge-weigths. Since we have seen the strong relation with PARTITION it seems we are bound to the total sum of the edge-weights. For the definition of treewidth and (nice) tree decomposition see Section 4.3.

6.2.1 Normalized k-Cut

Below we will give an algorithm for the NORMALIZED k-CUT, based on the algorithm for the unweighted variant in Section 5.1.1.

Theorem 6.6. *For every fixed integer $k \geq 2$, the* NORMALIZED k-CUT *problem can be solved in $O\left(k^{tw}2^{3k}W^{4k}n\right)$ time and in $O\left(k^{tw}2^{k}W^{2k}n\right)$ space, where tw is the treewidth of the input graph and W is the total sum of the edge-weights.*

Proof. This proofs follows the proof of Theorem 5.1. We will use a dynamic program which fills a table postorder, i.e. bottom-up. We use a nice tree decomposition. Let tw be the treewidth of input graph $G = (V, E)$ with weighted edges.

For each $i \in V(T)$ we define the following notation. Define $\mathcal{S} = (S_1, \ldots, S_k)$ to be a k-partition of V_i and let it be a solution for the NORMALIZED k-CUT problem on G_i. Let $c \in \{1, \ldots, k\}^{|X_i|}$ be a configuration vector, where $c[u] = j$ if and only if $u \in X_i \cap S_j$. To keep track of the value of the inner-edges and the outgoing edges we introduce the vectors $e_{in} \in \{0, \ldots, 2W\}^k$ and $e_{out} \in \{0, \ldots, W\}^k$, where $e_{in}[j] = w(S_j, S_j)$ and $e_{out}[j] = w(S_j, \bar{S_j})$.

We make a similar looking table as for the unweighted variant. We have $|V(T)|$ rows and at most $k^{tw}(2W)^k W^k = k^{tw}2^k W^{2k}$ columns. A cell in the table $\text{cut}[i, c, e_{in}, e_{out}]$ is equal to 1 if and only if there is a k-partition $(S_1, \ldots, S_k)$ of $V(G_i)$ such that for each $j \in \{1, \ldots, k\}$, it holds that $S_j \cap X_i = \{u \in X_i \mid c[u] = j\}$, $w(S_j, S_j) = e_{in}[j]$ and $w(S_j, \bar{S_j}) = e_{out}[j]$. If no such partition exists, then $\text{cut}[i, c, e_{in}, e_{out}] = 0$.

Remark that the only difference with the unweighted case is that the edge weights are not necessarily 1 in this problem. Since we already worked with the defined weight function, the algorithm will still work for the weighted variant. Only the running time will change a bit, because of the weights. So, we fill the table bottom-up using the recursive relations as defined in the proof of Theorem 5.1.

The table has size $O\left(k^{tw}2^{k}W^{2k}n\right)$. To initialize the leaf nodes we need constant time. Both the introduce and forget node cost $O(k)$ time. The join node will take $O\left((2W)^{2k}\right)$ time. We can find the minimum normalized k-cut of the graph in $O\left(k^{tw}(2W)^{2k}\right)$. This results in an algorithm that runs in $O\left(k^{tw}2^{3k}W^{4k}n\right)$ time. ■

6.2.2 Max Normalized k-Cut

Below we will give an algorithm for the MAX NORMALIZED k-CUT, based on the algorithm for the unweighted variant in Section 5.1.2 and for the normalized k-cut in Section 6.2.1.

Theorem 6.7. *For every fixed integer $k \geq 2$, the* Max Normalized k-Cut *problem can be solved in $O\left(k^{tw}2^{3k}W^{4k}n\right)$ time and in $O\left(k^{tw}2^{k}W^{2k}n\right)$ space, where tw is the treewidth of the input graph and W is the total sum of the edge-weights.*

Proof. Since we showed in Theorem 5.2 that for the Unweighted Max Normalized k-Cut problem we can use the same table as for the Unweighted Normalized k-Cut, and in Theorem 6.6 that we can use the same algorithm for the Normalized k-Cut problem as for the Unweighted Normalized k-Cut problem, we only need to change the calculation of the cut.

