Laboratoire d'Etude et de Recherche
en Informatique d'Angers

Calcul d'une base de cycles dans un graphe

Par Vincent IMARD et Kyanou LOW-KEIN

Sous la direction de Eric MONFROY

stage réalisé entre le 06/04/2021 et le 01/06/2021

Remerciements

Sous souhaitons remercier M. Eric MONFROY pour la proposition et l'encadrement de ce stage. Il s'est toujours montré disponible et à l'écoute. Il a également parfaitement su nous guider et conseiller, tout en nous donnant des exemples d'applications, ce qui a rendu le traitement du sujet très agréable.

Table des matières

Résumé

Dans ce rapport, nous étudions les deux principaux algorithmes de calcul d'une base minimale de l'espace des cycles d'un graphe non orienté, l'algorithme de Horton et l'algorithme de De Pina amélioré. Nous donnerons les bases théoriques nécessaires à la compréhension du sujet avant de comparer leurs performances. Les comparaisons seront faites sur le temps de calcul ainsi que sur la longueur du plus long cycle de la base calculée. Ce cycle devant être le plus petit possible en vu d'une application au Cyclic Bandwidth Problem. Les comparaisons montrerons que l'algorithme de Horton est plus rapide sur certains graphes que l'algorithme de De Pina malgré une complexité supérieure. Nous tenterons d'expliquer pourquoi et chercherons à savoir sur quels graphes l'algorithme de Horton est le plus rapide. Nous donnerons également des pistes d'améliorations afin d'affiner les résultats que nous avons obtenu.

1 Introduction

Soit $G = (V, E)$ un graphe non orienté, non pondéré, fini, sans boucle et sans arête multiples. On pose $|V| = n$ et $|E| = m$. Soit $\mu : E \to \{0, 1\}^m$ une application permettant d'obtenir la représentation vectorielle d'une arête de G. On définit l'espace des cycles de G par $\mathcal{C}_G = \{\mu(C) | \text{C élémentaire dans G}\}$. Ce stage avait pour objectif l'étude et la comparaison d'algorithmes permettant de calculer une base de cet espace des cycles. Pour ce faire, il existe plusieurs algorithmes divisés en deux catégories :

— Les algorithmes de calcul d'une base quelconque : algorithme de Porter, algorithme "Naïf" (principalement utile mathématiquement),...

— Les algorithmes de calcul d'une base minimale (i.e de poids minimal) : algorithme de Horton, de De Pina, d'Amaldi,...

Il est important de noter que dans le contexte de ce stage, les graphes sont non orientés. Il existe cependant bien des algorithmes de calcul de base minimale pour des graphes orientés, dont on ne parlera pas. Ceux-ci sont d'ailleurs tous basés sur l'approche de Horton ou de De Pina, voir les deux, et sont pour la majorité des algorithmes de Monte Carlo. Nous avons donc principalement fait nos tests sur l'algorithme de Horton et l'algorithme de De Pina. Les benchmark ont été réalisés sur les instances de Harwell-Boeing. Ces instances sont des matrices issues d'applications industrielles telle que la Chimie, la Physique des matériaux, la Relativité générale ou encore des

maillages utilisés par la méthode des éléments finis.

Dans les tests, deux résultats nous intéressaient particulièrement. Il s'agit du temps de calcul et de la longueur du plus long cycle de la base calculée. Nous avons donc comparé nos deux algorithmes sur ces caractéristiques.

La finalité de ces comparaisons est l'application au problème de la bande passante cyclique (Cyclic Bandwidth Problem). Il s'agit un problème d'optimisation d'étiquetage de graphe. Un étiquetage est une fonction associant à chaque sommet, un élément d'un autre ensemble. Dans le cadre de la bande passante cyclique, l'étiquetage est la fonction $etiq : V \to \{1, 2, \ldots, n\}$ et la bande passante cyclique de $etiq$ pour le graphe G est donnée par la formule suivante

$$B_c(G, etiq) = \max_{\{u,v\} \in E} \{|etiq(u) - etiq(v)|_n\}$$

avec $|a|_n = \min\{|a|, n - |a|\}$.

L'objectif du Cyclic Bandwidth Problem est de trouver une étiquetage $etiq^*$ tel que $B_c(G, etiq^*)$ soit minimisé, on note $B_c(G)$ cette valeur. Le lien entre les deux problèmes a été trouvé par Hugues Déprés, Guillaume Fertin et Eric Monfroy [1]. Ils ont notament démontré le résultat suivant

Théorème 1 *Soit G un graphe, soit l la longueur du plus grand cycle dans une base de cycles de G. Alors on a la minoration $B_c(G) \geq \min\{B(G), \lceil \frac{n}{l} \rceil\}$. En particulier, si $B(G) \leq \lceil \frac{n}{l} \rceil$ alors $B(G) = B_c(G)$. Où $B(G)$ est défini par $B(G) = \min_{etiq}\{\max_{\{u,v\} \in E}\{|etiq(u) - etiq(v)|\}\}$*

Ce théorème est la raison pour laquelle il est important que le plus long cycle d'une base de l'espace des cycles soit le plus court possible. En effet, supposons qu'un graphe G est tel que $|V| = 100$ et supposons également l'existence de deux bases $\mathcal{B}_1$, $\mathcal{B}_2$ de l'espace des cycles de G telles que, le plus long cycle de $\mathcal{B}_1$ soit de taille 50 et le plus long de $\mathcal{B}_2$ soit de taille 4. Alors, on a $B(G) \leq \lceil \frac{100}{50} \rceil = 2$ pour la base $\mathcal{B}_1$ et $B(G) \leq \lceil \frac{100}{4} \rceil = 25$ pour $\mathcal{B}_2$. Il est donc plus intéressant de maximiser la valeur de $\lceil \frac{n}{l} \rceil$.

2 Définitions

2.1 Définitions de base

Commençons simplement par donner quelques définitions élémentaires concernant les cycles.

Définition 1 (cycle) *Dans un graphe, un cycle C est une suite d'arêtes consécutives dont les deux sommets extrémités sont identiques.*
De plus, on peut décomposer C en $C^+ \cup C^-$ où C^+ (respectivement C^-) représente l'ensemble des arcs de C parcourus dans le sens positif (respectivement négatif).

Par la suite, le cycle $\{\{e_1, e_2\}, \{e_2, e_3\}, \ldots, \{e_n, e_1\}\}$. sera abrégé en $\{e_1, e_2, \ldots, e_n\}$

Définition 2 (cycle élémentaire) *Un cycle est dit élémentaire si il ne passe pas deux fois par le même sommet*

Définition 3 (Longueur d'un cycle) *La longueur d'un cycle C d'un graphe G est le nombre d'arêtes qui composent C*

Définition 4 (Poids d'un cycle) *Dans le cas d'un graphe valué, le poids d'un cycle C est donné par*

$$p(C) = \sum_{e \in C} w_e$$

Si le graphe n'est pas valué, alors le poids d'un cycle est simplement sa longueur.

Définition 5 (cycle isométrique) *On dit qu'un cycle est isométrique si la distance entre deux sommets dans le cycle est égale à leur distance dans le graphe.*

2.2 Définitions algébriques

Dans cette partie et dans les suivantes, on suppose que $\mathbb{K} = \mathbb{Q}$, $\mathbb{R}$ ou $\mathbb{C}$.

Maintenant que ces définitions classiques sur les cycles ont été posées, nous allons pouvoir nous interroger sur notre principal sujet d'étude, à savoir les bases de l'espace des cycles. Pour commencer, rappelons quelques définitions élémentaires d'algèbre linéaire. Les premières définitions concernent les notions de famille et de base.

