UCLouvain

École polytechnique de Louvain

Data sampling on graphs

Author: **Henri DEVILLEZ**
Supervisor: **Jean-Charles DELVENNE**
Readers: **Jean-Charles DELVENNE, Laurent JACQUES, Pierre-Antoine ABSIL**
Academic year 2018–2019
Master [120] in Mathematical Engineering

Acknowledgements

Firstly, I would like to thank my supervisor Pr. Jean-Charles Delvenne for its support and our insightful conversations. I am very grateful for its time and its ideas which greatly help me to guide my work.

Beside my advisor, my thanks goes to Pr. Pierre-Antoine Absil and Pr. Laurent Jacques for having accepted to read and assess my work. I also thank Dr. Michael Schaub for our exciting and fruitful interactions which made me want to work with cycle spaces and conservative flows.

Last but not least, I would like to express my gratitude to my family : my mother, my brother and my sister who showed support throughout the writing of this master thesis, despite my terrible mood following the few times I discovered flaws in my proves and reasoning.

Contents

Introduction

In our modern connected world, graphs are a powerful modeling tool : social network, Internet-Of-Things and autonomous cars are classic examples of applications. As industries thrive to gather more and more data for their best interests, collecting data on graphs is consequentially quite relevant. However, in the era of Big Data, it is often impossible to collect all the available data, and we have to restrict ourselves to only a small fraction. It is thus natural to find which observations allow us to recover a greater part of the data or are less prone to errors.

This master thesis is divided in two parts and four chapters.

In the first part, we will focus on signals defined over the nodes of a graph. As a sampling procedure for such signals has already been proposed in the *Graph Signal Processing* framework, we will start by reviewing the basic concepts of this framework. Then, we will attempt to model spatiotemporal signals and sample them by extending the existing techniques.

The second part, much more consequent than the first part, is about signals defined over the edges of a graph.

In the first chapter about edge signal sampling, we will extend the sampling theory to a common class of signals, conservative flows. When dealing with conservative flows, the concept of *cycle spaces* will be of an huge help, and we will thus take a moment to introduce this domain at the intersection of graph theory and algebra. Using tools of the cycle space theory, we are finally going to design and propose an effective algorithm to sample and recover such signals.

In the next chapter, an alternative sampling procedure called *cycle sampling* is introduced. As it may be difficult to directly observe the value of the signal of an edge in real-life applications, we will propose instead to observe

the agglomeration of the signal over a cycle of the graph. To better choose these cycles, we will then present an algorithm that takes advantage of the *matroid* structure of the cycle space.

Finally, our last chapter is about edge signal recovery from noisy observations. Since there are a lot of ways to choose edges that allow us to recover the signal from their observations, we will to try to design a procedure that minimizes the total impact of the noise.

Part I

Node sampling

Chapter 1

Node sampling through Graph Signal Processing

Graph Signal Processing (GSP) is an emergent field that aims to generalize and extend the widely established concepts of Signal Processing (operating in the 'time' domain) to networks (modeling 'space' domains). As sampling is a very important concept of signal processing, we will explore in this chapter how to translate this procedure from the very well studied time framework to the recent space framework. But before getting into the heart of the matter, it is necessary to review some basic concepts of graph signal processing.

1.1 Graph Signal Processing

In this section, we will review the building blocks of GSP. [Ort+18]

Let us formalize the setting. For a given undirected network $G = (V, \mathcal{E})$ of N nodes $\mathcal{V}$, we define a signal $x : \mathcal{V} \to \mathbb{C}$ that maps each node to a complex value. To make our notations simpler, we set an arbitrary indexing of the nodes from 1 to N, and represent the signal x as a vector of $\mathbb{C}^n$ whose i^{th} coordinate is the value of the signal of the i^{th} node. In the following subsections, we generalize and define some important concepts of signal processing to our graph framework.

1.1.1 Graph filters

We define a graph filter $\mathbf{H}(\cdot)$ as a a function taking a graph signal as input and producing another graph signal as output. A first relevant filter is the graph shift; for a graph with adjacency matrix $\mathbf{A}$ and given x as input, it produces $\tilde{x} = \mathbf{H}(x) = \mathbf{A}x$, ie $\tilde{x}_n = \sum_m \mathbf{A}_{n,m} x_m$. Taking the adjacency matrix as the graph shifting operator is quite arbitrary; other popular choices are the *Laplacian* of the network, defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$ with $\mathbf{D}$ the diagonal matrix whose elements are the degree of each vertex.

In particular, we often use linear shift-invariant graph filters. Linear filters are such that the image of a linear combination of inputs is the same linear combination of output of these inputs. We say that filters are shift-invariant if the result of the composition of these filters does not does depend on the order of processing of these filters. Interestingly, all linear shift-invariant graph filters are polynomials of the adjacency matrix, $\mathbf{H}(s) = h(\mathbf{A})s$ with $h(s) = \sum_{i=0}^{L} h_i A^i$.

1.1.2 Graph Fourier Transform

As we try to generalize signals to irregular domains, we would like to introduce some concept of *Fourier Transform* and frequency. Since Fourier basis are invariant to filtering by linear shift-invariant filters, it naturally corresponds to the eigenvectors of the graph shift operator, or the Jordan's eigenvectors if the matrix is not diagonalizable.

Definition 1. Let $\mathbf{A} = \mathbf{V}\mathbf{J}\mathbf{V}^{-1}$ be the Jordan's decomposition of the graph shift, with $\mathbf{J}$ the Jordan's form of $\mathbf{A}$ with eigenvalues sorted in ascending order and $\mathbf{V}$ the corresponding matrix of generalized eigenvectors. Then, the Graph Fourier Transform $\hat{s}$ of a graph signal s is defined as the coefficient of the signal in this Fourier basis :

$$\hat{s} = \mathbf{V}^{-1} s$$

and the inverse Fourier transform is respectively given by :

$$s = \mathbf{V}\hat{s}$$

As mentioned before, one common choice of the shift operator is the graph Laplacian, which is a positive semidefinite matrix and has thus a full set

of orthogonal eigenvectors. The spectrum of the Laplacian matrix and the structure of this eigenvector space is well studied; this is the object of study of the graph spectral theory. [Chu97] Frequency over a network, which is an inherently irregular structure, cannot be defined in a such elegant manner as in classical Discrete Signal Processing. However, the intuition behind the standard Fourier basis, a collection of orthogonal sinusoids, somewhat remains. The smallest eigenvalue of the Laplacian is zero and its corresponding eigenvector is constant; it may be interpreted as the DC component of the signal. On the other end, high eigenvalues correspond to highly oscillating eigenvector over the vertex space. This extension capture the concept of frequency in some way, as illustrated in Figure 1.1.

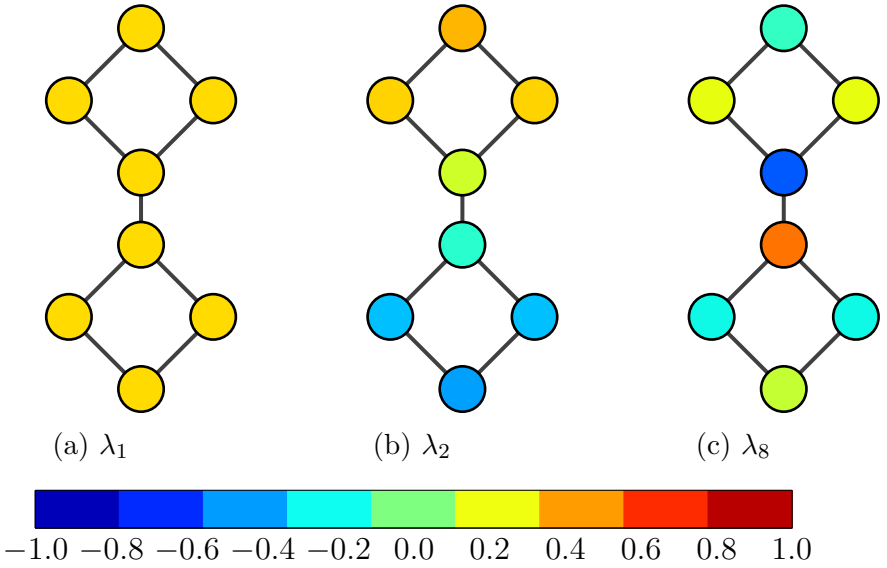


Figure 1.1: First, second and last element of the Fourier basis relative to the Laplacian operator of the graph formed by two 4-nodes-rings linked by an edge.

1.1.3 Special case : The ring graph and the classical DFT

Before carrying on with the extension of Graph Signal Processing to time signals, it is interesting to look at a special example : the ring graph of N nodes $\mathcal{G}_N^c$. Indeed, any discrete time periodic signal can be considered as such a graph (Figure 1.2). The adjacency matrix and its spectral decomposition is given by equation 1.1 with $\omega_N = \frac{1}{\sqrt{N}} e^{j\frac{2\pi}{N}}$.

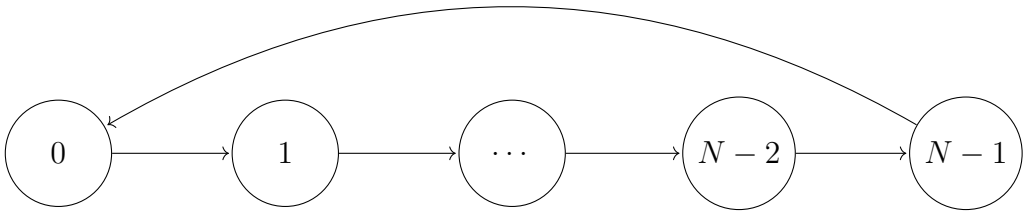


Figure 1.2: Any discrete periodic temporal signal can be reperesented as a ring graph.

$$\begin{aligned}\mathbf{A}_N^c &= \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \\ 1 & 0 & 0 & \dots & 0 \end{bmatrix} \\ &= \left[\omega_N^{nk}\right]^{-1} \begin{bmatrix} e^{-j\frac{2\pi 0}{N}} & & \\ & \ddots & \\ & & e^{-j\frac{2\pi(N-1)}{N}} \end{bmatrix} \left[\omega_N^{nk}\right] \end{aligned} \tag{1.1}$$

Thus, the Graph Fourier Transform of a time signal s, using the adjacency matrix as graph shift, gives exactly the classic Discrete Fourier Transform (Equation 1.2) and is totally coherent, as well as the other concepts we defined.

$$\hat{s} = \mathbf{V}s \text{ and in particular, } \hat{s}_k = \sum_{k=0}^{N-1} s_n e^{j\frac{2\pi kn}{N}} \tag{1.2}$$

1.2 Exact Graph Sampling

In this section, we are interested in recovering exactly a graph signal x given only partial measures of the signal at some given nodes, as presented in [Che+15].

Let suppose we want to sample M coefficients of the graph signal x given by $\mathcal{M}$, the ordered set of indices of the sampled nodes. We call $x_{\mathcal{M}}$ the values of the signal at those node. Then, we want to interpolate from these measures and build a signal x'. Ideally, we are interested in recovering exactly the original graph signal, ie $x = x'$.

Formally, we define the sampling operator $\Psi : \mathbb{C}^N \to \mathbb{C}^M$ as :

$$\Psi_{i,j} = \begin{cases} 1, & \text{if } j = \mathcal{M}_i \\ 0, & \text{otherwise} \end{cases}$$

and the interpolation operator $\Phi : \mathbb{C}^M \to \mathbb{C}^N$.

All together, we obtain :

$$\begin{cases} x_M & = \Psi x \\ x' & = \Phi x_M \end{cases}$$

if $M < N$, this task is undetermined; for a given set of value for some nodes, we can not determine for sure the original signal, there exist an infinite number of graph signal taking these values at those nodes. Thus, perfect recovery is only possible when $N = M$ and $\Phi\Psi$ is the identity matrix. To make the problem more interesting, we will suppose in the following that the signal has some structure we can exploit to recover it perfectly using only a few samples, an idea at the heart of compressed sensing [Don+06] [CRT06].