When the table is filled, we can find the minimum max normalized k-cut by checking all the table instances for the root r: $\text{cut}\,[r, c, e_{in}, e_{out}]$. If the instance is equal to 0, then we define the max normalized k-cut to be $+\infty$. If the instance is equal to 1, we can calculate the max normalized k-cut as follows:

$$\text{Ncut}_k^{\max}(\mathcal{S}) = \max_{1\leq j\leq k} \frac{e_{out}[j]}{e_{out}[j] + e_{in}[j]}.$$

We can find the minimum max normalized k-cut of the given graph by calculating:

$$\text{Ncut}_k^{\max}(G) = \min_{\mathcal{S}} \text{Ncut}_k^{\max}(\mathcal{S}).$$

So, as the Normalized k-Cut problem, we have an algorithm with a $O\left(k^{tw}2^{3k}W^{4k}n\right)$ running time and $O\left(k^{tw}2^{k}W^{2k}n\right)$ space usage. ■

6.3 Vertex Cover

Since a bounded vertex cover implies bounded treewidth, we can solve the problem for vertex cover with the treewidth algorithm. However, we will improve the running time and the space of the treewidth algorithm. To show this we will use the upper-bound $tw = \tau$.

We have an $O\left(k^{tw}2^{3k}W^{4k}n\right)$ time algorithm for treewidth and an $O\left(k^{\tau+1}n^{k+1}(3W)^k\right)$ time algorithm for vertex cover. Since we want the vertex cover algorithm to be an improvement of the treewidth algorithm, and we assumed that $k^{tw} = k^{\tau}$, we want $8^kW^{3k} > k \cdot n^k 3^k$ to hold. We assume that we have a connected graph. Furthermore, k is a chosen (small) number. With this, we get $W^{3k} > n^k$, and we see that the vertex cover algorithm will indeed be faster. Moreover, the treewidth algorithm uses $O\left(k^{tw}2^{k}W^{2k}n\right)$ space and the vertex cover algorithm uses $O\left(n^{k+1}(3W)^k\right)$ space, which is clearly an improvement.

6.3.1 Max Normalized k-Cut

First we give an algorithm for the Max Normalized k-Cut, based on the algorithm for the unweighted variant in Section 5.2.1.

Theorem 6.8. *For every fixed integer $k \geq 2$, the* Max Normalized k-Cut *problem can be solved in $O\left(k^{\tau+1}n^{k+1}(3W)^k\right)$ time and in $O\left(n^{k+1}(3W)^k\right)$ space, where τ is the vertex cover number of the input graph and W is the total sum of the edge-weights.*

Proof. This proof follows the same steps as the proof of Theorem 5.5. We will use a dynamic program to solve the decision variant of the Max Normalized k-Cut problem, i.e. for a given graph G and rational number N does $\text{Ncut}_k(G) \leq N$ hold?

Let τ be the vertex cover number of input graph $G = (V, E)$ with weighted edges. Let C be a vertex cover of G with $|C| = \tau$ and let $I = V \setminus C$ be the independent set. If a vertex cover is not given, we can find an optimal vertex cover in $O(2^{\tau}n)$ time [11]. For every

k-partition $(C_1, \ldots, C_k)$ of C we aim to assign the vertices in I to a C_j such that the max normalized k-cut will be at most N.

Remark that we now work with edge-weights that are not necessarily equal to 1. This will only influence the values that the variables can have, but not on the formulas as we designed in Theorem 5.5, since the cut will exist of the same edges. Again, the edge-count formula E and the weight-count formula w are equal. It follows that the cut value for S_j can be calculated as:

$$\frac{\operatorname{cut}\left(S_j, \bar{S}_j\right)}{\operatorname{vol}\left(S_j, V\right)} = \frac{w(C_j, V \setminus C_j) + w(A_j, C) - 2w(A_j, C_j)}{w(C_j, V \setminus C_j) + w(C_j, C_j) + w(A_j, C)}.$$

Again, we define $W_j = w(C_j, V \setminus C_j)$ and $c_j = w(C_j, C_j)$ and for every vertex $i \in I$: $x_i = w(i, C)$ and $y_{ij} = w(i, C_j)$. Now we can rewrite the cut value to:

$$(1 - N) \sum_{i \in A_j} x_i - 2 \sum_{i \in A_j} y_{ij} \leq (N - 1) W_j + N c_j$$

Solving this we use a variant of the bin packing problem as in Problem 5.4. We define $b_j = (N - 1) W_j + N c_j$.