Définition 6 (Famille) *Soit E un ensemble, un famille $\mathcal{F}$ de E est une suite d'éléments de E indexée par les entiers naturels.*

Définition 7 (Famille libre) *Soit E un espace vectoriel sur un corps $\mathbb{K}$, soit $\mathcal{F} = (u_1, \ldots, u_n)$ une famille d'éléments de E et soit $\lambda_1, \ldots, \lambda_n \in \mathbb{K}$. Alors on dit que $\mathcal{F}$ est libre si*

$$\sum_{i=1}^{n} \lambda_i u_i = 0_E \Longrightarrow \lambda_1 = \ldots = \lambda_n = 0_{\mathbb{K}}$$

Remarque 1 *0_E représente l'élément nul de l'ensemble E, par exemple $0_{\mathbb{R}^3}$ est le vecteur $(0, 0, 0)$.*

Définition 8 (Famille génératrice) *Soit E un espace vectoriel sur un corps $\mathbb{K}$, soit $\mathcal{F} = (u_1, \ldots, u_n)$ une famille d'éléments de E. Alors $\mathcal{F}$ est dite génératrice si*

$$\forall x \in E, \exists \lambda_1, \ldots, \lambda_n \in \mathbb{K}, x = \sum_{i=1}^{n} \lambda_i u_i$$

Nous pouvons désormais obtenir la définition d'une base d'un espace vectoriel

Définition 9 *Soit E un espace vectoriel sur un corps $\mathbb{K}$, une famille $\mathcal{F} = (u_1, \ldots, u_n)$ est une base de E si elle est libre et génératrice*

Nous allons également avoir besoin du théorème suivant

Théorème 2 (théorème de la base incomplète) *Soit un espace vectoriel E sur un corps $\mathbb{K}$,*

- *Toute famille libre de vecteurs peut être complétée en une base de E*
- *De toute famille génératrice de E, on peut extraire une sous-famille libre et génératrice.*

En particulier, ce théorème nous affirme qu'il est toujours possible d'extraire une base d'un espace vectoriel E.

2.3 Application aux Graphes

Dans cette partie nous allons voir comment il est possible d'utiliser les outils de l'algèbre linéaire sur des graphes. En effet, un graphe admet plusieurs représentations sous forme matricielle. Or l'ensemble des matrices, noté $\mathcal{M}_{n,m}(\mathbb{K})$ forme un espace vectoriel. Il devient donc possible d'appliquer les définitions et les théorèmes vu dans la partie précédente.

Commençons premièrement par définir la représentation vectorielle d'un cycle.

Définition 10 (vecteur représentatif d'un cycle) *Soit $G = (V, E)$ un graphe non orienté avec $E = \{e_1, \ldots, e_n\}$ On se donne un cycle C de G, le vecteur représentatif de C, noté*

$$\mu_i(C) = \begin{cases} 1 & si \quad e_i \in C \\ 0 & sinon \end{cases}$$

le vecteur représentatif de C est alors $\mu(C) = (\mu_1, \ldots, \mu_m) \in \mathbb{R}^m$

Remarque 2 *Dans le cadre des graphes non orientés, on remarque que le vecteur d'un cycle est unique et ne change pas après une ou plusieurs permutation(s) des éléments. En effet, le sens n'ayant pas d'importance, le cycle $\{a, b, c\}$ est identique au cycle $\{b, c, a\}$ par exemple.*

Nous pouvons donc définir l'espace vectoriel des cycles d'un graphe

Définition 11 (Espace des cycles) *Soit un graphe $G = (V, E)$. L'espace des cycles de G est noté $\mathcal{C}_G$ et est défini par*

$$\mathcal{C}_G = \{\mu(C) | C \text{ élémentaire de } G\}$$

Toutes ces définitions nous permettent d'énoncer le théorème suivant, qui nous permet de construire un algorithme simple de calcul d'une base de cycles.

Théorème 3 (Calcul d'une base de cycles) *Soit $G = (V, E)$ un graphe, soit $T = (V, E')$ un arbre couvrant de G, $\forall e \in E \setminus E'$ on pose C_e l'unique cycle du graphe $(V, E' \cup \{e\})$. Alors on a les propriétés suivantes :*

1. *la famille $\mathcal{F} = (\mu_e(C_e))_{e \in E \setminus E'}$ forme un base de $\mathcal{C}_G$.*
2. *$Dim(\mathcal{F}) = E - V + k$, k est le nombre de composantes connexes*

Remarque 3 *Une base de l'espace des cycles est dite minimale si les cycles qui la compose sont de poids minimum. Elle est souvent notée MBC(G) (minimum cycle basis)*

2.4 Les caractéristiques des graphes

Les définitions suivantes ne sont pas utiles pour la compréhension du sujet, cependant, elles permettent de définir certaines caractéristiques des graphes et peuvent permettre d'analyser les algorithmes.

Définition 12 (Distance entre deux sommets) *La distance entre deux sommets d'un graphe est la longueur d'un plus court chemin entre ces deux sommets.*

Définition 13 (Rayon) *Le rayon d'un graphe est la plus petite distance à laquelle puisse se trouver un sommet de tous les autres.*

Définition 14 (Diamètre) *Le diamètre d'un graphe est la plus grande distance possible entre deux sommets du graphe.*

Définition 15 (Densité) *Soit $G = (V, E)$ un graphe non orienté, la densité du graphe est définie par*

$$D = \frac{2|E|}{|V|(|V| - 1)}$$

On dit que le graphe est dense si il a beaucoup d'arêtes et creux sinon.

Remarque 4 *Si le graphe n'a pas d'arêtes, alors $D = 0$, et si le graphe est complet alors $D = 1$.*

Définition 16 (Sous-graphe commun) *Un sous-graphe commun de deux graphes G et $G^{'}$ est un graphe isomorphe à des sous-graphes de G et $G^{'}$.*

Définition 17 (Plus grand graphe commun) *Soient G_1 et G_2 des graphes, un plus grand sous-graphe partiel commun est un sous-graphe partiel commun avec un nombre d'arêtes maximal. on le note $MCPS(G_1, G_2)$.*

Définition 18 (Similarité) *Soient G_1 et G_2 des graphes. On note $sim(G_1, G_2)$ la similarité entre ces deux graphes. La similarité est définie par*

$$sim(G_1, G_2) = \frac{|MCPS(G_1, G_2)|}{\max\{|G_1|, |G_2|\}}$$

Où $|G|$ désigne le nombre d'arêtes de G.
De plus, elle admet les bornes suivantes $0 \leq sim(G_1, G_2) \leq 1$

Cette définition de la similarité, dite similarité au sens du $MCPS$ a été proposée par Horst Bunke et Kim Shearer en 1998 [2]. Leur article nous offre aussi la définition et de théorème suivant

Définition 19 (distance entre deux graphes) *La distance entre deux graphes non vides G_1 et G_2 est définie par*

$$d\left(G_1, G_2\right) = 1 - sim\left(G_1, G_2\right)$$

Théorème 4 (De Bunke et Shearer) *Pour tous graphes G_1, G_2 et G_3, on a les propriétés suivantes :*

1. $0 \leq d(G_1, G_2) \leq 1$
2. $d(G_1, G_2) = 0 \Longleftrightarrow G_1$ *est isomorphe à* G_2
3. $d(G_1, G_2) = d(G_2, G_1)$
4. $d(G_1, G_3) \leq d(G_1, G_2) + d(G_2, G_3)$

Autrement dit, c'est bien une distance au sens mathématique du terme. L'espace des graphes muni de cette distance est donc un espace métrique [2].

Remarque 5 *Les définitions 16 à 19 ainsi que le théorème 4 ne seront pas utilisés dans le reste du rapport. On a quand même décidé de les garder car ces outils peuvent être utilisés pour des analyses mathématiques. Par exemple, il aurait pu être intéressant de regarder le comportement des différents algorithmes quand la distance entre le graphe $G = (V, E)$ et K_V tend vers 0 afin d'obtenir un seuil de densité, à partir duquel, tel algorithme est meilleur qu'un autre.*

3 Algorithmes

Maintenant que les bases théoriques ont été posées, nous pouvons nous intéresser aux principaux algorithmes. En particulier, nous allons détailler le fonctionnement de l'algorithme de Horton et montrer deux cas d'optimalité à l'algorithme construit à partir du théorème 3. Nous donnerons également la complexité de tous les algorithmes de que nous présenterons.
Même si certains ne seront pas testés, nous avons trouvé intéressant d'en citer quelques-uns supplémentaires.

3.1 Les algorithmes de recherche d'une base

3.1.1 Algorithme naïf

L'algorithme naïf est l'algorithme construit à partir du théorème 3. Il a l'avantage de fonctionner sur les graphes orientés et d'être très facile à implémenter.