A very common prior on the structure of the graph is spatial smoothness, or in other words, we assume that the signal is bandlimited. [Che+15] A graph signal x is said K-bandlimited if for $K \in \{1, \ldots, N-1\}$, its graph Fourier transform $\hat{x}$ satisfies :

$$\hat{x}_i = 0 \text{ for } i = K+1, \ldots, N$$

If we reorder the eigenvalues and the corresponding eigenvectors in increasing value, K-bandlimited signals are obtained by a linear combination of the K least oscillating Fourier basis elements and represent thus smooth and noiseless signals. In the following, we denote as $V_{(K)}$ the truncated matrix of the first K columns of V, whose image spans the space of K-bandlimited signals. Using this assumption, the authors propose a theorem to recover a K-bandlimited graph signal using only K samples :

Theorem 1 (Chen, Varma, Kovacevic)**.** *Let the sampling operator* Ψ *satisfy*

$$rank(\Psi V_{(K)} = K)$$

Then for all signal x *with bandwith* K*, perfect recovery* $x = \Phi\Psi x$ *is obtained by choosing* $\Phi = V_{(K)}U$ *with* $U\Psi V_{(K)}$ *the* $K \times K$ *identity matrix.*

If $m = K$*, then* U *is the inverse of* $\Psi V_{(K)}$*.*

Intuitively, to recover the signal, you have to take K nodes such that $\Psi V_{(K)}$ if full rank. If the sampling matrix Ψ verifies this condition, it is called a *qualified sampling operator*. It is always possible to select such nodes respecting the condition: since the columns of V constitute an orthogonal basis, $V_{(K)}$ has rank K and there exists K rows linearly independent.

Of course, we can assume that the signal has a sparse representation in other bases than the Fourier Space. For example, another interesting choice is the graph wavelet transform, also defined using the spectrum of the Laplacian. [HVG11]

1.3 Temporal Graph Signal Processing

In the previous section, we have introduced the node sampling problem through the emergent field of Graph Signal Processing. While this allows to study signals lying on a discrete spatial domain (ie, a graph), it is interesting to try and extend this approach to spatiotemporal data.

Our idea to bring a temporal dimension to Graph Signal Processing is quite natural. Similarly to the previous chapter, we define an undirected network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of N indexed nodes $\mathcal{V}$ and M edges $\mathcal{E}$. For each node v, we are interested in a periodic temporal signal $x_{v,:}$ of period $T \in \mathbb{N}$, ie $x_{v,:} \in \mathbb{C}^T$.

We can represent this spatiotemporal signal as a matrix $\mathbf{X} \in \mathbb{C}^{N \times T}$, where the i^{th} row is given by temporal the signal of the i^{th} node $x_{v^i,:}$ or a vector $x = vec(\mathbf{X})$ built as the concatenation of the columns of $\mathbf{X}$.

We propose to bring the temporal component to the graph framework by building a larger network composed of the graph stacked several time, once for each time frame. For each node, we draw edges between the stacked copies that are consecutive in time, as represented in figure 1.3.

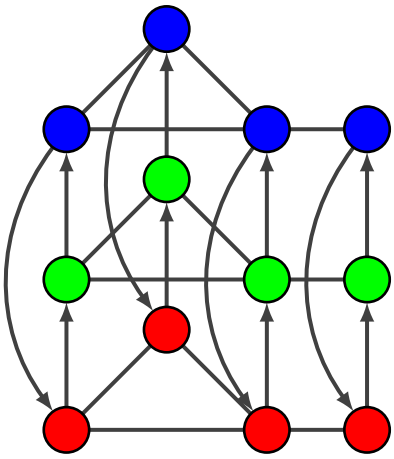


Figure 1.3: Illustration of the extension of GSP to spatiotemporal domains.

Formally, we build a graph $\mathcal{G}^T$ from $\mathcal{G}$ as the *Kronecker sum* of the adjacency matrix of $\mathcal{G}$ with the ring graph R_T of period T:

$$A_{\mathcal{G}^T} = A_{R_T} \oplus A_{\mathcal{G}} = A_{R_T} \otimes I_N + I_T \otimes A_{\mathcal{G}}$$

where I_n is the $n \times n$ identity matrix and $A \otimes B$ denotes the Kronecker product between matrices A and B.

While the numbers of nodes in this new graph grows quickly as N and T increases, the Kronecker sum structure of $\mathcal{G}^T$ makes it easy to compute the eigenvectors of its Laplacian $\mathcal{L}_{\mathcal{G}^T}$ from the eigenvector of the Laplacian $\mathcal{L}_{\mathcal{G}}$ of $\mathcal{G}$.

First, we can express $\mathcal{L}_{\mathcal{G}}$ as the Kronecker sum of the Laplacian of the ring graph and $\mathcal{G}$:

$$\begin{aligned}
\mathcal{L}_{\mathcal{G}^T} &= D_{\mathcal{G}^T} - A_{\mathcal{G}^T} \\
&= D_{R_T} \oplus D_{\mathcal{G}} - A_{R_T} \oplus A_{\mathcal{G}} \\
&= (D_{R_T} \otimes I_N + I_T \otimes D_{\mathcal{G}}) - (A_{R_T} \otimes I_N + I_T \otimes A_{\mathcal{G}}) \\
&= (D_{R_T} - A_{R_T}) \otimes I_N + I_T \otimes (D_{\mathcal{G}} - A_{\mathcal{G}}) \\
&= L_{R_T} \oplus L_{\mathcal{G}}
\end{aligned}$$

But also, the eigenvectors of $A \oplus B$ are given as the Kronecker product of the eigenvectors of B with the eigenvectors of A. [Lau05]

Lemma 1. Let $A \in \mathbb{R}^{n\times n}$ have independent eigenvectors $u_1, \dots, u_n$ and $B \in \mathbb{R}^{m\times m}$ have with independent eigenvectors $v_1, \dots, v_n$ such that u_i (resp. v_j) corresponds to eigenvalue λ_i (resp. μ_j) for $i = 1, \dots, n$ and $j = 1, \dots, m$.

Then the set of Kronecker products $\{v_j \otimes u_i\}_{\substack{i=1,\dots,n \\ j=1,\dots,m}}$ are independent eigenvectors of $A \oplus B$ and correspond to eigenvalues $\lambda_i + \mu_j$.

Proof.

$$\begin{aligned}
[A \otimes B](v_j \otimes u_i) &= [B \otimes I_n + I_m \otimes A](v_j \otimes u_i) \\
&= [B \otimes I_n](v_j \otimes u_i) + [I_m \otimes A](v_j \otimes u_i) \\
&= [Bv_j \otimes u_i] + [v_j \otimes Au_i] \\
&= [\mu_j v_j \otimes u_i] + [v_j \otimes \lambda_i u_i] \\
&= (\mu_j + \lambda_i)[v_j \otimes u_i]
\end{aligned}$$

□

So, we can compute the eigenvectors of $\mathcal{G}_T$ as the Kronecker product of the eigenvectors of the ring graph and $\mathcal{G}$. Since the eigenvectors of the ring graph are the discrete harmonics, we only have to compute the eigenvectors of $\mathcal{L}_G$ instead of computing the massive spectrum of $\mathcal{G}_T$ directly.

Also, if the spatial signal on $\mathcal{G}$ is K-bandlimited at any moment and each temporal signal on the nodes is L-bandlimited, we can recover the signal using only KL samples. If $\{u_1, \dots, u_K\}$ is a valid set of sampling nodes at any time and $\{t_1, \dots, t_L\}$ is a valid set of sampling time for any node, then the cartesian product $\{u_1, \dots, u_K\} \times \{t_1, \dots, t_L\}$ is a valid set of spatiotemporal sampling for $\mathcal{G}_T$.

Indeed, we have L valid time samples for each selected node u_i and we can recover perfectly their temporal signal. Then, for each time step, we have access to K observations (either samples, either interpolated) from a valid sampling set of nodes and can thus recover the whole signal.

Part II

Edge sampling

Chapter 2

Exact Edge sampling of conservative flows

While we were focusing so far on signals defined over nodes of a graph, we will tackle in the following chapters the natural complementary problem of sampling data lying on the edges of a graph.

One first obvious way would be to compute the *line graph* $\mathcal{G}_{LG}$ of the network and treat the edge signal as a node. Using this representation, we could thus easily generalize the existing methods of the node sampling problem to edge sampling. We define the line graph $\mathcal{G}_{LG}$ of a graph $\mathcal{G}$ as another graph whose nodes represent edges of $\mathcal{G}$ and for which two nodes are linked by an edge if their corresponding edges in $\mathcal{G}$ share a common endpoint. An example of line graph is given in Figure 2.1.

In chapter 1, we solved the problem of recovering a graph signal on the nodes from a few measurements by making the hypothesis that the signal is spatially smooth, *ie* neighboring nodes have similar values. Applying this technique to the line graph will thus be effective as soon as incident edges have a similar signal. However, edge data are often modeled as flows and this *spatially-close smoothness* prior is not well suited anymore for such signals. In particular, the line graph representation fails to preserve the structure of conservative flows. In this chapter, we will thus explore other ways to sample and interpolate edge signals.

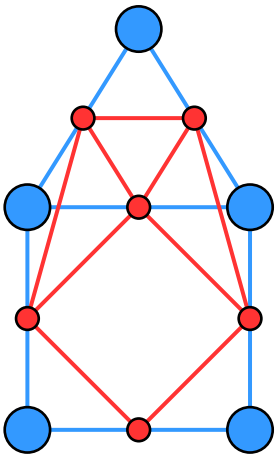


Figure 2.1: Line graph (in red) of the *house graph* (in blue).

2.1 Conservative flows

In many problems of network science, the data associated to edges represent a flow. For example, we can think of the traffic in a city, the distribution of energy in a power grid or the water flow in a network of pipelines, which all denotes the displacement of some resource from one point to another. In those settings, the orientation of the data is obviously crucial: a traffic flow from the city center to the suburbs is different from the other way around and has an opposite meaning. Furthermore, such flows are often *conservative*, which means that at each node, the incoming flow is equal to the outgoing flow.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected connected graph of N nodes and M edges that we index arbitrarily from 1 to M, we write the i^{th} edge of the graph as e^i. We define $f \in \mathbb{C}^M$ a complex signal over the edges, whose i^{th} component is the value of the signal for the i^{th} edge, according to the previously defined indexing. Since the orientation of the data is important in this framework, we arbitrarily assign a direction to every edge $e \in \mathcal{E}$ by characterizing the endpoints of the edge as either a head $h(e)$ or a tail $t(e)$. In the following applications, we will use the convention that f_i is positive if the flow goes from $h(e^i)$ to $t(e^i)$. Finally, we define the incidence matrix of the graph $\mathbf{B} \in \mathbb{R}^{N \times M}$ as $\mathbf{B}_{t(e^i),i} = -1$, $\mathbf{B}_{h(e^i),i} = 1$ or $\mathbf{B}_{k,i} = 0$ otherwise.

We can now formalize the concept of *conservative signal*. An edge signal f is said *conservative* if $\mathbf{B}f = 0$, ie $f \in ker(B)$. As each row of $\mathbf{B}$ represents

a node and its i^{th} entry is $1/-1$ if the i^{th} edge enters/leaves that node, $\mathbf{B}f$ computes the net flow at each node. Thus, the signal is conservative if at each node, all flow that enters the nodes immediately leaves it. Consequently, as soon as we know all but one edge data adjacent to a particular node, we can recover the last one such that it cancels out with the net sum of the other flows. But how much does this hypothesis help ? For a given network, a first question would be to determine the minimum number of edge samples to recover the signal perfectly. Before answering this question, we will introduce an important paradigm that will be useful for the rest of the chapter, the *Cycle Space* of the graph.

2.2 Cycle Spaces

A path P is a sequence of nodes and edges $\{v_0, e_0, v_1, e_1, \ldots, e_n, v_{n+1}\}$ such that consecutive links e_i and e_{i+1} share a common endpoint v_i. If $v_0 = v_{n+1}$, then we call the path a circuit and if every other nodes are different, then $\mathcal{P}$ is called a cycle. Furthermore, we say that if node v_i comes just before v_j in the sequence (ie $i = j - 1$), then v_i precedes v_j. We associate to each edge some characteristic function $\mathcal{X} : \mathcal{E} \to \{-1, 0, 1\}^M$ as follow :

$$\mathcal{X}(e^j)_i = \begin{cases} -1 & \text{if } i = j \text{ and } e^j = (h(e^j), t(e^j)) \\ 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \text{ and } e^j = (t(e^j), h(e^j)) \end{cases} \tag{2.1}$$

Finally, for any path P, we define an incidence vector $c(P)$ as the sum of the indicator vectors of the edges in the path :

$$c(\mathcal{P}) = \sum_{e \in P} \mathcal{X}(e) \tag{2.2}$$

The *Cycle Space* $CS(\mathcal{G})$ is the vector space generated by the incidence vectors of all cycles of $\mathcal{G}$. These incidence vectors are called *cycle vectors* and a basis of $CS(\mathcal{G})$ that only contains cycle vectors is called a *TUM cycle basis*, or *totally-unimodular cycle basis* .