Now, the algorithm for the MAX NORMALIZED k-CUT looks the same as for the UNWEIGHTED MAX NORMALIZED k-CUT problem, with as difference that some variables have more possible values. We get the following algorithm.

For every k-partition $(C_1, \ldots, C_k)$ of C we will solve the bin packing variant as described in Problem 5.4. Where we use $a_j = n$ as input for all $1 \leq j \leq k$. When **Step III** returns a 1, there is a possible solution. We can stop the algorithm and we are done. If for all k-partitions of C **Step III** returns a 0, then we cannot find a max nomalized k-cut with a value at most N.

Step I: We solve the bin packing variant with a dynamic program. Let $a'_1, \ldots, a'_k$ and $b'_1, \ldots, b'_k$ be integers such that $0 \leq a'_j \leq n$ and $0 \leq b'_j \leq b_j$ for every $1 \leq j \leq k$. Note that $\max_{1 \leq i \leq |I|} b_i \leq 3W$. We fill a table with $|I|$ rows and at most $(n \cdot 3W)^k$ columns.

A cell of this table $T\left(i; a'_1, \ldots, a'_k; b'_1, \ldots, b'_k\right)$ equals 1 if and only if there exists a k-partition $(A'_1, \ldots, A'_k)$ of $\{1, \ldots, i\}$ such that for each $j \in \{1, \ldots, k\}$ it holds that $|A'_j| = a'_j$ and $(1 - N) \sum_{i \in A'_j} x_i - 2 \sum_{i \in A'_j} y_{ij} \leq b'_j$. Otherwise, $T\left(i; a'_1, \ldots, a'_k; b'_1, \ldots, b'_k\right) = 0$.

Step II: The recurrence relation looks as follows. We have

$$T\left(i; a'_1, \ldots, a'_k; b'_1, \ldots, b'_k\right) = 1$$

if and only if for a certain $s \in \{1, \ldots, k\}$

$$T\left(i - 1; a'_1, \ldots, a'_s - 1, \ldots, a'_k; b'_1, \ldots, b'_s - z, \ldots, b'_k\right) = 1,$$

where $z = (1 - N) x_i - 2 y_{i,s}$.

The base case is: $T\left(1; a'_1, \ldots, a'_k; b'_1, \ldots, b'_k\right)$ equals 1 if and only if there is a $s \in \{1, \ldots, k\}$ such that $a'_s = 1$ and $(1 - N) x_1 - 2 y_{1,s} \leq b'_s$, and for all $j \in \{1, \ldots, k\} \setminus \{s\}$ we have $a'_j = 0$. Otherwise, it would be equal to 0.

Step III: We return

$$\max_{a_1 + \cdots + a_k = |I|} T\left(I; a_1, \ldots, a_k; b_1, \ldots, b_k\right).$$

If this value is equal to 1, then it is possible to find a max normalized k-cut of G for the given k-partition of C and some sum $a_1 + \cdots + a_k = |I|$.

We have k^τ possible k-partitions of C. The table we will fill in Step I is of size $n^{k+1}(3W)^k$. We need $O(k)$ time to calculate every cell in Step II. In Step III we read $O\left(n^k\right)$ possible solutions from the table. This results in a $O\left(k^{\tau+1}n^{k+1}(3W)^k\right)$ time algorithm. Moreover, we use $O\left(n^{k+1}(3W)^k\right)$ space. ■

6.3.2 Normalized k-Cut

Below we will give an algorithm for the Normalized k-Cut, based on the algorithm for the unweighted variant in Section 5.2.2 and on the proof of Theorem 6.8 for the max variant. This algorithm uses $O\left(2^kW^{2k}\right)$ more time than the same algorithm for the max variant. This algorithm will still be an improvement relative to the treewidth algorithm.