On suppose l'existence d'une fonction $SpanningTree(G)$ retournant un arbre couvrant de G. L'algorithme peut alors s'écrire

Algorithm 1: `Naive_Cycle_Basis`

Data: G = (V,E) un graphe
Result: une base de l'espace des cycles de G
$\mathcal{B} \leftarrow \emptyset$;
$A \leftarrow SpanningTree(G)$;
$W \leftarrow G \setminus A$;
for $e \in W$ **do**
  $\mathcal{B} \leftarrow \mathcal{B} \cup FindCycle(A \cup \{e\})$;
  $A \leftarrow A \setminus \{e\}$;
end
return $\mathcal{B}$

Complexité

Notons $T(A)$ la complexité algorithmique de la fonction $SpanningTree(G)$, et notons $T(W)$ la complexité du calcul de $W = G \setminus A$.
Analysons maintenant la complexité des instructions dans la boucle for :
La complexité de la boucle for est donnée par $\sum_{i=1}^{W} \mathcal{O}(FindCycle(A \cup \{e\}))$.

La recherche d'un cycle dans le graphe $A \cup \{e\}$ est effectuée à l'aide du parcours en profondeur (DFS) qui a une complexité en $\mathcal{O}(E+V)$. La complexité des instruction dans la boucle for est donc en $\mathcal{O}(E+V)$ et donc on a

$$\mathcal{O}(for) = \sum_{i=1}^{W} E + V = W(E+V) \leq \sum_{i=1}^{E} E + V = E^2 + EV \qquad (1)$$

La complexité totale de l'algorithme est donc en $\mathcal{O}\left(T(A) + T(W) + E^2 + EV\right)$

Remarque 6 *La complexité de la fonction $T(A)$ n'est pas donnée explicitement, car suivant l'arbre couvrant que l'on souhaite, celle-ci peut varier. Il en va de même pour la fonction $T(W)$, la complexité peut varier en fonction de la représentation choisie pour le graphe.*

Optimalité

En regardant l'algorithme, on se rend compte qu'en fonction de l'arbre couvrant choisi, la base calculée est "plus ou moins" minimale. En particulier, la base construite est minimale si le graphe $G = K_n$ ou tout autres graphes admettant $K_{1,n}$ en arbre couvrant.
Pour démontrer ce résultat, nous avons besoin de la définition suivante

Définition 20 *Le graphe $K_{1,n}$ est défini*

$$K_{1,n} = \left(V, \left\{ \bigcup_{k=1}^{n} \{0, k\} \right\}\right)$$

Nous avons ensuite besoin des lemmes suivants

Lemme 5 *Soit K_n le graphe complet à n sommets, alors il existe un arbre couvrant A de K_n tel que A est isomorphe à $K_{1,n-1}$*

Preuve 1 *Soit $K_n = (V, E)$ le graphe complet à n sommets. On choisit un sommet initial s_0. Par définition des graphes complets, on a que $\forall s \in V \setminus \{s_0\}$, $d(s_0, s) = 1$ et $\forall s \in V \setminus \{s_0\}, \{s_0, s\} \in E$. Il est donc possible de construire l'arbre $A = (V, \{\bigcup_{k=1}^{n} \{s_0, k\}\})$, qui est isomorphe à $K_{1,n-1}$.*

Lemme 6 *Soit K_n le graphe complet à n sommets, soit $\Gamma = K_n \setminus K_{1,n-1}$, alors $\forall e \in E_\Gamma$, l'unique cycle de $K_{1,n-1} \cup \{e\}$ est de taille 3.*

Preuve 2 *On démontre ce résultat par récurrence sur n*
Initialisation : $k = 3$, par le lemme 3, $K_{1,2} \subseteq K_3$, il est alors évident que l'ajout de la seule arête de $\Gamma = K_3 \setminus K_{1,2}$ à $K_{1,2}$ forme un cycle de taille 3.
On suppose que la propriété est vraie au rang $k = n$, et montrons qu'elle est vraie au rang $k = n + 1$.
On a $K_{n+1} = K_n \cup K_{1,n}$, par hypothèse de récurrence, la propriété est vraie sur K_n donc elle est également vraie sur $K_{1,n-1}$. Or $K_{1,n} = K_{1,n-1} \cup \{s_0, s_n\}$. Donc $\forall s^{'} \in K_{1,n-1}, \forall e \in K_{n+1} \setminus K_{1,n}$, le cycle formé est alors $\{s_0, s_n, s^{'}, e\} = \{s_0, s_n, s_j\}$, $j \in \{1, \ldots, n-1\}$ qui est de taille 3.

La propriété est donc vraie au rang $n + 1$, elle est donc vraie $\forall n \geq 3$

Nous pouvons donc énoncer le théorème suivant

Théorème 7 *$\forall n \geq 3$, la base construite par le le théorème 3 sur le graphe complet K_n est minimale.*

Preuve 3 *Soit K_n, par le lemme 3, l'arbre couvrant construit est $K_{1,n-1}$, par le lemme 4, tous les cycles construits sont de taille 3. 3 étant la taille minimale d'un cycle dans un graphe non-orienté, la base est minimale.*

Ceci démontre que l'algorithme est optimal sur les graphes complets. Ce résultat n'est pas anodin, en effet, le temps de calcul est très réduit (voir benchmark) pour ce genre de graphe comparé aux autres algorithmes de calcul de bases minimales testés.

Un raisonnement analogue permet d'étendre ce résultat à tous les graphes qui admettent une instance de $K_{1,n}$ en arbre couvrant. C'est une optimisation qui peut être utile sur des petits graphes. Cependant au delà d'une certaine densité et d'un nombre de sommets élevé, la recherche d'un arbre couvrant de distance minimale est longue. Il est également important de noter que la liste n'est pas exhaustive, il est possible de trouver plusieurs arbres couvrants d'un graphe qui permettent la construction d'une base minimale. Cependant le temps de calcul est trop élevé comparé aux autres algorithmes.

3.1.2 Algorithme de Paton

Il s'agit d'un algorithme de calcul de base de l'espace des cycles pour les graphes non orientés. Il a été trouvé par Keith Paton en 1969. Nous en parlons car il est disponible dans la fonction cycle_basis de la librairie Python NetworkX [3]. Comme il ne calcule pas une base minimale, il n'a pas été testé en détail. Cependant, sa complexité lui donne un temps d'exécution rapide, qui est en moyenne de 10 secondes pour traiter les 194 instances de Harwell-Boeing.

Remarque 7 *La moyenne a été calculée sur 100 exécutions du programme calculant une base des 194 instances.*

Complexité

La complexité de l'algorithme de Paton est intéressante. En effet, celle-ci ne dépend que du nombre de sommets du graphe, plus précisément, elle est en $\mathcal{O}(V^2)$ [4].

3.2 Les algorithmes de recherche d'une base minimale

Dans cette section, nous donnons les deux principaux algorithmes de calcul d'une base minimale dans un graphe non orienté. Historiquement, les deux algorithmes présentés ici sont les premiers algorithmes avec une complexité polynomiale pour ce problème. Ils sont aujourd'hui à la base de tous les meilleurs algorithmes connus (voir Algorithme d'Amaldi).

3.2.1 Algorithme de Horton

Il s'agit du premier algorithme avec une complexité polynomiale pour calculer une base de cycles. Il a été trouvé en 1987 par Joseph Douglas Horton
Le principe de l'algorithme est le suivant :

1. Trouver le plus court chemin $P(x, y)$ entre chaque paires de sommets
2. Pour chaque sommets v et arêtes (x,y) dans le graphe, créer le cycle $C(v, x, y) = P(v, x) + P(v, y) + (x, y)$
3. Trier les cycles par poids
4. Utiliser un algorithme glouton pour trouver la base des cycles depuis cet ensemble de cycles

Algorithme glouton

Dans cette partie, on notera e_1 le premier élément de E

Les trois premières étapes de l'algorithme sont assez simples, néanmoins, la dernière étape mérite quelques explications supplémentaires. Nous donnons ici une solution à ce problème.
Cet algorithme glouton repose sur un test de liberté de la base à laquelle on ajoute le plus petit cycle présent dans l'ensemble des cycles [5]. Plus précisément, son fonctionnement repose sur le théorème de la base incomplète, ainsi que sur la proposition suivante

Proposition 1 *Soit E un espace vectoriel, soit $\mathcal{F}$ une famille d'éléments de E, alors $\mathcal{F}$ est libre si $rg(\mathcal{F}) = |\mathcal{F}|$.*

Comme on a $rg(A) = rg(A^t)$, il suffit de créer la matrice M telle que

$$M = \begin{bmatrix} \mu(B_{1)} \\ \vdots \\ \mu(B_j) \\ \mu(e_1) \end{bmatrix} \tag{2}$$

et d'appliquer la proposition 1 pour obtenir son rang.