Lemma 2. The dimension of $CS(\mathcal{G})$ is $\mu(\mathcal{G}) = M - N + 1$.

Proof. We prove this result by induction.

1. Base case : Let $\mathcal{G}_{ST} = (\mathcal{V}, \mathcal{E}_{ST})$ a spanning tree of $\mathcal{G}$. By definition, a tree has no cycle and the dimension of $CS(\mathcal{G}_{ST})$ is 0.

2. For a subgraph $\mathcal{G}_i = (\mathcal{V}, \mathcal{E}_i)$ of $\mathcal{G}$ with i edges ($N - 1 \leq i < M$), let assume that $dim\ CS(\mathcal{G}_i) = i - N + 1$. For any subgraph $\mathcal{G}_{i+1} = (\mathcal{V}, \mathcal{E}_{i+1})$ of $i + 1$ edges such that $\mathcal{E}_i \subset \mathcal{E}_{i+1}$, we are proving that $dim\ CS(\mathcal{G}_{i+1}) = (i + 1) - N + 1$.

   Let $e = (u, v)$ be the edge in $\mathcal{E}_{i+1}$ but not in $\mathcal{E}_i$. Since there exists a path from u to v in $\mathcal{G}_i$, the concatenation of this path with e is a cycle whose incidence vector is not in $CS(\mathcal{G}_i)$. Thus, $dim\ CS(\mathcal{G}_{i+1}) \geq (i+1) - N + 1$.

   On the other hand, if $dim\ CS(\mathcal{G}_{i+1}) > (i + 1) - N + 1$, there exists at least two linearly independent incidence vectors in $CS(\mathcal{G}_{i+1})$ but not in $CS(\mathcal{G}_i)$. Thus, we can find a linear combination of these two vectors such that the entry relative to the edge e is null. Since this new indicator vector uses only the edges of $\mathcal{E}_i$, it should belong to $CS(\mathcal{G}_i)$. Hence, a contradiction and $dim\ CS(\mathcal{G}_{i+1}) = (i + 1) - N + 1$.

We conclude by induction that $dim\ CS(\mathcal{G}) = M - N + 1$. □

$\mu(\mathcal{G})$ is called the *circuit rank* or the *cyclomatic number* of $\mathcal{G}$. Usually, it is defined as $|\mathcal{E}| - |\mathcal{V}| + c(\mathcal{G})$ where $c(\mathcal{G})$ is the number of connected components of $\mathcal{G}$ and it has several applications in algebraic graph theory. Intuitively, it counts the maximum number of independent cycles in $\mathcal{G}$, where a set of cycles are called independent if none of them can be obtained as the symmetric difference of the others. [Ber01]

Thanks to the proof, we have a constructive algorithm to easily find a basis of the cycle space. Firstly, one can find a spanning tree of the graph, using for example a graph traversal algorithm such as *Depth First Search*. Then, for each edge not in the tree, we find a path in the tree linking both endpoints and build a cycle by concatenating the edge to this path. To avoid any ambiguity about the orientation of the cycle, we impose that the tail of the edge not in the tree precedes its head in the cycle.

Bases built that way, using a spanning tree, are called *Fundamental bases*. Equivalently, those bases can be defined as the one having the following property : each cycle in the basis contains an edge not contained in any other cycle of the basis. [BGV09] We call these edges the *cycle edges* relative to the

fundamental base. Later in this chapter, we will see how to use such bases to perfectly recover a conservative edge signal.

The following theorem makes the link between conservative flows and the cycle space of a graph.

Theorem 2. $ker(B) = CS(\mathcal{G})$

Proof.

1. $CS(\mathcal{G}) \subset ker(B)$

   Let $x \in CS(\mathcal{G})$. Then, $x = \sum_{c_i \in \mathcal{B}} \alpha_i c_i$ for some TUM cycle basis $\mathcal{B}$. Since each cycle vector c_i represents a unitary conservative flow along a cycle, $Bc_i = 0$ and $x \in ker(B)$.

2. $ker(B) \subset CS(\mathcal{G})$

   Let $f \in ker(B)$ and let apply the following iterative procedure :

   (a) Select a node u_0 with a non-zero outgoing remaining flow for some edge $e_0 = (u_0, u_1)$. If none of them exists, terminate.

   (b) For $i = 1, \ldots, k$, select an edge with non-zero remaining flow $e_i = (u_i, u_{i+1})$ such that $u_{i+1} \neq u_{i-1}$. If $u_{i+1} \in \{u_0, \ldots, u_{i-2}\}$, go to step (c).

   (c) Update f by removing to each selected edge $\{e_0, \ldots, e_k\}$ the flow of e_0. Go back to step (a).

   We build a sequence of flow $\{f = f_0, f_1, \ldots, f_j = 0\}$ by subtracting at each step a cycle vector. Each step terminates because we either visit a new node in the cycle or finish the step and since there are a finite number of nodes, it has to stop. The procedure terminates since at each iteration, only non-zero flows are updated and at least one edge has its value brougth to non-zero. Finally, since the remaining flow is conservative, a node cannot have only one incoming/outgoing non-zero flow. Thus, we can find at each step a non-zero outgoing flow for u_i. We conclude that f can be written as a linear combination of cycle vectors. □

Therefore, the space of conservative flows is exactly the cycle space of the graph and we can study the edge recovery problem through the well-established algorithms and theorems of this domain.

For reference, a review of cycle bases is given in [BGV09].

2.3 Exact edge sampling

Let formalize the edge sampling problem. We are given a conservative flow signal $f \in \mathbb{C}^M$ on the edges but have only access to the values of K edges we can choose. We denote by $\mathcal{M}$ the set of sampled edges and $f_{\mathcal{M}}$ the value of the flow at these entries. From these measures, we want to interpolate the unknown data and build a signal f' equal to f in a noiseless setting. Intuitively, the space of conservative flows has dimension $M - N + 1$ and we need to sample at least this number of edges to recover perfectly the signal. Theorem 3 states that it is always enough to pick $K = M - N + 1$ and gives which set of edges one has to sample to interpolate f.

Theorem 3. *Let $\mathcal{G}$ be a connected graph and f be a conservative flow. Then, we can recover perfectly f using only $M - N + 1$ measurements. Moreover, we can recover perfectly f with the edges $\mathcal{M}$ (1) if and only if the subgraph $(\mathcal{V}, \mathcal{E} \setminus \mathcal{M})$ is acyclic (2).*

Proof.

- (1) $\Rightarrow$ (2)

  We consider that $(\mathcal{V}, \mathcal{E} \setminus \mathcal{M})$ is not acyclic and call $\mathcal{C}$ one cycle of the subgraph. For a recovery attempt f' consistent with the measurements, $f'' = f' + \lambda c(\mathcal{C})$ is also a consistent conservative flow for $\lambda \in \mathbb{C}$. Thus, we cannot recover perfectly f.

- (2) $\Rightarrow$ (1)

  We start by building a spanning tree $\mathcal{T}$ of the graph such that $\mathcal{E} \setminus \mathcal{M} \subset \mathcal{T} \subset \mathcal{E}$ and we denote by $\mathcal{B} = \{c_1, \dots, c_K\}$ its induced fundamental basis. As $f \in CS(\mathcal{G})$, we can write it as $f = \sum_{i=1}^{K} c_i \alpha_i$ with $\alpha_i \in \mathbb{C}$. Since $\mathcal{B}$ is fundamental, each cycle c_i has at least one edge e^j that does not belong to any other cycle of the basis, and we find the coefficient $\alpha_i = f_j$ from the measurement of this edge. Thus, we can recover f from $\mathcal{M}$.

It follows from (2) $\Rightarrow$ (1) that we can recover perfectly f using $M - N + 1$ measurements. □

2.3.1 Naive algorithm

The proof also gives a first intuitive algorithm to build the flow of all edges from the measurements. A pseudocode is given in algorithm 1. As all other algorithms presented in this thesis, a python implementation is provided in the appendix.

The time complexity of the sampling procedure is $\mathcal{O}(N+M)$ since we can use a Depth First Search to build the spanning tree in linear time. Regarding the interpolation algorithm, the outer loop iterates $M-N+1$ times and the inner one at most N times. Moreover, we can find the path between two nodes on a tree in $\mathcal{O}(N)$. The total worst case time complexity is thus $\mathcal{O}(NM)$.

2.3.2 Leaves cutting algorithm

While it is practically possible to use the naive algorithm for moderately large graph, it soon becomes inappropriate for real life network with millions of nodes. Fortunately, it is possible to interpolate the data flow from the measures more efficiently in linear time, as I propose in algorithm 2.

The main idea behind the algorithm is to use iteratively the conservative equation for each node. After the sampling procedure, the set of edges for which we do not know the value of the signal is a spanning tree. Since each leaf of the spanning tree has only one unmeasured adjacent edge, we can infer the value of its flow using the conservative equation ; the net flow on each node has to be zero. Then, we remove from the tree the inferred edges and as cutting a leaf from a tree results in a smaller tree, we can repeat the algorithm until no edges remains. I call this procedure the *Leaves cutting algorithm*.

There remains some technical details to ensure a linear time complexity which makes the implementation quite tricky. Finding a leaf by looping through all nodes and checking if the degree is one would be inefficient as it would take $\mathcal{O}(N)$ steps for each subtree. Instead, I propose to find all the leaves of the initial tree once at the start of the algorithm and keep them on a list. Then, as we cut a leaf or a new one appears, we update the list to be sure it is consistent with the subtree at each time step. To detect emerging leaves, we can keep the degree of each node on an array and decrement it by one when we remove an adjacent edge. When the degree gets to one, we can

Algorithm 1 Edge sampling and interpolation algorithm

Require:
$\mathcal{G}$, an undirected connected graph

function SAMPLE($\mathcal{G}$)
  $\mathcal{T} \leftarrow$ SPANNINGTREE($\mathcal{G}$)
  $\mathcal{M} \leftarrow \{\}$
  for $e \notin \mathcal{T}$ **do**
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{e\}$
  end for
  return $\mathcal{M}$
end function

Require:
$\mathcal{G}$, an undirected connected graph
$\mathcal{M}$, the complementary set of a spanning tree of $\mathcal{G}$
f, an array of M elements.
$f[i]$ equals the measure of edge i if $i \in \mathcal{M}$, or 0 otherwise.

function INTERPOLATE($\mathcal{G}$, $\mathcal{M}$, f)
  $\mathcal{T} \leftarrow \{\}$
  for $e \notin \mathcal{M}$ **do**
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{e\}$
  end for
  for $e^i \in \mathcal{M}$ **do**
    $p \leftarrow$ GETTREEPATH($\mathcal{T}$, $t(e^i)$, $h(e^i)$)
    for $e^j \in p$ **do**
      $f[j] \leftarrow f[j] + f[i]$
    end for
  end for
  return f
end function

add the node to the list of leaves. Thanks to these considerations, we can find a leaf of the subtree in $\mathcal{O}(1)$ at each step.

Theorem 4. *The worst case time complexity of the leaves cutting algorithm is $\mathcal{O}(M)$.*

Proof. Let $T(N)$ be the time needed to interpolate the edge data on a tree of N nodes and u_i the leaf cut at step i. It takes $O(deg(u_i))$ steps to compute the net flow of node u_i and find the edge with the missing data.

$$\begin{aligned} T(N) = &\text{ time to find a leaf + time to compute the net flow } + T(N-1) \\ &= \mathcal{O}(1) + \mathcal{O}(deg(u_0)) + T(N-1) \\ &= \sum_{u \in \mathcal{V}} \mathcal{O}(deg(u)) \\ &= \mathcal{O}(2M) \text{ from the handshake lemma} \end{aligned}$$

□

2.4 The Edge-Laplacian

The main focus of Graph Signal Processing is the study of node data, and it is only recently that some research has been considering edge data having a flow structure. Two independent works proposed methods to smooth and denoise flow data [SS18] [BSC18] and have been the starting point of my extension to the sampling and interpolation problem using the cycle space theory.

Both approaches use the concept of *Edge-Laplacian*, defined as $\mathbf{L}_1 = \mathbf{B}^T\mathbf{B}$. As a reminder, the classical Laplacian as $\mathbf{L} = \mathbf{B}\mathbf{B}^T$. What can we say about this Edge Laplacian ?