Theorem 6.9. *For every fixed integer $k \geq 2$, the* Normalized k-Cut *problem can be solved in $O\left(k^{\tau+1}n^{k+1}6^kW^{3k}\right)$ time and in $O\left(n^{k+1}(3W)^k\right)$ space, where τ is the vertex cover number of the input graph.*

Proof. This proof follows the proof of Theorem 5.3. We will use the same dynamic program as for the Max Normalized k-Cut problem to solve the decision variant of the Normalized k-Cut problem, i.e. for a given graph G and rational number N does $\text{Ncut}_k(G) \leq N$ hold?

We ask ourselves for every $1 \leq j \leq k$ if $\frac{\text{cut}\left(S_j,\bar{S}_j\right)}{\text{vol}(S_j,V)} \leq N_j$ such that $N_1 + \cdots + N_k = N$. Then we can follow the same algorithm as for the max variant of this problem. These N_j will be a fraction between 0 and 1. For the numerator, which represents the cut value, we have W options and for the denominator, which represents the volume, we have $2W$ options. In total this gives us $2W^2$ options for every N_j, thus 2^kW^{2k} options for the sum up to N.

Now we run the algorithm with $O\left(k^{\tau+1}n^{k+1}(3W)^k\right)$ time $O\left(2^kW^{2k}\right)$ times more, which results in a $O\left(k^{\tau+1}n^{k+1}6^kW^{3k}\right)$ time algorithm. Furthermore, the space stays the same, so this algorithm uses $O\left(n^{k+1}(3W)^k\right)$ space. ■

7 Conclusion

In this thesis we have proven several variants of the Normalized Cut problem to be NP-complete and fixed parameter tractable for treewidth, the vertex cover number and/or the total sum of the edge weights.

The Normalized Cut, Max Normalized 2-Cut, Normalized 3-Cut, Normalized 4-Cut and the Max Normalized k-Cut problems are NP-complete. These NP-completeness proofs use a reduction from the Partition problem. For the Normalized k-Cut we have conjectured its hardness, because an exhaustive case-based proof would become quite big for this problem.

Since the difficulty of the Partition problem strongly relates with the weights of the elements, this indicates that the weights of the input graph will have influence on the running time of the designed algorithms. This is reflected by the running times of our designed algorithms. For the Normalized (k-)Cut and the Max Normalized $2/k$-Cut problems we have given algorithms parameterized by treewidth or the vertex cover number. In all the algorithms we have seen the total sum of the edge weights being part of the running time and space usage. For the unweighted variant of the same problems we have seen that not the total sum of the edge-weights, but the number of edges is part of the running time. For the overview of the algorithm running times, see Table 6 and 7.

It is good to notice that the algorithms for the 2-cut problems are really an improvement on the algorithms for the k-cut variants on both the running times and the space usage.

Table 6: The overview of algorithms designed in this thesis in the field of the normalized cut, with parameters: tw - the treewidth of the input graph and W - the total edge-weight of the graph.

Problem	Running time	Space usage
Ncut	$O\left(2^{tw}nW^2\right)$	$O\left(2^{tw}nW\right)$
$\text{Ncut}_2^{\max}$	$O\left(2^{tw}nW^2\right)$	$O\left(2^{tw}nW\right)$
Ncut_k for $k \geq 2$	$O\left(k^{tw}2^{3k}W^{4k}n\right)$	$O\left(k^{tw}2^kW^{2k}n\right)$
$\text{Ncut}_k^{\max}$ for $k \geq 2$	$O\left(k^{tw}2^{3k}W^{4k}n\right)$	$O\left(k^{tw}2^kW^{2k}n\right)$
Ncut_k for $k \geq 2$ (unweighted)	$O\left(k^{tw}2^km^{4k}n\right)$	$O\left(k^{tw}2^km^{2k}n\right)$
$\text{Ncut}_k^{\max}$ for $k \geq 2$ (unweighted)	$O\left(k^{tw}2^km^{4k}n\right)$	$O\left(k^{tw}2^km^{2k}n\right)$

Table 7: The overview of algorithms designed in this thesis in the field of the normalized cut, with parameters: τ - the vertex cover number and W - the total edge-weight of the graph.