On peut ainsi écrire l'algorithme glouton comme ceci

Algorithm 2: `Algorithme Glouton`

Data: E l'ensemble construit à l'étape 3
Result: une base de l'espace des cycles de G
$B \leftarrow \{e_1\}$;
$E \leftarrow E \setminus \{e_1\}$;
while $|B| < |E| - |V| + 1$ **do**
  if $rg(B \cup \{e_1\}) = |B| + 1$ **then**
    $B \leftarrow B \cup \{e_1\}$;
  $E \leftarrow E \setminus \{e_1\}$
end
return B

Horton a démontré que l'ensemble construit à l'étape de 3, appelé Ensemble de Horton, souvent noté $\mathcal{H}$, contient une base de l'espace des cycles. En utilisant le théorème de la base incomplète, on sait que l'on peut compléter l'ensemble $\mathcal{B} = \emptyset$ en une base de l'espace des cycles en lui ajoutant des

éléments de $\mathcal{H}$. Tout ceci permet d'affirmer que cet algorithme se termine et calcul bien une base. La minimalité est assurée à l'aide trois premières étapes et au choix du premier élément de $\mathcal{H}$ à chaque tour de boucle.

Remarque 8 *Il est important d'effectuer le test d'indépendance linéaire sur* $\mathbb{Z}/2\mathbb{Z}$ *et non sur* $\mathbb{R}$.

Complexité

Horton a démontré que son algorithme donne une base minimale de l'espace des cycles en $\mathcal{O}\left(E^3V\right)$ opérations [6]. Il s'agit par ailleurs de la complexité de l'étape 4.

3.2.2 Algorithme de De Pina

Cet algorithme a été trouvé en 1995 par José Coelho de Pina durant sa thèse à l'Université d'Amsterdam [7]. Son approche est différente de celle de Horton et consiste en l'application d'une fonction recherchant un cycle de longueur impaire dans le graphe avec un poids minimum.

De Pina nous fourni le pseudo-code suivant à la page 94 de sa thèse

Algorithm 3: Algorithme de De Pina

Data: G = (V,E) un graphe
Result: une base de l'espace des cycles de G
$T \leftarrow SpanningTree(G)$;
$E \leftarrow G \setminus T = \{e_1, \ldots, e_\nu\}$;
$\mathcal{S}_{1,i} \leftarrow \{e_i\}_{1 \leq i \leq \nu}$;
for $k \leftarrow 1, \ldots, \nu$ **do**
  Trouver le plus petit cycle C_k de poids minimal avec un nombre impair d'arêtes dans $\mathcal{S}_{k,k}$;
  for $i \leftarrow k+1, \ldots, \nu$ **do**
    $$\mathcal{S}_{k+1,i} = \begin{cases} S_{k,i} \Delta S_{k,k} & \text{si } C_k \text{ est un cycle impair dans } S_{k,i} \\ \mathcal{S}_{k,i} & \text{sinon} \end{cases}$$
  end
end
return $\{C_1, \ldots, C_n\}$

Remarque 9 *l'opération* $A\Delta B$ *désigne la différence symétrique entre deux ensembles, elle est définie par*

$$A\Delta B = (A \cup B) \setminus (A \cap B)$$

Version algébrique

Dans sa formulation originale, l'algorithme de De Pina peut être assez obscur. Notamment le fait que le cycle recherché doit être de longueur impaire. Pour comprendre cet algorithme, il est nécessaire de passer par une version algébrique que voici

Algorithm 4: Algorithme de De Pina version algébrique

Data: G = (V,E) un graphe
Result: une base de l'espace des cycles de G
$S = \{S_1, \ldots, S_\nu\}$ la base canonique;
for $i = 1$ *à* ν **do**
  Trouver le plus petit cycle C_i de G tel que $\langle C_i, S_i \rangle = 1$;
  for $k = i + 1$ *à* ν **do**
    if $\langle S_k, Ci \rangle = 1$ **then**
      $S_k = \langle S_k, S_i \rangle$
  end
end

Explications

On sait que le produit scalaire entre deux vecteurs de E^n est défini par

$$\langle u, v \rangle = \sum_{i=1}^{n} u_i v_i$$

Or, l'espace des cycles d'un graphe non orienté est un $GF(2)-$espace vectoriel et donc doit respecter la propriété $\forall u, v \in GF(2)^n$, $u + v \in GF(2)^n$. L'addition doit alors être définie comme un xor, c'est-à-dire

$$\begin{aligned} \oplus : GF(2)^n \times GF(2)^n &\to GF(2)^n \\ (a, b) &\mapsto (a + b)[2] \end{aligned}$$

Nous pouvons donc définir le produit scalaire sur $GF(2)^n$ comme ceci

$$\forall u, v \in GF(2)^n, \langle u, v \rangle = \sum_{i=1}^{n} u_i v_i [2] = \bigoplus_{i=1}^{n} u_i v_i$$

Cette définition du produit scalaire implique que si $\langle u, v \rangle = 1$, alors il existe un nombre impair d'indices j tel que $u_j v_j = 1$. Ce qui n'est possible que si $|u|_1 \geq 3$, $|v|_1 \geq 3$ et $|u|_1 = |v|_1 \equiv 1 \pmod 2$, donc qui n'est possible que si le cycle est impair. Pour finir, l'orthogonalité implique l'indépendance linéaire

et la recheche du plus petit cycle implique la minimalité. Nous pouvons conclure que l'algorithme construit bien une base minimale de l'espace des cycles.

Complexité

La compléxité de l'algorithme de De Pina est en $\mathcal{O}\left(E^3 + EV^2\log(V)\right)$ [8]. Cependant, il a été montré que la complexité pouvait descendre en $\mathcal{O}\left(E^2 + EV^2\log(V)\right)$ avec un bon algorithme de multiplication matricielle [9].

3.2.3 Algorithme d'Amaldi

L'algorithme d'Amaldi est l'algorithme ayant la complexité la plus basse connue pour la recherche de base minimale de l'espace des cycles. Il s'agit d'une modification de celui de De Pina, en particulier, il utilise un algorithme de type Monte Carlo pour calculer un cycle isométrique minimal C_i tel que $\langle C_i, S_i\rangle \neq 0$ [10].

Complexité

Amaldi a démontré que son algorithme a une complexité en $\mathcal{O}\left(E^\omega\right)$, ω est une constante quelconque et indique que son algorithme a une chance de se tromper d'au plus 1/2. [11] [10] [12].

Remarque 10 *Amaldi indique que ω est la puissance de l'algorithme de multiplication matricielle. L'un des meilleurs algorithme connu a une complexité en $\mathcal{O}(n^{2.37286})$ [13]. L'algorithme d'Amaldi a une complexité qui respecte*

$$\mathcal{O}(E^2) \leq \mathcal{O}(E^\omega) \leq \mathcal{O}(E^{2,37286})$$

.

4 Benchmark et comparaison

Tous les programmes et algorithmes ont été écrit en Python à l'aide de la libraire Networkx [14]. Il s'agit d'une libraire écrite en Python disposant d'un large éventail d'algorithmes sur les graphes.
Nous avons utilisé au maximum cette dernière afin de rendre nos algorithmes parfaitement compatibles avec leur classe Graph et le reste de la librairie.
Tous les tests ont été effectués sur un ordinateur disposant d'un processeur Intel i5-6400 disposant de 4 coeurs cadencés de 2,70 à 3,30 GHz et de 20Go de mémoire vive DDR4 candencée à 2133MHz tournant sous Windows 10.

Pour les benchmarks, nous avons utilisé un pack de 113 graphes issus des instances de Harwell-Boeing. Pour des raisons de temps et d'éviter un blocage d'exécution, on a premièrement sélectionné les graphes sur lesquels les algorithmes terminaient en moins de 10 minutes. Cependant, il est à noté que sur certains graphes, seul l'algorithme de Horton ne terminait pas. Les premiers test ont donc été effectués sur les 50 instances que Horton a pu terminé. Nous avons toutefois lancé un test indépendant sur 109 des 113 instances sans timeout, à l'aide de l'algorithme de De Pina fournit dans la libraire NetworkX de Python. Nous analyserons ces résultats dans la partie associée à l'algorithme.