Theorem 5. *Given a graph with Edge-Laplacian $\mathbf{L}_1 = \mathbf{B}^T\mathbf{B}$, the following properties hold :*

1. *The null space of $\mathbf{L}_1$ is equivalent to the cycle space ker($\mathbf{B}$) of $\mathcal{G}$, also known as the harmonic space*
2. *The image of $\mathbf{L}_1$ is equivalent to the gradient space im($\mathbf{B}^T$), also known as the cut space*

Algorithm 2 Leaves cutting algorithm

Require:
$\mathcal{G}$, an undirected connected graph
$\mathcal{M}$, the complementary set of a spanning tree of $\mathcal{G}$
f, an array of M elements.
$f[i]$ equals the measure of edge i if $i \in \mathcal{M}$, or 0 otherwise.

```
1: function LeavesCuttingInterpolation(G, M, f)
2:     T ← {}
3:     for e ∉ M do
4:         T ← T ∪ {e}
5:     end for
6:     leaves ← GetLeaves(T)
7:     while T is not empty do
8:         curr ← leaves.pop()
9:         net_flow ← 0
10:        for e^i adjacent to curr do
11:            if e^i ∉ T then
12:                if h(e^i) == curr then
13:                    net_flow ← net_flow +f[i]
14:                else
15:                    net_flow ← net_flow −f[i]
16:                end if
17:            else
18:                out_edge ← e^i
19:            end if
20:        end for
21:
22:        if h(out_edge) == curr then
23:            f[out_edge] ← - net_flow
24:        else
25:            f[out_edge] ←  net_flow
26:        end if
27:
28:        T ← T \ {out_edge}
29:        if degree(out_edge.other_end) == 1 then
30:            leaves.append(out_edge.other_end)
31:        end if
32:    end while
33:    return f
34: end function
```

It results from this proposition that the cycle space is the vector space spanned by the first $M - N + 1$ eigenvectors of the Edge-Laplacian, and our conservative prior implies that the signal belong to this space. As we have seen previously, our spatially-smooth prior in the node domain was very similar. Indeed, that prior implied that the signal lied in the space generated by the first eigenvectors of the Laplacian, which makes a nice comparison with this new *conservative flows* setting.

Chapter 3

Exact Cycle sampling of conservative flows

In the previous chapter, we introduced methods to solve the exact edge sampling problem. However, in real-life applications, we do not always have access directly to the edges we want to sample, because the measure could be too expansive or too complex. Instead, we will consider that we observe the net value of the flow over cycles of the graph. We denote by $\mathcal{M}_c = \{\mathcal{C}_1, \dots, \mathcal{C}_K\}$ the set of observed cycles and for each cycle $\mathcal{C}_i$, we have access to $b_i = \sum_j f_j c_j(\mathcal{C}_i)$. We call this procedure *cycle sampling*, in contrast with the *edge sampling* procedure we have established previously.

3.1 Motivation

This alternative way to observe the signal is in particular quite natural for some applications for which we have only access to a memory-limited agent able to wander in the network. Given a cycle path, the agent can explore the edges of the path and only remember the sum of their values with respect to the edges orientation. Even better, the cycle does not need to include the starting point of the agent. If there is a path p from the starting point to a node of the cycle, the agent would sum up irrelevant signal values alongside this path. But as it comes back from the cycle to the starting point, the agent crosses the edges of the path in the opposite directions and their signal values is subtracted from the total. We finally get the net flow of the distant cycle.

This setting is particularly handful in closed networks with an unique access point. For example, we can think about measuring the water flows of an underwater cave using a limited submarine (figure 3.1). This could also maybe be applied to estimate the load of a computer network for which we do not have access to intermediate devices, as the various routers. Indeed, a common simple model suggests that the local delay of a packet is proportional to the load of the link when the network is congested. The total delay a packet takes to go from one device to another is can thus be seen as a linear combination of the load of the network links, which is our observation. This is already used with the software *traceroute* or in the context of network tomography with directed graphs.

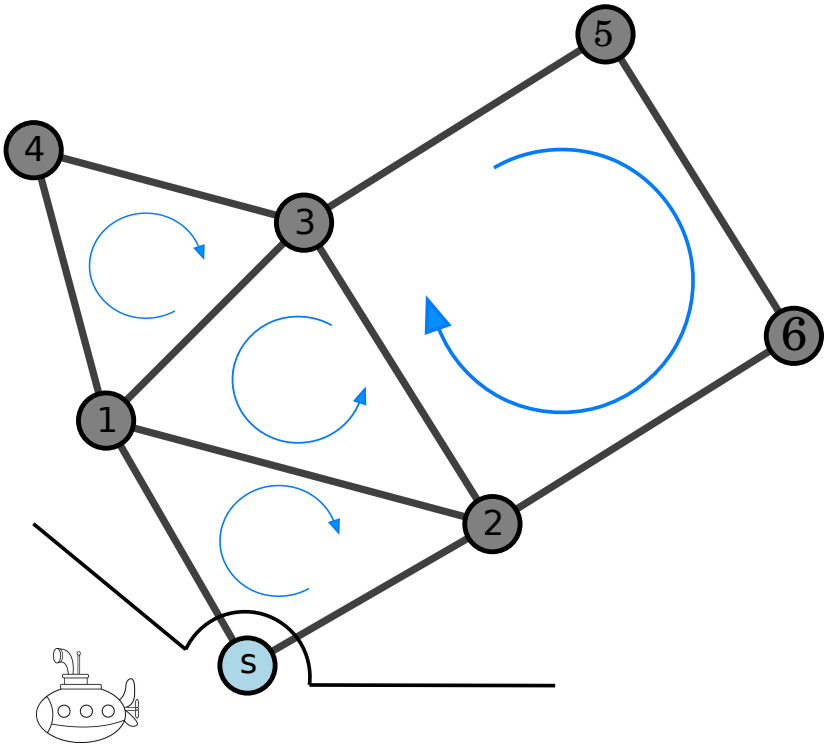


Figure 3.1: Illustration of the submarine motivation. Gray nodes and edges are not directly accessible and one wants to use a submarine to sample the flows on the underwater cave.

3.2 Signal recovery

As we did in the previous edge sampling problem, we have to ask ourselves which set of cycles enable us to recover the original signal perfectly.

Theorem 6. *Let $\mathcal{G}$ be a connected graph with N nodes and M edges. We consider an unknown conservative flow f that we want to sample. We can perfectly recover f from a cycle sampling procedure using only $M - N + 1$ measurements. Moreover, we can recover perfectly f with the set of cycle $\mathcal{M}_c$ (1) if and only if $\mathcal{M}_c$ is a generating sequence of the space $CS(\mathcal{G})$ (2).*

Proof. Let $\mathbf{C} \in \mathbb{N}^{N\times(M-N+1)}$ be a matrix whose columns represents a cycle basis of $CS(\mathcal{G})$ and b the vector of observations. As f is a conservative and belongs to $CS(\mathcal{G})$, we can write it as $f = \mathbf{C}\alpha$ for some coefficients $\alpha \in \mathbb{C}^K$. Then, we note $\tilde{\mathbf{C}} \in \mathbb{N}^{N\times K}$ the matrix whose columns are the cycle vectors of the measured cycles $\mathcal{M}_c$.

Since the measure is the sum of the signal over the edges of each cycle, we have that $b = \tilde{\mathbf{C}}^T f = \tilde{\mathbf{C}}^T\mathbf{C}\alpha$.

- (1) $\Rightarrow$ (2)

  Let consider that $\mathcal{M}_c$ is not a generating sequence of $CS(\mathcal{G})$. We find the rank of the matrix associated to the system $rank(\tilde{\mathbf{C}}^T\mathbf{C}) \leq \min(rank(\tilde{\mathbf{C}}), rank(\mathbf{C})) \leq rank(\tilde{\mathbf{C}}) < N - M + 1$ since the column of $\tilde{\mathbf{C}}$ does not generate $CS(\mathcal{G})$. Thus, the system is undetermined and we cannot recover f, which prove the result by contraposition.

- (2) $\Rightarrow$ (1)

  First, let restrict $\mathcal{M}_c$ to a basis of $N - M + 1$ elements, and without loss of generality, use this basis to define $\mathbf{C}$. Since $\tilde{\mathbf{C}}$ is equal to $\mathbf{C}$ up to a permutation of the columns $\mathbf{\Pi}$. The system $\mathbf{\Pi}^T\mathbf{C}^T\mathbf{C}\alpha = b$ is invertible and we can thus uniquely recover α and the signal f. This also proof that we only need $M - N + 1$ measurements.

□

Recovering a conservative signal using cycle sampling can thus be done in two steps :

1. Solve the system $\mathbf{C}^T\mathbf{C}\alpha = b$
2. Compute $f = \mathbf{C}\alpha$

Where the columns of $\mathbf{C}$ are the independent cycle vectors used in the sampling process.

3.3 Optimal choice of basis

In general, there are a lot of ways to build a cycle basis, since the number of simple cycles can be exponential in the number of nodes. In the complete graph K_N, there are even $(N-1)!$ cycles of length N, and we do not even count smaller cycles. Thus, we are tempted to define some criteria to help us choose a suitable basis from all these cycles.

We will focus on this section on the *restricted access* network I have already briefly introduced. We would like to cycle-sample a network $\mathcal{G}$ using a memory-limited agent that sums up the flow of the edges it goes through. However, the network is not easily accessible and we can start the agent at only one node s. Ideally, we would like our agent to travel as few edges as possible. Thus, our task is to find the cycle basis $\{\mathcal{C}_1, \ldots, \mathcal{C}_K\}$ that minimizes:

$$\sum_{i=1}^{K} l(\mathcal{C}_i) + 2 \min_{u \in \mathcal{C}_i} l(SP(s,u)) \tag{3.1}$$

where $l(\cdot)$ gives the length of a cycle or a path and $SP(s,u)$ is an arbitrary shortest path between s and u.

Minimizing only the total length of the cycles of the basis is a well-known task in cycle-space theory, called the *Minimum Weight Cycle Basis* problem (MWCB). We know polynomial algorithms to solve it, Horton's algorithm for example [Hor87], which has greatly inspired the algorithm proposed in the next sections of this chapter. As a reference to this well-studied problem, we name our task the *Fixed Source Minimum Weight Cycle Basis* problem (FSMWCB).

As we have seen in the previous chapter, it is easy to build a cycle basis in linear time, using for example a Depth First Search algorithm. However, by using this methodology, we restrict ourselves to fundamental cycle basis, which is a much smaller subset of cycles bases. In fact, it is very difficult to

work in the space generated by fundamental cycles, as we will see in the next chapter; the minimum weight cycle basis problem is even *NP-HARD* when we restrict ourselves to such cycles.

Instead, we will take advantage of one of the most useful property of cycles spaces : $CS(\mathcal{G})$ can be seen as a *matroid*. Especially, it will allow us to work with optimal greedy algorithms.

Remark
Since direction of the edge flows are not relevant when we are looking to minimize the length of some cycles and some paths, we will work on $GF(2)$, the Galois Field with 2 elements. In this space, the indicator function of paths does not take into account the in which direction each edge is taken, as -1 becomes 1. We will use an ordered tuple of edges $((v_0, v_1), \ldots, (v_n, v_{n+1}))$ to represent paths and we define the addition of two paths $P_1 + P_2$ as the their symmetric difference.

3.3.1 Matroids

First, lets review the concept matroid. [DH+]

A matroid is a pair $(E, \mathcal{F})$ where E is a finite set and $\mathcal{F}$ is a family of subsets of E satisfying the following properties :

1. $\emptyset \in \mathcal{F}$
2. If $B \in \mathcal{F}$ and $A \subset B$, then $A \in \mathcal{F}$
3. If $A, B \in \mathcal{F}$ and $|A| < |B|$, then there exists $x \in B \setminus A$ such that $A \cup \{e\} \in \mathcal{F}$

The elements of $\mathcal{F}$ are called the *independent sets* of E and a maximal independent set is called a basis of the matroid.

Matroids can be seen as a generalization of the concept of linear independence in vector space to other spaces, for example cycle spaces. Indeed, we can define E as the set of cycles of $\mathcal{G}$ and $\mathcal{F}$ as containing sets of cycles such that their cycle vectors are independent. It naturally corresponds to the definition.

Furthermore, we can assign a weight $w(x)$ to each element x of E and define the weight of a basis $\mathcal{B} = \{x_1, \ldots, x_k\}$ as the sum of the weight of its elements. We call such a structure a *weighted matroid*. A great property about weighted matroids is that the polynomial-time greedy algorithm (algorithm 3) returns a basis with minimum weight, which often makes it easy to work with such structures. Usually, the trickiest part is to provide an efficient procedure that checks if a set is independent (line 5). However, when the matroid can be represented as a vector space, as $CS(\mathcal{G})$ through cycle vectors, we can simply use Gauss elimination to check independence.