Problem	Running time	Space usage
Ncut	$O\left(2^{\tau}nW\right)$	$O\left(nW\right)$
$\text{Ncut}_2^{\max}$	$O\left(2^{\tau}nW\right)$	$O\left(nW\right)$
Ncut_k for $k \geq 2$	$O\left(k^{\tau+1}n^{k+1}6^kW^{3k}\right)$	$O\left(n^{k+1}(3W)^k\right)$
$\text{Ncut}_k^{\max}$ for $k \geq 2$	$O\left(k^{\tau+1}n^{k+1}(3W)^k\right)$	$O\left(n^{k+1}(3W)^k\right)$
Ncut_k for $k \geq 2$ (unweighted)	$O\left(k^{\tau+1}n^{k+1}6^km^{3k}\right)$	$O\left(n^{k+1}(3m)^k\right)$
$\text{Ncut}_k^{\max}$ for $k \geq 2$ (unweighted)	$O\left(k^{\tau+1}n^{k+1}(3m)^k\right)$	$O\left(n^{k+1}(3m)^k\right)$

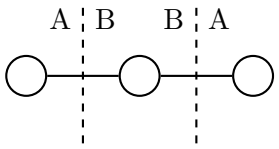
The results we found are based on the results of the Sparsest Cut problem by Javadi and Nikabadi [15]. The most important difference between the Normalized Cut problem and the Sparsest Cut problem is the denominator in both fractions. By the normalized cut we divide the cut by its volume, where the cut in the sparsest cut fraction is divided by the amount of vertices in its corresponding set. This makes the numerator and denominator of the Normalized Cut problem dependent on each other. Therefore, translating the results of the Sparsest Cut problem to the Normalized Cut problem required some extra tricks. As mentioned above the calculation of the normalized cut value strongly depends on the weights of the edges in a given graph, which follows from the NP reduction from the Partition problem. Therefore, we see in the weighted variants that the total sum of the edge weights is part of the running time of the given algorithms. For the Sparsest Cut we do not have this relation and we do not see the total sum of the edge weight being part of the running time.

We have seen many nice results, but there are enough open questions left in this field. The first question follows from our conjecture: *Is the* Normalized k-Cut *problem NP-complete for graphs with vertex cover number equal to k?* Furthermore, we have no hardness results for the Unweighted (Max) Normalized k-Cut problems. However, the Unweighted k-Sparsest Cut problem is NP-complete for unweighted (simple) graphs, for $k \geq 2$. So we believe the Unweighted (Max) Normalized k-Cut problems will be hard as well.

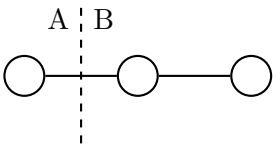
In the algorithms for the (Unweighted) Normalized k-Cut problems, parameterized by the vertex cover number, we have seen that there is a factor 2^kW^{2k} or 2^km^{2k} more than

the max variant of the same problem, for the weighted and unweighted variant respectively. This is caused by the fact that we use a decision variant of the problem and we have to check for given N for every possible sum $N_1 + \cdots + N_k = N$ if this problem has a solution. This extra factor is an upper bound and it may be reducible.

Another question that we did not find an answer to, is: *After cutting a graph, do the k parts form connected induced subgraphs?*. We could not find a counterexample, nor have we found a proof. In Figure 4 we see two possible cuts, one where the induced graphs after the cut are disconnected, and one where the induced graphs after the cut are connected. In the formula of the normalized cut we want to reduce the cut value and increase the volume. If we put two independent graph pieces together in a cut (as shown in Figure 4 (A)), we see the cut increase, while the volume does not. If we compare this with the cut in graph (B), the cut will be lower, while the volume of B will be higher. This indicates that it will not pay to put two independent graph pieces together. However, this does not guarantee that it is never better to have two independent pieces together. It remains an interesting open question.



(A) An example of a disconnected induced graph after cut $\{A, B\}$.