4.1 Algorithme de Horton

Voici quelques statistiques obtenus sur 50 sur 113 instances.
— le temps total est de 20140 secondes, soit environs 5h35
— moyenne de 419 secondes par graphes, soit environs 7 minutes
— mediane à environs 2 minutes
— ecart-type de 604 secondes, soit environs 10 minutes

4.2 Algorithme de De Pina

La version rapide de l'algorithme de De Pina est implémentée dans la fonction `minimum_cycle_basis` de la librairie Python NetworkX [15].

Résultats sur 50 instances

Voici quelques statistiques obtenus sur 50 sur 113 instances.
— le temps total est de 7654 secondes, soit environs 2h07
— moyenne de 156 secondes par graphes, soit environs 2m36
— mediane à 92 secondes, soit environs 1m32

— ecart-type de 198 secondes, soit environs 3m18

Résultats sur 109 instances

Voici quelques statistiques obtenus sur 109 sur 113 instances.
— le temps total est de 466153 secondes, ce qui représente 5 jours, 9 heures et 39 minutes
— moyenne de 4356 secondes par graphes, soit environs 1h12
— mediane à 1245 secondes, soit environs 20 minutes
— ecart-type de 8421 secondes, soit environs 2h20

La figure 3 compile le temps de calcul en secondes et le nombre d'arêtes des graphes. Les graphes y sont triés à la fois sur le nom et sur le nombre d'arêtes. Ceci permet d'effectuer des comparaisons entre les différentes familles d'instances. Il devient alors possible d'expliquer certains temps de calcul. En effet, on s'attend à ce que le temps le plus grand arrive sur le dernier graphe de la famille, ce qui n'est pas le cas pour deux d'entre eux, à savoir l'instance bcsstk19 et bp__1000. Ces derniers sont sûrement le résultat d'une instabilité de l'ordinateur.

Remarque 11 *Sur les 113 instances, trois des graphes possédaient un nombre d'arêtes important. En effet, 24h n'ont pas suffi pour terminer le calcul sur plus petit des trois. On a donc pris la décision de les enlever de la liste. Pour le quatrième graphe manquant, il s'agit d'une erreur de lecture du nom du fichier.*

4.3 Tableau récapitulatif

Le tableau ci-dessous est le récapitulatif des résultats obtenus avec l'algorithme de NetworkX sur les 109 instances. On y trouve le nom de l'instance, le nombre de sommets, d'arêtes, la densité ainsi que le temps de calcul et la longueur du plus long cycle de la base calculée.

file	nodes	edges	density	l	time
494_bus.mtx.rnd	494	586	0.0048123116341329	19	110.71381688118
662_bus.mtx.rnd	662	906	0.0041409381555914	23	527.46754360199
685_bus.mtx.rnd	685	1282	0.0054723182652495	22	1485.4226880074
arc130.mtx.rnd	130	715	0.085271317829457	4	53.188780784607
ash292.mtx.rnd	292	958	0.022548604246105	3	330.50416231155
bcspwr01.mtx.rnd	39	46	0.062078272604588	13	0.067818880081177

file	nodes	edges	density	l	time
bcspwr02.mtx.rnd	49	59	0.050170068027211	8	0.14164876937866
bcspwr03.mtx.rnd	118	179	0.025930754744314	10	4.4311490058899
bcspwr04.mtx.rnd	274	669	0.017887222266784	10	158.92124223709
bcspwr05.mtx.rnd	443	590	0.0060263730427055	14	144.09071969986
bcsstk01.mtx.rnd	48	176	0.15602836879433	4	1.6216638088226
bcsstk04.mtx.rnd	132	1758	0.20333102012491	4	209.51163196564
bcsstk05.mtx.rnd	153	1135	0.097609219126247	4	195.61720395088
bcsstk06.mtx.rnd	420	3720	0.042277531537675	4	4047.3594536781
bcsstk19.mtx.rnd	817	3018	0.0090539275685795	11	12525.488302469
bcsstk20.mtx.rnd	467	1295	0.011901370265874	58	1613.8319373131
bcsstk22.mtx.rnd	110	254	0.042368640533778	22	12.305026292801
bcsstm07.mtx.rnd	420	3416	0.038822593476531	4	4844.1784944534
bp_____0.mtx.rnd	822	3260	0.009661234444968	7	11312.316076756
bp___200.mtx.rnd	822	3788	0.011225998796791	6	11062.260090828
bp___400.mtx.rnd	822	4015	0.011898728925321	7	12091.544835329
bp___600.mtx.rnd	822	4157	0.012319555701758	6	15931.007796764
bp___800.mtx.rnd	822	4518	0.01338940405594	6	16929.411557436
bp__1000.mtx.rnd	822	4635	0.013736141611174	6	22045.535763741
bp__1200.mtx.rnd	822	4698	0.013922846448607	6	18032.748785734
bp__1400.mtx.rnd	822	4760	0.014106587717193	6	16367.775210381
bp__1600.mtx.rnd	822	4809	0.014251802590752	7	19907.798105717
can__144.mtx.rnd	144	576	0.055944055944056	24	68.87552022934
can__161.mtx.rnd	161	608	0.047204968944099	3	87.761899232864
can__292.mtx.rnd	292	1124	0.02645577366662	5	412.41434216499
can__445.mtx.rnd	445	1682	0.017026014778824	10	1434.6449372768
can__715.mtx.rnd	715	2975	0.011655011655012	16	9170.6192395687
can__838.mtx.rnd	838	4586	0.013076591874036	15	17443.090572119
curtis54.mtx.rnd	54	124	0.086652690426275	9	1.4566614627838
dwt__209.mtx.rnd	209	767	0.035287081339713	8	160.03981852531
dwt__221.mtx.rnd	221	704	0.0289592760181	12	159.06713747978
dwt__234.mtx.rnd	117	162	0.023872679045093	8	3.9515764713287
dwt__245.mtx.rnd	245	608	0.0203412512546	11	140.67086219788
dwt__310.mtx.rnd	310	1069	0.022319657584299	3	504.68915605545
dwt__361.mtx.rnd	361	1296	0.01994459833795	3	840.44473791122
dwt__419.mtx.rnd	419	1572	0.017951148211166	11	1212.2153713703
dwt__503.mtx.rnd	503	2762	0.021876707880209	8	3505.3606228828
dwt__592.mtx.rnd	592	2256	0.01289614487584	3	3418.3382258415

file	nodes	edges	density	l	time
dwt__878.mtx.rnd	878	3285	0.0085324010462256	3	10879.050904512
dwt__918.mtx.rnd	918	3233	0.007681104672573	20	11322.70097065
dwt__992.mtx.rnd	992	7876	0.016023241430943	3	55408.878312349
fs_183_1.mtx.rnd	183	701	0.042094517504354	4	109.11057209969
fs_541_1.mtx.rnd	541	2466	0.016882316697474	3	2130.1699185371
fs_680_1.mtx.rnd	680	1464	0.0063415056744347	5	2181.6548907757
fs_760_1.mtx.rnd	760	3518	0.01219748977186	6	9737.436722517
gent113.mtx.rnd	104	549	0.10250186706497	6	27.119455099106
gr_30_30.mtx.rnd	900	3422	0.0084587813620072	3	12041.427155972
gre__115.mtx.rnd	115	267	0.040732265446224	4	10.074062824249
gre__185.mtx.rnd	185	650	0.038190364277321	4	94.189111948013
gre__343.mtx.rnd	343	1092	0.018617973505192	4	480.49771738052
gre__512.mtx.rnd	512	1680	0.012842465753425	4	1660.1477262974
gre_216a.mtx.rnd	216	660	0.028423772609819	4	111.44671344757
hor__131.mtx.rnd	434	2138	0.02275412139079	3	1915.7568175793
ibm32.mtx.rnd	32	90	0.18145161290323	5	0.3510890007019
impcol_a.mtx.rnd	206	557	0.026379351172152	9	80.642760753632
impcol_b.mtx.rnd	59	281	0.16423144360023	6	4.4291689395905
impcol_c.mtx.rnd	137	352	0.037784456848433	6	20.842258691788
impcol_d.mtx.rnd	425	1267	0.014062153163152	21	841.97515511513
impcol_e.mtx.rnd	225	1187	0.047103174603175	6	285.32083678246
jagmesh1.mtx.rnd	936	2664	0.0060880296174414	48	10479.945447683
jpwh_991.mtx.rnd	983	2678	0.0055484996467441	10	13782.495739222
lns__131.mtx.rnd	123	275	0.036652005864321	4	12.425818920135
lns__511.mtx.rnd	503	1425	0.011286860510245	4	1580.6417756081
lund_a.mtx.rnd	147	1151	0.107259342093	3	141.9346909523
lund_b.mtx.rnd	147	1147	0.10688659025254	3	141.17763710022
mcca.mtx.rnd	168	1662	0.11847733105218	3	285.53162693977
nnc261.mtx.rnd	261	794	0.023401119952844	6	259.45973467827
nnc666.mtx.rnd	666	2148	0.0096999254893992	8	5063.1830050945
nos1.mtx.rnd	158	312	0.025155204386036	4	20.65675163269
nos2.mtx.rnd	638	1272	0.0062597501021146	4	1630.062128067
nos3.mtx.rnd	960	7442	0.016167014250956	4	44236.462475777
nos4.mtx.rnd	100	247	0.04989898989899	4	8.0215501785278
nos5.mtx.rnd	468	2352	0.021523087904244	5	2850.9462924004
nos6.mtx.rnd	675	1290	0.0056709528519618	4	1434.5014977455
nos7.mtx.rnd	729	1944	0.0073260073260073	4	3428.3116652966