Algorithm 3 Greedy matroid algorithm

Require:
E a set of elements
w a weight function $E \to \mathbb{R}$

```
1: function MINIMUM_WEIGHT_BASIS(E, w)
2:     L ← a sorted list of the elements of E
3:     B ← {}
4:     for x ∈ L do
5:         if B ∪ x is an independent set then
6:             B ← B ∪ x
7:         end if
8:     end for
9:     return B
10: end function
```

Example : Kruskal's algorithm

One of the most famous algorithm in graph theory is *Kruskal's algorithm*. Given a weighted graph, Kruskal's algorithm returns the minimum spanning tree. In fact, Kruskal's algorithms is nothing more than the greedy matroid algorithm applied to the matroid $(E, \mathcal{F})$ where E is the set of edges of the graph and $\mathcal{F}$ is the family of acyclic subgraphs. The independence is verified thanks to a disjoint sets structure, as *Union-Find* for example, but it can also be done less efficiently using the incidence vector representation of the edges over GF-(2) through Gaussian elimination. Such matroids representing acyclic graphs are called *graphic matroids*.

3.3.2 Horton's algorithm

To solve the $MWCB$ problems, Horton proposed in 1987 to use the matroid greedy algorithm with the weight of each cycle set as its length. Although the algorithm is correct, we face a big issue : the worst case time complexity is polynomial in the number of different cycles in the graph, which can be exponential in the number of nodes or even worse for some classes of graphs. To resolve this issue, Horton ingenuously showed that only a small fractions of these cycles are relevant in the construction of a minimum weight basis and gave a way to generates those suitable cycles.

Lemma 3. Let $\mathcal{B}$ a cycle basis of $CS(\mathcal{G})$ and $\mathcal{C} \in \mathcal{B}$. If $\mathcal{C} = \mathcal{C}_1 + \mathcal{C}_2$ for two cycles $\mathcal{C}_1$ and $\mathcal{C}_2$, then $\mathcal{B} \setminus \{\mathcal{C}\} \cup \{C_i\}$ is a cycle basis for $i = 1$ or 2

Proof. See [Hor87]. □

Theorem 7. *Let $\mathcal{C}$ be a cycle in a minimum cycle basis of $\mathcal{G}$ and u an arbitrary node of $\mathcal{C}$. Then there is an edge $(v, w) \in \mathcal{C}$ such that $\mathcal{C}$ consists of a shortest $u - v$ path, a shortest $u - w$ path and the edge (v, w).*

Proof. Let's consider that it is not the case.

We divide the proof in three parts.

- First, let $a \neq u$ be a node of $\mathcal{C}$ and $b \neq u$ one of its neighbour. We are going to prove that for any shortest paths $SP_1(a, u)$ and $SP_2(u, a)$ between a and u, $\tilde{\mathcal{C}} = SP_1(a, u) + SP_2(u, a)$ is generated by $\mathcal{B}_{\mathcal{C}} = \mathcal{B} \setminus \{\mathcal{C}\}$. If it is not true, then we can replace $\mathcal{C}$ by $\tilde{\mathcal{C}}$ in $\mathcal{B}$ and obtain another basis. We are showing here that this basis has a smaller weight, which cannot be possible since $\mathcal{B}$ is a minimum basis.

  Let's denote by P_1 (resp. P_2) the path in $\mathcal{C}$ that goes from u to a (resp. from b to u) without passing through b (resp. a). Obviously, we have $\mathcal{C} = P_1 + (a, b) + P_2$.

  We are assuming that theorem 2 is false, hence either P_1 or P_2 is not a shortest path (or both). Since $SP_2(u, a)$ is a shortest path, it is shorter than the path $(a, b) + SP(b, u)$ for any shortest path between b and u. We conclude by exploring both cases :

  1. If $l(SP_1(a, u)) < l(P_1)$, then

$$\begin{aligned} l(SP_1(a, u)) + l(SP_2(u, a)) &\leq l(SP_1(a, u)) + l((a, b) + SP(b, u)) \\ &< l(P_1) + l(a, b) + l(P_2) = l(\mathcal{C}). \end{aligned}$$

2. If $l(SP(b,u)) < l(P_2)$, then

$$l(SP_2(u,a)) \leq l((a,b) + SP(b,u)) < l((a,b) + P_2)$$
$$\text{and } l(SP_1(a,u)) + l(SP_2(u,a)) < l(P_1) + l((a,b) + P_2) = l(\mathcal{C}).$$

$\tilde{\mathcal{C}}$ has thus a strictly smaller weight than $\mathcal{C}$, and if we could replace $\mathcal{C}$ by this element, $\mathcal{B}$ would not be a minimum basis. Thus, we conclude that $\tilde{\mathcal{C}}$ is generated by $\mathcal{B}_{\mathcal{C}}$.

- Secondly, there exists an edge $e = (v,w)$ of $\mathcal{C}$ such that $SP(w,u) + SP(u,v) + (v,w)$ is not generated by $\mathcal{B}_{\mathcal{C}}$ for any shortest paths $SP(v,u)$ and $SP(w,u)$. Let us consider that it is not true.

  Then for each edge $(v',w') \in \mathcal{C}$, there exists two shortest paths $SP(u,v')$ and $SP(u,w')$ such that $\mathcal{C}_{(v',w')} = SP(w',u) + SP(u,v') + (v',w') \in span(\mathcal{B}_{\mathcal{C}})$.

  The sum of all those sets of cycles is given by :

$$\begin{aligned}\sum_{e' \in \mathcal{C}} \mathcal{C}_{e'} =& \mathcal{C} + \sum_{(v',w') \in \mathcal{C}} SP(v',u) + SP(w',u) \\ & \mathcal{C} + \sum_{\text{node } a \text{ in } \mathcal{C}} SP_1(a,u) + SP_2(a,u)\end{aligned}$$

  where $SP_{i=1,2}(a,u)$ are the two possibly different shortest paths used in the two cycles corresponding to the two edges containing a.

  Using the previous result, $SP_1(a,u) + SP_2(a,u) \in span(\mathcal{B}_{\mathcal{C}})$ for each node a of the cycle. Since we also have that $\sum_{e' \in \mathcal{C}} \mathcal{C}_{e'} \in span(\mathcal{B}_{\mathcal{C}})$, we find that $\mathcal{C}$ is not independent from $\mathcal{B}_{\mathcal{C}}$, which is not possible because $\mathcal{B}$ is a basis.

- Finally, we know from the last part that there exists an edge (v,w) such that $\mathcal{C}$ can be replaced in $\mathcal{B}$ by $\tilde{\mathcal{C}} = SP(w,u) + SP(u,v) + (v,w)$ for two shortest paths. Let us use again our notation P_1 and P_2 for the paths in $\mathcal{C}$ that goes from u to v and w. We have that either $l(SP(u,v)) < l(P_1)$ or $l(SP(u,w)) < l(P_2)$ (or both) from our assumption that our theorem is false. Thus $\tilde{\mathcal{C}}$ has smaller weight than $\mathcal{C}$, which would make the basis $\mathcal{B} \setminus \{\mathcal{C}\} \cup \tilde{\mathcal{C}}$ better than $\mathcal{B}$. This is impossible because $\mathcal{B}$ is optimal and we conclude that the theorem is correct. □

Remark : This is not the original proof published by Horton in 1987. Indeed, he made strong assumptions in his proof such as unicity of the

shortest path for example. I thus decided to produce another proof, since it will generalize well for the FSMWCB problem.

Theorem 7 gives a very strong property of minimum cycle bases. In fact, we can found all relevant cycles that could appear in a minimum cycle basis by considering for each node u and each edge (v, w) the circuits $\mathcal{C} = SP(u, v) + (v, w) + SP(w, u)$ for all possible disjoint shortest paths.

Even more, for each tuple $(u, (v, w))$, we do not need to generate all possible shortest paths as we can choose any cycle with arbitrary shortest paths and still get a set of circuits that contains a minimum weight basis. For example, we can choose the lexicographically smallest cycle, ie the cycle that has the lexicographically smallest set of nodes. We call a set of cycles built this way an *Horton's set* and as it has at most NM elements, we can apply the greedy algorithm efficiently.

Actually, we often have less than NM elements in the Horton's set because two different pairs of node and edge can give the same lexicographically smallest cycle. To speed up computations, this suggests to find good ways to select the shortest paths that generate a smaller set of relevant cycles, for example as in [MM09].

Theorem 8. *The matroid greedy algorithm applied to the Horton set returns a minimum cycle basis.*

Proof. Let us consider a run of the greedy algorithm on the full set of cycles sorted by weight, and in case of tie, cycle with lexicographically smallest set of nodes come first.

Let $\mathcal{B}$ be the basis generated by the algorithm and $\mathcal{C}$ be any circuit selected by the algorithm not in the Horton's set. Using theorem 7, there exists an edge (v, w) for any node u such that $\mathcal{C} = SP_1 + SP_2 + (v, w)$ where SP_1 (resp. SP_2) is a shortest path between u and v (resp. u and w).

When we generated the Horton's set, we chose a circuit $\mathcal{C}^\star$ for the node u and the edge (v, w) that is lexicographically smaller than $\mathcal{C}$. We note $\mathcal{C}^\star = SP_1^\star + SP_2^\star + (v, w)$, and $SP_i \neq SP_i^\star$ for at least one i.

We define $\mathcal{C}_1 = SP_1 + SP_2^\star$, $\mathcal{C}_2 = \mathcal{C}^\star$ and $\mathcal{C}_3 = SP_2 + SP_1^\star$ where at least $\mathcal{C}_1$ or $\mathcal{C}_3$ is not empty. Each $\mathcal{C}_i$ has lower weight or is lexicographically smaller than $\mathcal{C}$, thus they all have been considered by the greedy algorithm before $\mathcal{C}$. As $\mathcal{C} = \mathcal{C}_1 + \mathcal{C}_2 + \mathcal{C}_3$, we find that at least one of them could replace $\mathcal{C}$ in $\mathcal{B}$ using lemma 3. But since such cycle would have been chosen by the greedy

algorithm because it has been considered before $\mathcal{C}$, we have a contradiction and such $\mathcal{C}$ does not exists. □

Taking inspiration from this clever trick, I will propose a similar algorithm to solve the Fixed Source Minimum Weight Cycle Basis.

3.3.3 Fixed Source Minimum Weighted Cycle Basis

As a reminder, the FSMWCB problem is to find a cycle basis that minimizes the sum over each cycle of the basis of its length plus its distance from a source s. We are going to work again with the matroid structure of $CS(\mathcal{G})$ and for each cycle $\mathcal{C}$ of the graph, we set its weight to $l(\mathcal{C}) + 2\ min_{u \in \mathcal{C}} l(SP(s,u))$.

Again, we can apply the greedy algorithms to have a minimal basis, but the number of cycles is highly prohibitive. Fortunately, we can easily generalize theorems 7 and 8 that allows to build a minimum basis while considering much fewer cycles.

Theorem 9 (**Extension of theorem 2**). *Let $\mathcal{C}$ be a cycle in a fixed source minimum cycle basis of $\mathcal{G}$ and u an arbitrary node of $\mathcal{C}$. Then there is an edge $(v,w) \in \mathcal{C}$ such that $\mathcal{C}$ consists of a shortest $w-u$ path, a shortest $u-s$ path, a shortest $s-v$ path and the edge (v,w).*

Remark : an empty shortest path is allowed if $u = s$ or $w = s$.

Proof. Let's consider that it is not the case.

We divide the proof in three parts.

- First, let $a \neq u$ be a node of $\mathcal{C}$ and $b \neq u$ one of its neighbour. We are going to prove that for any shortest paths $SP(u,a)$, $SP(a,s)$ and $SP(s,u)$, $\tilde{\mathcal{C}} = SP(u,a) + SP(a,s) + SP(s,u)$ is generated by $\mathcal{B}_{\mathcal{C}} = \mathcal{B} \setminus \{\mathcal{C}\}$. If it is not true, then we can replace $\mathcal{C}$ by $\tilde{\mathcal{C}}$ in $\mathcal{B}$ and obtain another basis. We are showing here that this basis has a smaller weight, which cannot be possible since $\mathcal{B}$ is a minimum basis.

  Let us define c as the node in $\mathcal{C}$ that is the closest from s (in case, of several closest nodes, we can choose it arbitrarily). We assume that c is in the path in $\mathcal{C}$ that goes from a to u without passing through b, as

the proof for the converse is similar. We denote by P_1, P_2 and P_3 the paths in $\mathcal{C}$ that goes respectively from u to c, c to a and b to u without passing through the edge (a, b). Finally, we define P_4 as a shortest path from c to s chosen arbitrarily. An illustration of $\mathcal{C}$ and $\tilde{\mathcal{C}}$ is given in figure 3.2 to help visualize the proof. [1]

Clearly, we have $\mathcal{C} = P_1 + P_2 + (a, b) + P_3$ and $w(\mathcal{C}) = l(P_1) + l(P_2) + l(a, b) + l(P_3) + 2l(P_4) + l(a, b)$.