(B) An example of a connected induced graph after cut $\{A, B\}$.

Figure 4: Two examples of a normalized cut.

Moreover, in this thesis we have only presented algorithms and hardness results for general graphs. It may be interesting to examine if the Normalized k-Cut and the Max Normalized k-Cut problems are NP-complete on trees and other graph classes. Also, it may be possible that the algorithms can be improved for specific graph classes.

Another interesting question is, in which fields could the (Max) Normalized (k-)Cut be useful. If we know for which problems the normalized cut is helpful and what specific properties the graphs may have in this context, we may find new useful parameters for which we can check if this problem is FPT.

References

[1] `https://www.wolframalpha.com/`.

[2] Ralf Banisch and Péter Koltai. Understanding the geometry of transport: Diffusion maps for Lagrangian trajectory data unravel coherent sets. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(3):035804, 2017.

[3] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on Computing*, 25(6):1305–1317, 1996.

[4] Hans L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.

[5] Hans L. Bodlaender, Pål Grønås Drange, Markus S. Dregi, Fedor V. Fomin, Daniel Lokshtanov, and Michal Pilipczuk. An $O\left(c^k n\right)$ 5-approximation algorithm for treewidth.

In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 499–508. IEEE Computer Society, 2013.

[6] Paul S. Bonsma, Haitze J. Broersma, Viresh Patel, and Artem V. Pyatkin. The complexity status of problems related to sparsest cuts. In Costas S. Iliopoulos and William F. Smyth, editors, *Combinatorial Algorithms - 21st International Workshop, IWOCA 2010, London, UK, July 26-28, 2010, Revised Selected Papers*, volume 6460 of *Lecture Notes in Computer Science*, pages 125–135. Springer, 2010.

[7] Paul S. Bonsma, Haitze J. Broersma, Viresh Patel, and Artem V. Pyatkin. The complexity of finding uniform sparsest cuts in various graph classes. *Journal of Discrete Algorithms*, 14:136–149, 2012.

[8] NASA Scientific Visualization Studio YouTube channel. retrieved 24-11-2020, `https://www.youtube.com/channel/UCM2GOiW_Dxn1D7HHP8OIrBg`.

[9] Amir Daneshgar and Ramin Javadi. On the complexity of isoperimetric problems on trees. *Discrete Applied Mathematics*, 160(1-2):116–131, 2012.

[10] Xianwen Ding and Xiaofeng Li. Coastline detection in SAR images using multiscale normalized cut segmentation. In *2014 IEEE Geoscience and Remote Sensing Symposium, IGARSS 2014, Quebec City, QC, Canada, July 13-18, 2014*, pages 4447–4449. IEEE, 2014.

[11] Rodney G. Downey and Michael R. Fellows. Parameterized computational feasibility. *Feasible Mathematics II. Progress in Computer Science and Applied Logic*, 13:219–244, 1995.

[12] Neng Fan and Panos M. Pardalos. Multi-way clustering and biclustering by the ratio cut and normalized cut in graphs. *Journal of Combinatorial Optimization*, 23(2):224–251, 2012.

[13] Alireza Hadjighasem, Daniel Karrasch, Hiroshi Teramoto, and George Haller. Spectral-clustering approach to Lagrangian vortex detection. *Physical Review E*, 93:063107, June 2016.

[14] Ramin Javadi and Saleh Ashkboos. Multi-way sparsest cut problem on trees with a control on the number of parts and outliers. *CoRR*, abs/1702.05570, 2017.

[15] Ramin Javadi and Amir Nikabadi. On the parameterized complexity of sparsest cut and small-set expansion problems. *CoRR*, abs/1910.12353, 2019.

[16] Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[17] Twan van Laarhoven and Elena Marchiori. Local network community detection with continuous optimization of conductance and weighted kernel k-means. *Journal of Machine Learning Research*, 17:1–28, 2016.

[18] Bojan Mohar. Isoperimetric numbers of graphs. *Journal of Combinatorial Theory, Series B*, 47(3):274–291, 1989.

[19] NOAA/GFDL. Data visualizations – oceans. retrieved 24-11-2020, `https://www.gfdl.noaa.gov/visualizations-oceans/`.