file	nodes	edges	density	l	time
orsirr_2.mtx.rnd	886	2542	0.0064837841629363	4	7267.4387631416
plat362.mtx.rnd	362	2712	0.041505333557797	3	2050.0563349724
plskz362.mtx.rnd	362	880	0.013467807349138	4	376.95552611351
pores_1.mtx.rnd	30	103	0.2367816091954	4	0.42184495925903
pores_3.mtx.rnd	456	1769	0.017052245999614	4	1627.001875639
saylr1.mtx.rnd	238	445	0.015778463284048	4	60.653790473938
saylr3.mtx.rnd	681	1373	0.0059298609311566	12	1633.0095584393
sherman1.mtx.rnd	681	1373	0.0059298609311566	12	1632.5958662033
sherman4.mtx.rnd	546	1341	0.0090130053432806	4	1245.8086948395
shl____0.mtx.rnd	663	1682	0.0076645113076604	7	2216.3135340214
shl__200.mtx.rnd	663	1720	0.0078376691136599	6	2301.2343840599
shl__400.mtx.rnd	663	1709	0.0077875444856074	6	2292.5605986118
steam1.mtx.rnd	240	1761	0.061401673640167	4	560.6246342659
steam2.mtx.rnd	600	6580	0.036616583194213	4	22259.799015045
steam3.mtx.rnd	80	424	0.13417721518987	3	13.669355154037
str____0.mtx.rnd	363	2446	0.037228132657565	7	1732.0520355701
str__200.mtx.rnd	363	3049	0.046405795777971	7	2363.7399950027
str__600.mtx.rnd	363	3244	0.049373696787057	6	2597.5334653854
west0132.mtx.rnd	132	404	0.046726810085589	5	24.725525379181
west0156.mtx.rnd	156	371	0.030686517783292	7	26.53409409523
west0167.mtx.rnd	167	489	0.03527883991054	6	46.540175676346
west0381.mtx.rnd	381	2150	0.029700234839066	5	1415.8222551346
west0479.mtx.rnd	479	1889	0.016500554677195	6	1776.4899435043
west0497.mtx.rnd	497	1715	0.013914129940936	7	1594.904753685
west0655.mtx.rnd	655	2841	0.013264234190069	7	5731.8415782452
will199.mtx.rnd	199	660	0.033500837520938	6	96.353856801987
will57.mtx.rnd	57	127	0.079573934837093	4	1.2536654472351

TABLE 1:

4.4 Comparaison et Analyse des résultats

L'objectif principal de ce stage était l'analyse et la comparaison de ces algorithmes. En particulier, nous devions comparer leurs temps d'exécution mais également de comparer la longueur du plus long cycle de la base calculée.

Comparaison du temps de calcul

Dans cette partie, aucun algorithme parallèle n'a été utilisé. De plus, il est à noter que Python n'a utilisé que 25-30% du processeur.

La figure 2 montre que l'algorithme de Horton que nous avons implémenté, hors rares cas, est beaucoup plus lent que l'algorithme de la libraire Networkx. Ce résultat est notamment dû à une opération. Il s'agit du calcul d'indépendance linéaire. En effet, pour ce faire nous utilisons le rang de la matrice. L'algorithme est une modification de la méthode du pivot de Gauss sur le corps algébrique $\mathbb{Z}/2\mathbb{Z}$ (également appelé $GF(2)$). Le calcul du rang d'une matrice s'en voit cependant nettement ralenti. Un algorithme plus adapté à la vérification d'indépendance linéaire sur $\mathbb{Z}/2\mathbb{Z}$ réduirait très probablement le temps de calcul. La méthode du pivot de Gauss étant parallélisable, il est également possible d'utiliser cette propriété pour augmenter l'efficacité du temps de calcul de l'algorithme de Horton.

Comparaison de la longueur du plus cycle de la base

La figure 1 montre une comparaison de la longueur du plus cycle de la base entre les deux algorithmes sur chacun des 50 graphes. Pour une application au problème de la bande passante cyclique, ce cycle doit être le plus petit possible. On remarque que les deux algorithmes donnent les mêmes résultats, ce qui est normal. Les deux algorithmes calculent des bases minimales, ils se doivent de donner le même plus long cycle.

Comparaison des deux algorithmes sur 50 instances

Le tableau ci-dessous compile les résultats des deux algorithmes. Les noms en rouges sont les instances où l'algorithme de Horton a été le plus rapide.

file	nodes	edges	radius	diameter	density	l_horton	time	l	time_de_pina
494_bus.mtx.rnd	494	586	14	26	0.004812312	19	36.22364759	19	168.6619124
arc130.mtx.rnd	130	715	2	4	0.085271318	4	791.1461713	4	88.60184097
ash292.mtx.rnd	292	958	13	25	0.022548604	3	336.3934734	3	620.5493205
ash85.mtx.rnd	85	219	7	13	0.061344538	3	3.282242298	3	8.934787512
bcspwr01.mtx.rnd	39	46	6	11	0.062078273	13	0.059858561	13	0.102279425
bcspwr02.mtx.rnd	49	59	6	9	0.050170068	8	0.096699953	8	0.217054605
bcspwr03.mtx.rnd	118	179	7	14	0.025930755	10	2.278880119	10	7.001795292
bcspwr04.mtx.rnd	274	669	10	20	0.017887222	10	587.6556215	10	270.4042008
bcspwr05.mtx.rnd	443	590	11	20	0.006026373	14	59.93980527	14	279.3072593
bcsstk01.mtx.rnd	48	176	3	4	0.156028369	4	3.486672878	4	2.75996685
bcsstk22.mtx.rnd	110	254	12	16	0.042368641	22	20.63489151	22	15.79751277
can__144.mtx.rnd	144	576	12	13	0.055944056	24	391.6403971	24	80.85421491
can__161.mtx.rnd	161	608	8	10	0.047204969	3	89.90764832	3	103.6169167
can__292.mtx.rnd	292	1124	5	10	0.026455774	5	2961.682968	5	606.3963048
curtis54.mtx.rnd	54	124	4	7	0.08665269	9	2.094423532	9	1.770426035
dwt__209.mtx.rnd	209	767	6	12	0.035287081	8	1674.370786	8	245.189606
dwt__221.mtx.rnd	221	704	14	27	0.028959276	12	843.9216113	12	252.451385
dwt__234.mtx.rnd	117	162	7	14	0.023872679	8	1.321375608	8	5.617615938
dwt__245.mtx.rnd	245	608	9	17	0.020341251	11	568.676477	11	221.5725074
dwt__310.mtx.rnd	310	1069	20	39	0.022319658	3	434.7463524	3	775.4185748
fs_183_1.mtx.rnd	183	701	3	5	0.042094518	4	881.3040562	4	184.7593374
fs_183_3.mtx.rnd	183	701	3	5	0.042094518	4	880.0048709	4	145.9417377
fs_183_4.mtx.rnd	183	701	3	5	0.042094518	4	879.5726719	4	162.442857
fs_183_6.mtx.rnd	183	701	3	5	0.042094518	4	879.2667568	4	144.5108614
gent113.mtx.rnd	104	549	3	6	0.102501867	6	679.6178894	6	40.2715888
gre_216a.mtx.rnd	216	660	10	20	0.028423773	4	130.1272087	4	178.2710564
gre_216b.mtx.rnd	216	660	10	20	0.028423773	4	130.0929449	4	166.4659793
gre__115.mtx.rnd	115	267	4	8	0.040732265	4	6.596351147	4	15.48992038
gre__185.mtx.rnd	185	650	7	14	0.038190364	4	87.58798862	4	144.0924964
gre__343.mtx.rnd	343	1092	12	24	0.018617974	4	705.0932548	4	730.2715306
ibm32.mtx.rnd	32	90	3	4	0.181451613	5	0.74101615	5	0.509914398
impcol_a.mtx.rnd	206	557	6	11	0.026379351	9	831.15979	9	126.5968835
impcol_b.mtx.rnd	59	281	3	4	0.164231444	6	107.7151272	6	6.038231373
impcol_c.mtx.rnd	137	352	4	7	0.037784457	6	47.03619385	6	31.2838974
lns__131.mtx.rnd	123	275	5	10	0.036652006	4	7.753255844	4	17.84512091