We are assuming that theorem 9 is false, hence one of those inequality is strict:

- $l(SP(u, a)) \leq l((a, b)) + l(P_3)$
- $l(SP(a, s)) \leq l(P_2) + l(P_4)$
- $l(SP(s, u)) \leq l(P_1) + l(P_4)$

$\tilde{\mathcal{C}}$ has thus a strictly smaller weight than $\mathcal{C}$, and if we could replace $\mathcal{C}$ by this element, $\mathcal{B}$ would not be a minimum basis. Thus, we conclude that $\mathcal{C} \in span(\mathcal{B}_\mathcal{C})$.

- Secondly, there exists an edge $e = (v, w)$ of $\mathcal{C}$ such that $SP(u, v) + (v, w) + SP(w, s) + SP(s, u)$ is not generated by $span(\mathcal{B}_\mathcal{C})$ for some arbitrary shortest paths. Let us consider that it is not true.

  Then for each edge $(v', w') \in \mathcal{C}$, there exists three shortest paths $SP(u, v')$, $SP(w', s)$ and $SP(s, u)$ such that $\mathcal{C}_{(v',w')} = SP(u, v') + (v', w') + SP(w', s) + (s, u) \in span(\mathcal{B}_\mathcal{C})$.

  The sum of all those cycles is given by :

$$\begin{aligned}\sum_{e' \in \mathcal{C}} \mathcal{C}_{e'} =& \mathcal{C} + \sum_{(v',w') \in \mathcal{C}} SP(u, v') + SP(w', s) + SP(s, w') \\ & \mathcal{C} + \sum_{\text{node } a \text{ in } \mathcal{C}} SP(u, a) + SP(a, s) + SP(s, u)\end{aligned}$$

  Using the previous result, $SP(u, a)+SP(a, s)+SP(s, u) \in span(\mathcal{B}_\mathcal{C})$ for each node a of the cycle. Since we also have that $\sum_{e' \in \mathcal{C}} \mathcal{C}_{e'} \in span(\mathcal{B}_\mathcal{C})$, we find that $\mathcal{C}$ is not independent from $\mathcal{B}_\mathcal{C}$, which is not possible because $\mathcal{B}$ is a basis.

- Finally, we know from the last part that there exists an edge (v, w) such that $\mathcal{C}$ can be replaced in $\mathcal{B}$ by $\tilde{\mathcal{C}} = SP(u, v) + (v, w) + SP(w, s) +$

[1] This illustration is only an example ; s or c may not belong to $\tilde{\mathcal{C}}$, neither P_1 nor P_3.

$SP(s,u)$. Let us use again our notation P_1, P_2, P_3 and P_4 characterizing $\mathcal{C}$ by replacing a by w and b by v. We have at least that $l(SP(u,v)) < l(P_3))$ or $l(SP(w,s)) < l(P_2)+l(P_4)$ or $l(SP(s,u)) < l(P_1)+l(P_4)$ from our assumption that our theorem is false. Thus $\tilde{\mathcal{C}}$ has smaller weight than $\mathcal{C}$, which would make the basis $\mathcal{B} \setminus \{\mathcal{C}\} \cup \tilde{\mathcal{C}}$ better than $\mathcal{B}$. This is impossible because $\mathcal{B}$ is optimal and it proves that the theorem is correct □.

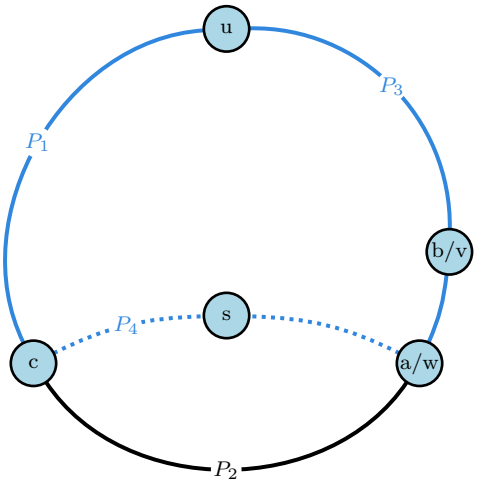


Figure 3.2: Illustration of the cycle $\mathcal{C}$ (plain lines) and $\tilde{\mathcal{C}}$ (in blue) in the proof of theorem 9

From theorem 9, we can extend the Horton's set to our fixed source setting. We can thus for each node u and each edge (v,w), build a cycle composed of a shortest $w-s$, $s-u$ and $u-v$ path plus (v,w). As the set of those cycles contains a fixed source minimum basis of $\mathcal{G}$, we can apply the greedy algorithm.

Theorem 10 (**Extension of theorem 8**). *The matroid greedy algorithm applied to the fixed source Horton set returns a fixed source minimum cycle basis.*

Proof. The proof is basically the same as theorem 8. □

This conclude our discussion about the fixed source minimum weight cycle basis problem.

3.4 Generalization

This setting can be generalized to any observation given as a linear function of the data.

Instead of asking for the values of particular edges or cycles, we define a linear operator $\mathbf{\Psi} \in \mathbb{C}^{K \times N}$ and ask for $b = \mathbf{\Psi} f$. Similarly to the previous proof, we can perfectly recover f if and only if $rank(\mathbf{\Psi C}) = K$. In cycle sampling, $\mathbf{\Psi} = \tilde{\mathbf{C}}^T$, while in the edge sampling setting, $\mathbf{\Psi}$ corresponds to a $rank(K)$ submatrix of the $N \times N$ identity matrix.

Chapter 4

Noisy sampling of conservative flows

As we have seen when designing an edge sampling procedure, every spanning tree corresponds to a valid sampling set of edges for perfect recovery, and vice versa. In general, a graph has a lot of different spanning trees, up to N^{N-2} for the complete graph of N nodes[1] and we have thus a lot of freedom in the choices of those sampled edges. A natural question is to determine if all choices are equivalent or if some clever pick of the sampled edges gives interesting properties. Here, we will explore this question through the lens of noisy sampling.

4.1 Minimum total error sampling problem

We use the same model as the previous section with the only difference being that instead of getting the exact value f_i of each sampled flow, we get a noisy signal $\tilde{f}_i = f_i + e_i$. We consider that the noise e_i are *iid* and zero-mean but do not make any additional assumption about their distribution.

Applying algorithm 2 with a fundamental basis $\mathcal{B}$, we recover the flow $\tilde{f} = \sum_{c_i \in \mathcal{B}} \tilde{f}_i c_i = \sum_{c_i \in \mathcal{B}} f_i c_i + \sum_{c_i \in \mathcal{B}} e_i c_i = f + \sum_{c_i \in \mathcal{B}} e_i c_i$. The total error is thus directly given by $e = \sum_{c_i \in \mathcal{B}} e_i c_i$ and only depends on the choice of this fundamental basis. Moreover, this noise e is a conservative flow and we can bound its effect for the ℓ_p norm :

[1] According to *Cayley's formula*

$$
\begin{aligned}
\|e\|_p &= \left(\sum_{j=1}^{M} \left| \sum_{i=1}^{K} e_i c_{i,j} \right|^p \right)^{\frac{1}{p}} \leq \left(\sum_{j}^{M} \sum_{i}^{K} |e_i|^p \, |c_{i,j}|^p \right)^{\frac{1}{p}} \\
&\leq \left(\sum_{j}^{M} \sum_{i}^{K} E^p \, |c_{i,j}|^p \right)^{\frac{1}{p}} \text{ with } E = \max(|e_0|, \ldots, |e_K|) \\
&= E \left(\sum_{i}^{K} l(c_i) \right)^{\frac{1}{p}} \text{ with } l(c_i) \text{ the length of the cycle } c_i
\end{aligned} \tag{4.1}
$$

Therefore, we are looking for the fundamental basis $\mathcal{B}^*$ minimizing the sum of the length of the cycles of the basis. While we have shown in the last chapter that finding the minimum weight cycle basis can be solved in polynomial time with Horton's algorithm, this is not the case anymore when dealing only with fundamental cycles. Unfortunately, finding such a basis is an $\mathcal{NP}$-Hard problem [DPK82]. In fact, this is even an $\mathcal{APX}$-Hard problem [GA03], which means that there does not exist any polynomial algorithm that approximate the optimal solution within a constant factor unless $\mathcal{P} = \mathcal{NP}$. An example of a graph for which any minimum cycle basis is not fundamental is given in figure 4.1.

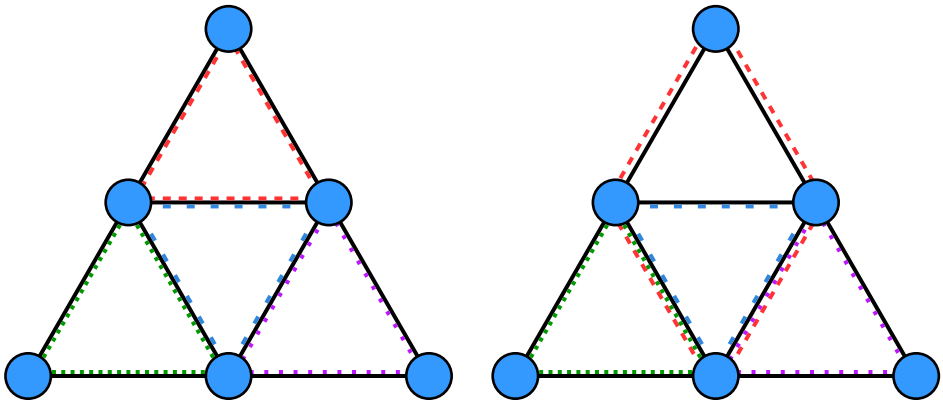
We call this problem the *Minimum total error sampling problem.*

4.2 Minimum local error sampling problem

Since it is infeasible to minimize the total error on our recovered signal, we will focus in this section on another objective : finding the set of sampling edges that minimize the maximal error among all edges. In other words, this corresponds to the ℓ_∞ norm of the error. We call this problem the *Minimum local error sampling problem*, in contrast with the *total error* of the last section. Similarly to equation 4.1, we can bound this error :

$$
\|e\|_\infty = \max_{j=1,\ldots,M} \left| \sum_{i=1}^{K} e_i c_{i,j} \right| \leq E \max_{j} \sum_{i}^{K} |c_{i,j}| \tag{4.2}
$$

For a set of sampled edges, the local error is thus proportional to the maximal number of time an edge is included in a cycle of the fundamental



(a) minimal length cycle basis (b) minimal length fundamental basis

Figure 4.1: Example of the minimal length cycle basis of the triforce graph with and without restriction to fundamental basis. It is impossible to build a fundamental basis with four cycles of length three.

basis associated to the sampling scheme. To minimize the local error, we search a fundamental basis minimizing the largest number of time an edge is taken in a cycle of the basis. I propose to call this value the fundamental basis number as an extension of the concept of basis number introduced by Schmeichel in 1981 [Sch81] to fundamental basis :

A basis of the cycle space $CS(\mathcal{G})$ is called k-fold if each edge of $\mathcal{G}$ occurs in at most k of the cycles of the basis. The basis number (resp. fundamental basis number) of $\mathcal{G}$, denoted by $b(\mathcal{G})$ (resp. $b_f(\mathcal{G})$) is the smallest integer k such that $CS(\mathcal{G})$ has a k-fold basis (resp. fundamental basis).

From now, we will call a basis whose fold correspond to the basis number an *optimal* basis.

In its original paper, Schmeichel found the basis number of some classes of graphs, such as the complete and the n-cube graphs. Some papers extended his results to other classes of graphs as [AJ04] [AW98] [Jar03] for example, but we do not know a polynomial algorithm finding this basis number for a general graph, let alone the complexity class of the problem. However, a much older result, the well-known *Mac Lane's planarity criterion*, links the basis number of a graph to its planarity. [Mac37]

Theorem 11 (Mac Lane's planarity criterion, 1937)**.** *An undirected graph is planar if and only if the cycle space of the graph has a cycle basis in which*

each edge of the graph is in a most two basis cycle, ie if the basis number of the graph is strictly less than three.

In the following of the chapter, I will propose a generalization of the Mac Lane's theorem for fundamental bases, as well as the fundamental basis number of some classes of graph,

4.2.1 Induced paths representation

Before getting into the heart of the matter, I would like to introduce an alternative representation of fundamental basis number and optimal basis.