[20] Kathrin Padberg-Gehle and Christiane Schneide. Network-based study of Lagrangian transport and mixing. *Nonlinear Processes in Geophysics*, 24(4):661–671, 2017.

[21] James K. Park and Cynthia A. Phillips. Finding minimum-quotient cuts in planar graphs. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 766–775. ACM, 1993.

[22] Syama Sundar Rangapuram, Pramod Kaushik Mudrakarta, and Matthias Hein. Tight continuous relaxation of the balanced k-cut problem. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3131–3139, 2014.

[23] Vincent Rossi, Enrico Ser-Giacomi, Cristóbal López, and Emilio Hernández-García. Hydrodynamic provinces and oceanic connectivity from a transport network help designing marine reserves. *Geophysical Research Letters*, 41:2883–2891, 2014.

[24] Enrico Ser-Giacomi, Vincent Rossi, Cristobal López, and Emilio Hernández-García. Flow networks: A characterization of geophysical fluid transport. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 25:036404, 2015.

[25] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. In *1997 Conference on Computer Vision and Pattern Recognition (CVPR '97), June 17-19, 1997, San Juan, Puerto Rico*, pages 731–737. IEEE Computer Society, 1997.

[26] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, Aug 2000.

[27] P. Soundararajan and S. Sarkar. An in-depth study of graph partitioning measures for perceptual organization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(6):642–660, 2003.

[28] Erik van Sebille, Stephen M. Griffies, Ryan Abernathey, Thomas P. Adams, Pavel Berloff, Arne Biastoch, Bruno Blanke, Eric P. Chassignet, Yu Cheng, Colin J. Cotter, Eric Deleersnijder, Kristofer Döös, Henri F. Drake, Sybren Drijfhout, Stefan F. Gary, Arnold W. Heemink, Joakim Kjellsson, Inga Monika Koszalka, Michael Lange, Camille Lique, Graeme A. MacGilchrist, Robert Marsh, C. Gabriela Mayorga Adame, Ronan McAdam, Francesco Nencioli, Claire B. Paris, Matthew D. Piggott, Jeff A. Polton, Siren Rühs, Syed H.A.M. Shah, Matthew D. Thomas, Jinbo Wang, Phillip J. Wolfram, Laure Zanna, and Jan D. Zika. Lagrangian ocean analysis: Fundamentals and practices. *Ocean Modelling*, 121:49–75, 2018.

[29] David Wichmann, Christian Kehl, Henk A. Dijkstra, and Erik van Sebille. Detecting flow features in scarce trajectory data using networks derived from symbolic itineraries: an application to surface drifters in the north atlantic. *Nonlinear Processes in Geophysics*, 27(4):501–518, 2020.

[30] Eric P. Xing and Michael I. Jordan. On semidefinite relaxation for normalized k-cut and connections to spectral clustering, June 2003. Report No. UCB/CSD-3-1265.

A Calculations of the counter example

We will give an example of the calculation for the normalized cut and the conductance of the graph in Figure 1 in Section 2. In Figure 6 we see all the calculations in one big table. Below we show one calculation.

abe	2
cut	16
cdf	6
Ncut(abe, cdf)	1,371
phi(abe, cdf)	0,800

Figure 5: An example calculation for the normalized cut and the conductance by the graph in Figure 1.

We look at Figure 5. In this example we have $S = \{a, b, e\}$ and $\bar{S} = \{c, d, f\}$. The value 2 after *abe* is the value of the inner edges of set S. The value 6 after *cdf* is the value of the inner edges of set $\bar{S}$. The value 16 after cut is the value of the edges in the cut.