file	nodes	edges	radius	diameter	density	l_horton	time	l	time_de_pina
lund_a.mtx.rnd	147	1151	7	13	0.107259342	3	1733.389991	3	221.8990021
lund_b.mtx.rnd	147	1147	7	13	0.10688659	3	1764.11869	3	212.0995777
nnc261.mtx.rnd	261	794	8	16	0.02340112	6	650.7362192	6	310.564512
nos1.mtx.rnd	158	312	39	78	0.025155204	4	9.794821739	4	33.90350366
nos4.mtx.rnd	100	247	7	13	0.04989899	4	4.887949705	4	13.11513996
plskz362.mtx.rnd	362	880	16	31	0.013467807	4	205.7395854	4	627.3590617
pores_1.mtx.rnd	30	103	3	6	0.236781609	4	0.788919687	4	0.576864958
saylr1.mtx.rnd	238	445	15	29	0.015778463	4	16.72126341	4	92.1688633
steam3.mtx.rnd	80	424	10	19	0.134177215	3	16.64971995	3	20.00543618
west0132.mtx.rnd	132	404	4	7	0.04672681	5	78.51655555	5	38.07433319
west0156.mtx.rnd	156	371	5	7	0.030686518	7	58.39590955	7	34.26397586
west0167.mtx.rnd	167	489	4	7	0.03527884	6	154.1080742	6	57.81367898
will199.mtx.rnd	199	660	4	5	0.033500838	6	383.4127805	6	140.8959146

TABLE 2: Comparaison algorithme de De Pina et de Horton

Étude du tableau

Nous allons regarder en détail les différences entre les instances rouges et les autres dans le tableau 2. premièrement, voici quelques statistiques

- instances en rouge
  - temps moyen : 108 secondes
  - temps médian : 16 secondes
  - nombre moyen de sommets : 202
  - nombre moyen d'arêtes : 495
  - rayon moyen : 11
  - diamètre moyen : 21
  - densité moyenne : 0.03
- instances en noire
  - temps moyen : 134 secondes
  - temps médian : 126 secondes
  - nombre moyen de sommets : 156
  - nombre moyen d'arêtes : 572
  - rayon moyen : 6
  - diamètre moyen : 10
  - densité moyenne : 0.07

Nous pouvons observer que le nombre d'arêtes moyen est plus élevé dans les instances en noir. Il en va de même pour la densité moyenne. Il semblerait cependant que le rayon, le diamètre ainsi que le nombre de sommets ne soient pas des facteurs de ralentissement de l'algorithme. Ce comportement peut de nouveau s'expliquer avec le calcul du rang de la base.

Comparaison supplémentaire

Dans cette partie bonus, nous allons comparer trois algorithmes. Dans la partie sur l'algorithme naïf, nous avons démontré le théorème 7 montrant qu'il est optimal sur les graphes complets. Il est donc intéressant de regarder son temps de calcul par rapport aux autres. La procédure de test est simple, on génère un graphe complet à n sommets, $n = 3$ au début, puis on calcul une base minimale, si le temps de calcul est inférieur à 60 secondes, alors $n \leftarrow n+1$ et on recommence. De plus, une vérification simple de la longueur du plus cycle et de la dimension de la base a été effectuée. Voici les résultats

- Pour l'algorithme de NetworkX : $i = 59$
- Pour l'algorithme de Horton : $i = 40$
- Pour l'algorithme Naïf : $i = 261$

Ce petit programme met en avant l'efficacité assez remarquable de l'algorithme Naïf sur les graphes complets. Il permet également de confirmer l'hypothèse du temps de calcul plus lent de l'algorithme de Horton que celui de De Pina sur les graphes denses.

4.5 Conclusion des tests

Tous ces tests ont mis en avant les forces et les faiblesses de l'algorithme de Horton et de De Pina. On a notamment remarqué que malgré la complexité supérieure de l'algorithme de Horton, le temps de calcul est moins élevé sur des graphes peu dense et avec peu d'arêtes. Cependant, il a été observé qu'il ce comportait mal sur la majorité des graphes. Ceci pourrait être dû à deux facteurs, le tri de l'étape 3, qui peut être long suivant la taille de l'ensemble construit à l'étape 2, ainsi qu'au test d'indépendance linéaire sur des grands ensembles. Une solution à ce problème pourrait être de paralléliser l'algorithme de Horton, en utilisant par exemple, une version parallélisée de la méthode du pivot de Gauss pour l'indépendance linéaire. Il pourrait ainsi être intéressant de refaire ces tests en utilisant une autre libraire Python comme Graph-Tool [16] par exemple. Cette dernière est écrite en C++ et la majorité des algorithmes sont implémentés avec OpenMP. Elle ne dispose cependant pas d'algorithmes de calcul d'une base de cycles, mais pourrait être une piste d'amélioration du temps de calcul tout en gardant la rapidité de développement du Python.

Pour conclure cette partie, nous pouvons dire que l'algorithme de Horton n'est pas utilisable dans le cas général, à moins d'en avoir une version très bien optimisée. L'algorithme de De Pina reste la meilleure option entre les deux pour une utilisation générale hors grands graphes complets. Dans ce cas précis, il est préférable d'utiliser l'algorithme Naïf, celui-ci est y est optimal, simple à implémenter et plus rapide que l'algorithme de Horton et de De Pina.

5 Conclusion

Conclusion générale

Dans ce rapport, nous avons essayé d'interpréter au mieux les résultats des comparaisons des deux algorithmes. Il a été très intéressant d'étudier l'approche des algorithmes de Horton et de De Pina. En effet, pour Horton, son approche algorithmique simple et les nombreuses optimisations possibles permettent de le rendre rapide malgré une complexité assez élevée. Cependant, l'algorithme de De Pina est également très intéressant de par son approche plus algébrique et géométrique. Tout cela pour dire qu'il n'est pas étonnant que la totalité des meilleurs algorithmes actuels reposent sur l'une des deux approches voir les deux en même temps [17]. Cela laisse apparaître le fait qu'il ne faut pas seulement regarder un algorithme sur sa complexité et qu'il est important de faire des tests. Le comparer avec les autres algorithmes existant pour le même problème, tout en essayant de d'isoler les cas pour lesquels l'un est meilleur que l'autre est probablement la meilleur chose à faire.

Pour conclure, nous espérons que nos résultats vont aider à choisir le meilleur algorithme en fonction du graphe et de la tâche à réaliser. Choix qui n'est pas si simple à faire en pratique.

Conclusion personnelle

Pour finir, nous souhaitons chacun écrire une petite conclusion personnelle concernant ce stage.