Let $\mathcal{T}$ be a spanning tree of $\mathcal{G}$ and (u, v) an edge of $\mathcal{E}$ not in the spanning tree. We call the unique shortest path between u and v in $\mathcal{T}$ the induced $u - v$ path and note it as $P^{\mathcal{T}}_{u,v}$. We denote the number of induced paths passing through an edge as the *load of the edge* and its maximal value among all edges of the tree as the *load of the tree*.

There is a clear relation between the load of a tree and the k-fold concept of its induced basis :

Theorem 12. *Let $\mathcal{B}$ be a fundamental basis of $CS(\mathcal{G})$ and $\mathcal{T}$ one of its associated spanning tree. Then $\mathcal{B}$ is k-fold if and only if the load of $\mathcal{T}$ is at most k.*

Proof. Each cycle of $\mathcal{B}$ can be written as $(u, v) + P^{\mathcal{T}}_{u,v}$ for some edge $(u, v) \notin \mathcal{T}$ by definition. Furthermore, each edge taken in a cycle is also taken in the induced paths (and vice versa), except the cycle edges which will never be the only most taken edges. We conclude that those two quantities are equal by definition. □

An optimal basis of a graph is thus given by any spanning tree with minimal load ; we will call such trees *optimal spanning trees* onwards. This change of representation will prove to be very useful to approach the problem from another perspective, especially because spanning trees are way easier to apprehend and visualize.

4.2.2 Characterization of 2-fold fundamental cycle basis

Thanks to the Mac Lane planarity criterion, we know that a graph is planar if and only if its basis number is at most 2. Graphs with 2-fold fundamental cycle basis are thus inevitably planar, but which other properties do they have ? We will denote the class of such graphs as $\mathcal{B}_f^2$ and characterize it in the following.

First, a planar graph has not necessarily a 2-fold fundamental basis. For example the triforce graph, given in Figure 4.1 is planar but has at least a 3-fold basis. However, the class of Halin graphs $\mathcal{H}$ is a better candidate to characterize $\mathcal{B}_f^2$. [SP06]

$\mathcal{G}$ is called an Halin graph if it can be built from a planar embedding of a tree by connecting with a cycle all the leaves in their clockwise order in the embedding. It is pretty clear that any Halin graph has a 2-fold fundamental basis (lemma 4).

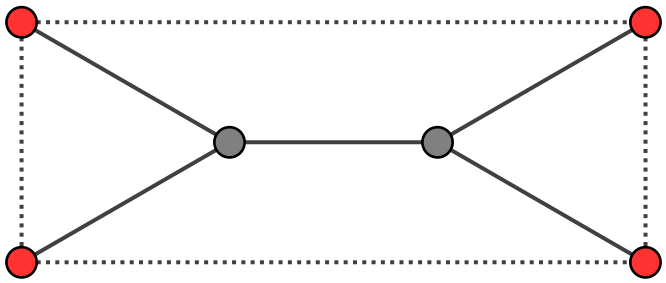


Figure 4.2: Example of Halin graph. Red nodes are the leaves and dotted edges are the cycle edges.

Lemma 4. If $\mathcal{G}$ is an Halin graph, then $b_f(\mathcal{G}) = 2$.

Proof. Let $\mathcal{B}$ be the fundamental basis associated to the tree used in the graph construction. Each cycle of $\mathcal{B}$ represents an inner face of the planar graph. But as each edge of the tree is adjacent to two faces, it belongs to only two cycles. □

Nevertheless, not every graph of $\mathcal{B}_f^2$ is an Halin graph. For example, connecting 2 Halin graphs with an edge does not result in another Halin graph, while still having a 2-fold basis. Also, removing an outer edge of an

Halin graph is equivalent to remove a cycle of the fundamental basis, which thus remains 2-fold while not Halin anymore.

To circumvent this issue, we introduce the class of *hole-free graphs* $\mathcal{H}_{\mathcal{F}}$ and *hole-free complex* $\mathcal{H}_{\mathcal{F}}^{+}$.

We call a graph hole-free if it can be built from a Halin graph for which each outer edge can be removed and possibly replaced by a sequence of edges $(u_0, v_0), \ldots, (u_n, v_n)$ such that :

- u_i, v_i are in the induced path of the edge that has been replaced
- u_i precedes v_i in the induced path
- if $i < j$, then either $v_i = u_j$ or v_i precedes u_j in the induced path

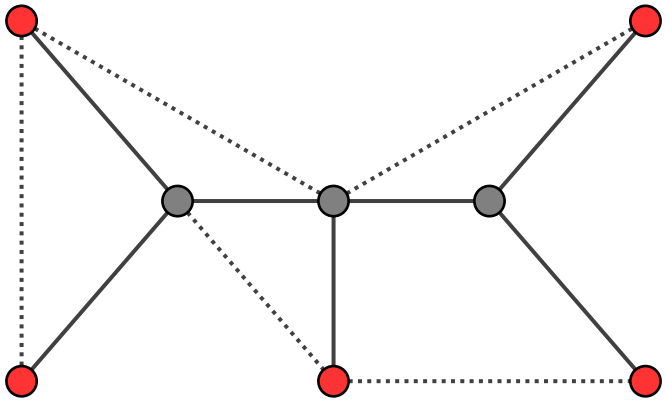


Figure 4.3: Example of hole-free graph. Red nodes are the leaves and dotted edges are the cycle edges.

Unsurprisingly, every hole-free graph has also a $2 - fold$ fundamental basis since it is planar.

Lemma 5. If $\mathcal{G}$ is a hole-free graph, then $b_f(\mathcal{G}) \leq 2$.

Hole-free complexes introduce the concept of *biconnected components* : they are graphs whose biconnected components are hole-free. A graph is said *biconnected* if it remains connected despite the removal of any node and naturally, a biconnected component is a maximal biconnected subgraph. Biconnected graphs are particularly relevant when studying cycles basis since each edge is guaranteed to belong to at least one cycle. Even more, if we

have a fundamental basis $\mathcal{B}_i$ for each biconnected component of a graph, the union of the bases $\bigcup_i \mathcal{B}_i$ is a fundamental basis of the graph. It allow us to work on the several biconnected components of a graph separately and take advantage of the biconnectevity.

While it was straightforward to show that $\mathcal{H}_{\mathcal{F}}^{+} \subset \mathcal{B}_f^2$, the other inclusion is way harder. The following intermediate lemma will be of a great help later.

Lemma 6. Let $\mathcal{G}$ be a biconnected graph with fundamental basis number equal to at most two and $\mathcal{T}$ an optimal tree. Then each inner node u has at most two adjacent edges with load equal to one.

Moreover, it has exactly one adjacent edge with load one if and only if it is adjacent to exactly one edge not in $\mathcal{T}$.

Proof. Let assume there exists an inner node u of the graph with at least three adjacent edges with load equal to one that we will denote (u, a), (u, b) and (u, c).

As $\mathcal{G}$ is biconnected, there should exist two paths P_1 and P_2 from a to b and from b to c that do not pass through u. The cycles $(u, a) + P_1 + (b, u)$ and $(u, b) + P_2 + (c, u)$ are independent and thus they both have at least one cycle edge. So, there are two induced paths from these cycles that pass through u via (a, u), (b, u) and(c, u), and as one of those edge has to be taken by the two paths, its load cannot be one.

Moreover, each induced path passing through one node has to leave it, except if the node is the starting or ending point of the path. In that case, the node is one endpoint of the cycle edge inducing the path.

It is thus clear that the sum of the load of the adjacent edges of a node is even if the node is not adjacent to any cycle edge and odd otherwise. □

Thanks to this lemma, we can now prove that $\mathcal{B}_f^2 \subset \mathcal{H}_{\mathcal{F}}^{+}$.

Lemma 7. If $b_f(\mathcal{G}) \leq 2$ and $\mathcal{G}$ is biconnected, then $\mathcal{G}$ is an hole-free graph.

Proof. Let $\mathcal{C}$ be an optimal basis of $\mathcal{G}$ and $\mathcal{T}$ one of its associated spanning tree.

We are going to transform $\mathcal{G}$ into an Halin graph $\tilde{\mathcal{G}}$ without changing its basis number in three steps :

1. First, we will transform the graph until there does not exist any inner node u adjacent to exactly one cycle edge (u, v).

   If one of them exists, it has at least two adjacent edges in $\mathcal{T}$ since it is not a leaf, and exactly one of those neighbouring edges has a load equal to one because of lemma 6. We will note this adjacent edge as (u, w) and transform the graph $\mathcal{G}$ into $\mathcal{G}'$ by replacing (u, v) by (u, w).

   $\mathcal{G}'$ has the same fundamental basis number as $\mathcal{G}$ because only one induced path in the underlying tree got extended by an edge, and this edge has a load of one. Finally, we repeat this basic transformation step until no such inner node adjacent to only one cycle edge exists anymore. It is worth mentioning that this process cannot run indefinitely as the weight of the fundamental basis gets increased by one at each step and cannot excess $2(M - N + 1)$.

2. Secondly, we will transform the graph until each leaf is adjacent to exactly two cycle edges.

   Let u be a leaf for which it is not the case. Since the graph is biconnected, u is adjacent to exactly one cycle edge, and the load of its adjacent edge in the tree is then equal to one. As each inner node is adjacent to either zero, either two edges having load equal to one (lemma 6), we can thus build a path from this leaf to another leaf v adjacent to only one cycle edge.

   For each such pair of leaves, we transform $\mathcal{G}$ into $\mathcal{G}'$ by adding the edge (u, v). As before, the fundamental basis number of the graph does not change since the new induced path in $\mathcal{T}$ contained only nodes having a load equal to one.

3. Finally, we will transform the graph into an Halin graph.

   Let e be a cycle edge between a leaf u and an inner node. As each inner node is adjacent to two cycle edges, we can build a path from this leaf to another leaf v using only cycle edges.

   For each such pair of leaves, we transform $\mathcal{G}$ into $\mathcal{G}'$ by adding the edge (u, v) and removing all the edges of the path. As the new induced path is the concatenation of the induced path of the removed edges, the fundamental basis number remains the same.

After those three steps, the new graph $\tilde{\mathcal{G}}$ is an Halin graph.

Indeed, there cannot be any inner node adjacent to a cycle edge, as each edge of the tree is already in two induced paths from the outer cycle edges

linking the leaves. Furthermore, each leave in $\mathcal{T}$ is connected by a cycle composed of the edges of $\mathcal{G}$ not in $\mathcal{T}$ and none of our edges addition or replacement breaks the planar embedding.

Finally, we can build the original graph $\mathcal{G}$ as an hole-free graph from this Halin graph and this conclude the proof. □

From lemma 5 and 7, we can finally fully characterize the class of graphs having a fundamental basis number of at most two and extend the Mac Lane planarity criterion to fundamental cycle bases :

Theorem 13 (2-fold basis characterization)**.** *An undirected graph is a hole-free complex if and only if the cycle space of the graph has a fundamental cycle basis in which each edge of the graph is in a most two basis cycle, ie if the fundamental basis number of the graph is strictly less than three.*

To summarize, we have :

$$\mathcal{H} \subset \mathcal{H}_{\mathcal{F}} \subset \mathcal{H}_{\mathcal{F}}^{+} = \mathcal{B}_{f}^{1} \subset \mathcal{P} = \mathcal{B}^{2}$$

with $\mathcal{P}$ the class of planar graphs.

4.2.3 Fundamental basis number of the complete graph

In 1981, Schmeichel introduced the concept of basis number [Sch81] and gave its value for several common classes of graph. In particular, he cleverly showed that the basis number of any complete graph with at least 5 nodes is 3.

To conclude this chapter, we will show below that while the basis number of K_N was constant and small, as soon as fundamental cycles are concerned, we cannot do better than have a basis with edges taken in a linear number of cycles.

Theorem 14 (Fundamental basis number of the complete graph)**.** *For every $N \geq 3$, we have $b_f(K_N) = N - 2$.*

Proof.

1. $b_f(K_N) \leq N - 2$

   Let's consider a star-tree as the sampling spanning tree. Each tree-edge is used by exactly $N - 2$ induced paths which link the leaf of the tree-edge to the $N - 2$ other leaves.

2. $b_f(K_N) \geq N - 2$

   The minimum cut of K_N is $N - 2$. □

Conclusion

There are definitely a lot to say about data sampling on graphs. In this master thesis, we have explored several problems under this theme, sampling from spatially-smooth signals lying on nodes to conservative flows through cycles.