The normalized cut is calculated as follows:

$$\begin{aligned}\text{Ncut}(abe, cdf) &= \frac{cut}{cut + 2 \cdot abe} + \frac{cut}{cut + 2 \cdot cdf} \\ &= \frac{16}{16 + 2 \cdot 2} + \frac{16}{16 + 2 \cdot 6} = 1.371.\end{aligned}$$

The conductance is calculated as follows:

$$\text{phi}(abe, cdf) = \frac{cut}{cut + 2 \cdot \min(abe, cdf)} = \frac{16}{16 + 2 \cdot \min(2, 6)} = 0.800.$$

In Figure 6 we see the calculations of all the possible cuts in the given graph in Figure 1. The cut values that are optimal for the given graph are green. The values between the optimal value and 1 are yellow/orange and the values of 1 or bigger are red.

a	0	b	0	c	0	d	0
cut	2	cut	3	cut	7	cut	9
bcdef	22	acdef	21	abdef	17	abcef	15
Ncut(a, bcdef)	1,043	Ncut(b, acdef)	1,067	Ncut(c, abdef)	1,171	Ncut(d, abcef)	1,231
phi(a, bcdef)	1,000	phi(b, acdef)	1,000	phi(c, abdef)	1,000	phi(d, abcef)	1,000
e	0	f	0	ab	2	ac	0
cut	15	cut	12	cut	1	cut	9
abcdf	9	abcde	12	cdef	21	bdef	15
Ncut(e, abcdf)	1,455	Ncut(f, abcde)	1,333	Ncut(ab, cdef)	0,223	Ncut(ac, bdef)	1,231
phi(e, abcdf)	1,000	phi(f, abcde)	1,000	phi(ab, cdef)	0,200	phi(ac, bdef)	1,000
ad	0	ae	0	af	0	bc	1
cut	11	cut	17	cut	14	cut	8
bcef	13	bcdf	7	bcde	10	adef	15
Ncut(ad, bcef)	1,297	Ncut(ae, bcdf)	1,548	Ncut(af, bcde)	1,412	Ncut(bc, adef)	1,011
phi(ad, bcef)	1,000	phi(ae, bcdf)	1,000	phi(af, bcde)	1,000	phi(bc, adef)	0,800
bd	0	be	0	bf	0	cd	6
cut	12	cut	18	cut	15	cut	4
acef	12	acdf	6	acde	9	abef	14
Ncut(bd, acef)	1,333	Ncut(be, acdf)	1,600	Ncut(bf, acde)	1,455	Ncut(cd, abef)	0,375
phi(bd, acef)	1,000	phi(be, acdf)	1,000	phi(bf, acde)	1,000	phi(cd, abef)	0,250
ce	0	cf	0	de	3	df	0
cut	22	cut	19	cut	18	cut	21
abdf	2	abde	5	abcf	3	abce	3
Ncut(ce, abdf)	1,846	Ncut(cf, abde)	1,655	Ncut(de, abcf)	1,500	Ncut(df, abce)	1,778
phi(ce, abdf)	1,000	phi(cf, abde)	1,000	phi(de, abcf)	0,750	phi(df, abce)	1,000
ef	12	abc	3	abd	2	abe	2
cut	3	cut	6	cut	10	cut	16
abcd	9	def	15	cef	12	cdf	6
Ncut(ef, abcd)	0,254	Ncut(abc, def)	0,667	Ncut(abd, cef)	1,008	Ncut(abe, cdf)	1,371
phi(ef, abcd)	0,143	phi(abc, def)	0,500	phi(abd, cef)	0,714	phi(abe, cdf)	0,800
abf	2	acd	6	ace	0	acf	0
cut	13	cut	6	cut	24	cut	21
cde	9	bef	12	bdf	0	bde	3
Ncut(abf, cde)	1,184	Ncut(acd, bef)	0,533	Ncut(ace, bdf)	2,000	Ncut(acf, bde)	1,778
phi(abf, cde)	0,765	phi(acd, bef)	0,333	phi(ace, bdf)	1,000	phi(acf, bde)	1,000
ade	3	adf	0	aef	12		
cut	20	cut	23	cut	5		
bcf	1	bce	1	bcd	7		
Ncut(ade, bcf)	1,678	Ncut(adf, bce)	1,920	Ncut(aef, bcd)	0,436		
phi(ade, bcf)	0,909	phi(adf, bce)	1,000	phi(aef, bcd)	0,263		

Figure 6: All the possible cuts for the graph in Figure 1 in Section 2 with the normalized cut value and the conductance value.