Although the study of signal lying on nodes of a network through tools of signal processing, *Graph Signal Processing* , is the most mature domain we have considered, it is still quite recent as it appeared around 2013. Even more, the extension of the sampling concept to Graph Signal Processing has been firstly proposed in 2015. Here, we reviewed the building blocks of this emergent field and gave a first attempt to generalize this framework to spatiotemporal data through *Kronecker algebra*.

Concerning the sampling of data signal lying on the edges of a graph, we restricted ourselves to *conservative flows* . First, because a lot of attention is brought to this class of signal since it has many applications. But also, the structure of the space of conservative flows, known as the *cycle space* , leads to very interesting, diverse and elegant problems.

One of those problem is the *fixed source minimum weight cycle basis* problem, a completely new task I introduced and solved in chapter 3. My solution, based on Horton's algorithm, highlighted the matroid structure of cycle spaces which has particularly useful properties.

I also introduced the *Minimum local error sampling problem* in the setting of sampling noisy observations, which consists in finding the *fundamental basis number* of a graph. Despite a lot of efforts, I did not manage to find an efficient algorithm to solve it, neither to show that this problem is hard, which I strongly suspect. However, I managed to characterize the graph having a basis number equal to two and I proposed an extension of Mac Lane's planarity criterion to fundamental cycle basis.

Finally, as this is an emergent field, there are still a lot to explore. In the spatiotemporal node signal framework, we can for example design other sampling procedures to spread the observations and the load of computations across all nodes. It would also be interesting to design a distributed message-passing protocol to sample and interpolate the signal without need of a centralized entity. About edge signals, future directions include considering other prior than the conservation of flows. For example, it could be worth studying in a less theoretical framework a mixed prior, a signal that is conservative and spatially-smooth for example. I could also imagine further work in the noisy setting framework by including redundancy to limit the effect of the noise.

Bibliography

[AJ04] MY Alzoubi and MMM Jaradat. "The basis number of the composition of theta graphs with stars and wheels". In: *Acta Mathematica Hungarica* 103.3 (2004), pp. 255–263.

[AW98] Salar Y Alsardary and Jerzy Wojciechowski. "The basis number of the powers of the complete graph". In: *Discrete mathematics* 188.1-3 (1998), pp. 13–25.

[Ber01] C. Berge. *The Theory of Graphs*. Dover books on mathematics. Dover, 2001, pp. 23–52. ISBN: 9780486419756.

[BGV09] Franziska Berger, Peter Gritzmann, and Sven de Vries. "Minimum Cycle Bases and Their Applications". In: *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation*. Ed. by Jürgen Lerner, Dorothea Wagner, and Katharina A. Zweig. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 34–49. ISBN: 978-3-642-02094-0. DOI: 10.1007/978-3-642-02094-0_2. URL: https://doi.org/10.1007/978-3-642-02094-0_2.

[BSC18] Sergio Barbarossa, Stefania Sardellitti, and Elena Ceci. "Learning from Signals Defined over Simplicial Complexes". In: *2018 IEEE Data Science Workshop, DSW 2018, Lausanne, Switzerland, June 4-6, 2018*. 2018, pp. 51–55. DOI: 10.1109/DSW.2018.8439885. URL: https://doi.org/10.1109/DSW.2018.8439885.

[Che+15] Siheng Chen et al. "Discrete Signal Processing on Graphs: Sampling Theory". In: *IEEE Transactions on Signal Processing* 63.24 (2015), pp. 6510–6523.

[Chu97] Fan R.K Chung. *Spectral graph theory*. American Mathematical Society, 1997.

[CRT06] E. J. Candes, J. Romberg, and T. Tao. "Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information". In: *IEEE Transactions on Information*

Theory 52.2 (Feb. 2006), pp. 489–509. ISSN: 0018-9448. DOI: 10.1109/TIT.2005.862083.

[DH+] Jean-Charles Delvenne, Julien Hendrickx, et al. "Matroid problems". In: *LINMA2450 Combinatorial Optimization lecture notes*, pp. 31–34.

[Don+06] David L Donoho et al. "Compressed sensing". In: *IEEE Transactions on information theory* 52.4 (2006), pp. 1289–1306.

[DPK82] N Deo, G Prabhu, and Mukkai Krishnamoorthy. "Algorithms for Generating Fundamental Cycles in a Graph". In: *ACM Trans. Math. Softw.* 8 (Mar. 1982), pp. 26–42. DOI: 10.1145/355984.355988.

[GA03] Giulia Galbiati and Edoardo Amaldi. "On the Approximability of the Minimum Fundamental Cycle Basis Problem". In: Sept. 2003, pp. 151–164. DOI: 10.1007/978-3-540-24592-6_12.

[Hor87] Joseph Horton. "A Polynomial-Time Algorithm to Find the Shortest Cycle Basis of a Graph". In: *SIAM J. Comput.* 16 (Apr. 1987), pp. 358–366. DOI: 10.1137/0216026.

[HVG11] David K. Hammond, Pierre Vandergheynst, and Rémi Gribonval. "Wavelets on graphs via spectral graph theory". In: *Applied and Computational Harmonic Analysis* 30.2 (2011), pp. 129–150. ISSN: 1063-5203. DOI: https://doi.org/10.1016/j.acha.2010.04.005. URL: http://www.sciencedirect.com/science/article/pii/S1063520310000552.

[Jar03] MMM Jaradat. "On the basis number of the direct product of graphs". In: *Australasian Journal of Combinatorics* 27 (2003), pp. 293–306.

[Lau05] Alan J. Laub. *Matrix Analysis for Scientists and Engineers*. Society for Industrial and Applied Mathematics, 2005.

[Mac37] Saunders Mac Lane. "A combinatorial condition for planar graphs". eng. In: *Fundamenta Mathematicae* 28.1 (1937), pp. 22–32. URL: http://eudml.org/doc/212919.

[MM09] Kurt Mehlhorn and Dimitrios Michail. "Minimum Cycle Bases: Faster and Simpler". In: 6 (Dec. 2009). DOI: 10.1145/1644015.1644023.

[Ort+18] A. Ortega et al. “Graph Signal Processing: Overview, Challenges, and Applications”. In: *Proceedings of the IEEE* 106.5 (May 2018), pp. 808–828. ISSN: 0018-9219. DOI: 10.1109/JPROC.2018.2820126.

[Sch81] Edward F Schmeichel. “The basis number of a graph”. In: *Journal of Combinatorial Theory, Series B* 30.2 (1981), pp. 123–129. ISSN: 0095-8956. DOI: https://doi.org/10.1016/0095-8956(81)90057-5. URL: http://www.sciencedirect.com/science/article/pii/0095895681900575.

[SP06] Maciej Syslo and Andrzej Proskurowski. “On Halin graphs”. In: Nov. 2006, pp. 248–256. DOI: 10.1007/BFb0071635.

[SS18] Michael T. Schaub and Santiago Segarra. “Flow smoothing and denoising: Graph Signal Processing in the edge-space”. In: *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)* (2018), pp. 735–739.

Appendices

Perfect recovery edge sampling

```
def spanning_tree(G):
    """ Compute a spanning tree of G

    Keyword arguments:
    G -- a tuple (V, E) ; represents a graph with nodes V and
         edges E
        V -- a list ; each element of the list represents a
↪   node
        E -- a list of tuples (u, v); each element of the
↪   list
              represents an edge between nodes u and v
            u, v -- two int ;  index of the nodes in the list
↪   V
    """
    V, E = G
    adj_list = [[] for i in range(len(V))]

    for e in E:
        u, v = e
        adj_list[u].append(v)
        adj_list[v].append(u)

    dfs_stack = []
    dfs_stack.append(V[0])
    visited = [False] * len(V)
    visited[V[0]] = True
    tree = []

    while(len(dfs_stack)):
        curr = dfs_stack[-1]
        dfs_stack.pop()

        for neigh in adj_list[curr]:
            if not visited[neigh]:
                visited[neigh] = True
                dfs_stack.append(neigh)
                tree.append((curr, neigh))
```

```
    return tree


def sample(G):
    """ Compute a valid set of sampling edges for G

    Keyword arguments:
    G -- a tuple (V, E) ; represents a graph with nodes V and
         edges E
    """
    _, E = G
    T = spanning_tree(G)
    M = []

    tree_set = set(T)

    for i, e in enumerate(E):
        u, v = e
        if (u, v) not in tree_set and (v, u) not in tree_set:
            M.append(i)

    return M


def get_tree_path(T, x, y):
    """ Find a path from nodes x to y in tree T

    Keyword arguments:
    T -- a tuple (V, E) ; represents a tree with nodes V and
         edges E
    x, y -- two elements of V
    """
    _, E = T
    adj_list = [[] for i in range(len(V))]

    for i, e in enumerate(E):
        u, v = e
        adj_list[u].append(v)
        adj_list[v].append(u)

    def dfs(x, y, par):
```

```
        if x == y:
            return []
        else:
            for neigh in adj_list[x]:
                if neigh != par:
                    ans = dfs(neigh, y, x)
                    if ans != -1:
                        ans.append((x, neigh))
                        return ans
            return -1
    return dfs(x, y, -1)


def interpolate(G, M, f):
    """ Recover a conservative flow f in the edges of a graph
        G from a set of sampled edges M.

    Keyword arguments:
    G -- a tuple (V, E) ; represents a graph with nodes V and
         edges E
    M -- a list of int ; each element of the list represents
         the index of a sampled edge between nodes u and v
    f -- a numpy array of |E| elements;
        f[i] equals the measurement of the flow on the i-th
↪   edge
        if it is sampled (ie. if it is in M), or 0 otherwise
    """
    V, E = G
    E_t = []
    sampling_set = set(M)

    edges_indices = {}
    for i, e in enumerate(E):
        u, v = e
        edges_indices[(u, v)] = (i, 1)
        edges_indices[(v, u)] = (i, -1)

    for i, e in enumerate(E):
        if i not in sampling_set:
            E_t.append(e)
```

```
    for ei in M:
        u, v = E[ei]
        p = get_tree_path((V, E_t), v, u)

        for ep in p:
            i, sign = edges_indices[ep]
            f[i] += sign*f[ei]

    return f


def leaves_cutting_interpolation(G, M, f):
    """ Recover efficiently a conservative flow f in the
    ↪  edges of
        a graph G from a set of sampled edges M.

    Keyword arguments:
    G -- a tuple (V, E) ; represents a graph with nodes V and
         edges E
    M -- a list of int ; each element of the list represents
         the index of a sampled edge between nodes u and v
    f -- a numpy array of |E| elements;
        f[i] equals the measurement of the flow on the i-th
↪   edge
        if it is sampled (ie. if it is in M), or 0 otherwise
    """
    V, E = G

    adj_list = [[] for i in range(len(V))]
    for i, e in enumerate(E):
        u, v = e
        adj_list[u].append((v, i, 1))
        adj_list[v].append((u, i, -1))

    marked_edges = [False] * len(E)
    remaining_degree = [len(adj_list[i]) for i in
    ↪  range(len(V))]
    net_flow = [0] * len(E)

    leaves_stack = []
    for i in M:
```

```
            u, v = E[i]

            remaining_degree[u] -= 1
            remaining_degree[v] -= 1
            net_flow[u] -= f[i]
            net_flow[v] += f[i]

            if remaining_degree[u] == 1:
                leaves_stack.append(u)
            if remaining_degree[v] == 1:
                leaves_stack.append(v)

            marked_edges[i] = True

        while(len(leaves_stack)):
            curr = leaves_stack[-1]
            leaves_stack.pop()

            for e in adj_list[curr]:
                neigh, i, sign = e
                if not marked_edges[i]:
                    marked_edges[i] = True
                    f[i] = sign * net_flow[curr]
                    remaining_degree[neigh] -= 1
                    net_flow[neigh] += sign * f[i]

                    if remaining_degree[neigh] == 1:
                        leaves_stack.append(neigh)
        return f
```

Matroid greedy algorithm

```
from abc import ABC, abstractmethod

class independent_set(ABC):
    @abstractmethod
    def add(self, x):
        pass
```

```
    @abstractmethod
    def is_independent():
        pass


    @abstractmethod
    def clear():
        pass


def greedy(E, w, F):
    """ Compute a minimum weighted basis of E

    Keyword arguments:
    E -- A set or a list of object
    w -- a function taking an object of E as argument and
         returning its weight
    F -- A class implementing the abstract class
         "independent_set"
    """
    L = list(E)
    L.sort(key=w)

    B = F()
    for x in L:
        if B.is_independent(x):
            B.add(x)
        pass
    return B
```

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**