"Ce stage a été très enrichissant pour moi. En effet, celui-ci m'a permis de découvrir un domaine que je ne connaissais pas vraiment : celui de la recherche. Cela m'a, je pense, permis de développer de nouvelles compétences comme celui d'aller chercher l'information au lieu d'attendre qu'on me la donne et cela ne pourra qu'être bénéfique pour moi. De plus, j'ai pu approfondir mes capacités dans le langage python que je n'ai découvert que cette année grâce au module « Python et Machine Learning », durant la licence, mais que j'ai bien apprécié et l'utiliser au cours de ce projet m'a été très formateur pour ce langage. Enfin, j'ai également pu découvrir un nouveau langage qui est latex et qui pourra être une très bonne alternative à certains logiciels de « traitement de texte » pour mes projets futurs."

Kyanou LOW-KEIN

"Personnellement, ce stage m'a énormément plu. Je l'ai pris comme une véritable invitation à la théorie algébrique des graphes, domaine qui m'intéressait déjà depuis plusieurs années. Cela m'a notamment permis de revoir et d'approfondir mes connaissances en algèbre linéaire et en théorie des graphes. De plus, la majorité des algorithmes ne se trouvent que dans des articles ou des thèses, j'ai donc également dû apprendre à chercher ces articles, puis à en extraire l'essentiel tout en essayant de comprendre le sujet. Néanmoins, j'aurai aimé avoir plus de temps afin de comprendre un peu mieux le Cyclic Bandwidth Problem, pour regarder si il existe des résultats concernant les bases de cycles dans d'autres théories telles que la théorie des groupes ou encore la théorie spectrale des graphes ou encore regarder quelles sont les applications des bases de cycles. En conclusion, je remercie M. Eric Monfroy pour la proposition de ce sujet, que je ne regrette pas d'avoir choisi."

Vincent IMARD

6 Annexes

6.1 Tableau comparatif de l'algorithme de Paton et de l'algorithme Naïf

file	l cycle_basis	nx.cycle_basis	l naive	Naive
494_bus.mtx.rnd	67	0.00099754333496094	26	0.46176433563232
arc130.mtx.rnd	4	0.00099706649780273	9	0.80483603477478
ash292.mtx.rnd	72	0.0019731521606445	49	2.3028419017792
ash85.mtx.rnd	32	0	21	0.12965202331543
bcspwr01.mtx.rnd	17	0	17	0.0039889812469482
bcspwr02.mtx.rnd	9	0	19	0.0079569816589355
bcspwr03.mtx.rnd	16	0.00099754333496094	26	0.07781982421875
bcspwr04.mtx.rnd	47	0.0010001659393311	23	1.1409542560577
bcspwr05.mtx.rnd	58	0.0019948482513428	42	0.70512723922729
bcsstk01.mtx.rnd	15	0.00099754333496094	12	0.07184910774231
bcsstk22.mtx.rnd	48	0.00099754333496094	34	0.17653775215149
can__144.mtx.rnd	48	0	41	0.7400074005127
can__161.mtx.rnd	69	0.00099802017211914	25	0.84270524978638
can__292.mtx.rnd	75	0.0020058155059814	20	2.7556173801422
curtis54.mtx.rnd	13	0	14	0.040891408920288
dwt__209.mtx.rnd	41	0.00099825859069824	28	1.2855200767517
dwt__221.mtx.rnd	49	0.00099658966064453	24	1.1758270263672
dwt__234.mtx.rnd	26	0.00097084045410156	18	0.060830354690552
dwt__245.mtx.rnd	50	0.00099992752075195	28	0.9564037322998
dwt__310.mtx.rnd	120	0.00099730491638184	34	2.7965490818024
fs_183_1.mtx.rnd	11	0.00099778175354004	13	1.1139798164368
fs_183_3.mtx.rnd	11	0.00099706649780273	13	1.1030495166779
fs_183_4.mtx.rnd	11	0.00096607208251953	13	1.1020770072937
fs_183_6.mtx.rnd	11	0.00099802017211914	13	1.1160147190094
gent113.mtx.rnd	14	0.00095462799072266	12	0.55252170562744
gre_216a.mtx.rnd	34	0.00099968910217285	30	0.20940852165222
gre_216b.mtx.rnd	48	0.0010004043579102	26	1.0063064098358
gre__115.mtx.rnd	54	0.0019669532775879	26	2.9870116710663
gre__185.mtx.rnd	78	0.00099849700927734	24	1.0930771827698
gre__343.mtx.rnd	78	0.00099754333496094	24	1.0941030979156
ibm32.mtx.rnd	10	0	11	0.023949861526489
impcol_a.mtx.rnd	53	0.00099682807922363	22	0.88662767410278

file	l cycle_basis	nx.cycle_basis	l naive	Naive
impcol_b.mtx.rnd	8	0.00099754333496094	11	0.16158127784729
impcol_c.mtx.rnd	31	0.00099730491638184	25	0.34410810470581
lns__131.mtx.rnd	11	0.0010006427764893	14	0.19842576980591
lund_a.mtx.rnd	20	0.0010001659393311	18	1.9697294235229
lund_b.mtx.rnd	20	0.0019955635070801	18	1.9607546329498
nnc261.mtx.rnd	42	0.002018928527832	36	1.6176753044128
nos1.mtx.rnd	4	0	8	0.26231026649475
nos4.mtx.rnd	30	0.00099778175354004	17	0.16256475448608
plskz362.mtx.rnd	100	0.0019946098327637	43	3.0679585933685
pores_1.mtx.rnd	12	0	10	0.032698154449463
saylr1.mtx.rnd	36	0	46	0.730388879776
steam3.mtx.rnd	3	0	6	0.41705012321472
west0132.mtx.rnd	28	0	15	0.47099161148071
west0156.mtx.rnd	49	0.00099587440490723	25	0.6126127243042
west0167.mtx.rnd	54	0.00099706649780273	19	1.2349629402161
will199.mtx.rnd	66	0.0059843063354492	21	2.1440119743347
will57.mtx.rnd	5	0.00099706649780273	7	0.076659679412842

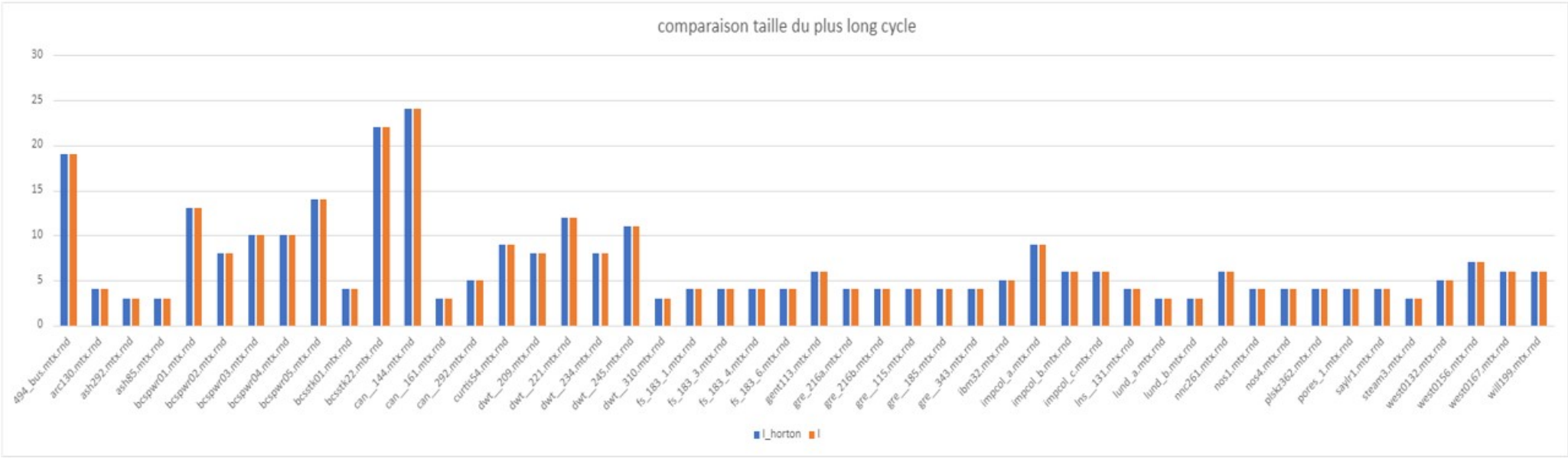

FIGURE 1 – comparaison de la taille du cycle le plus long

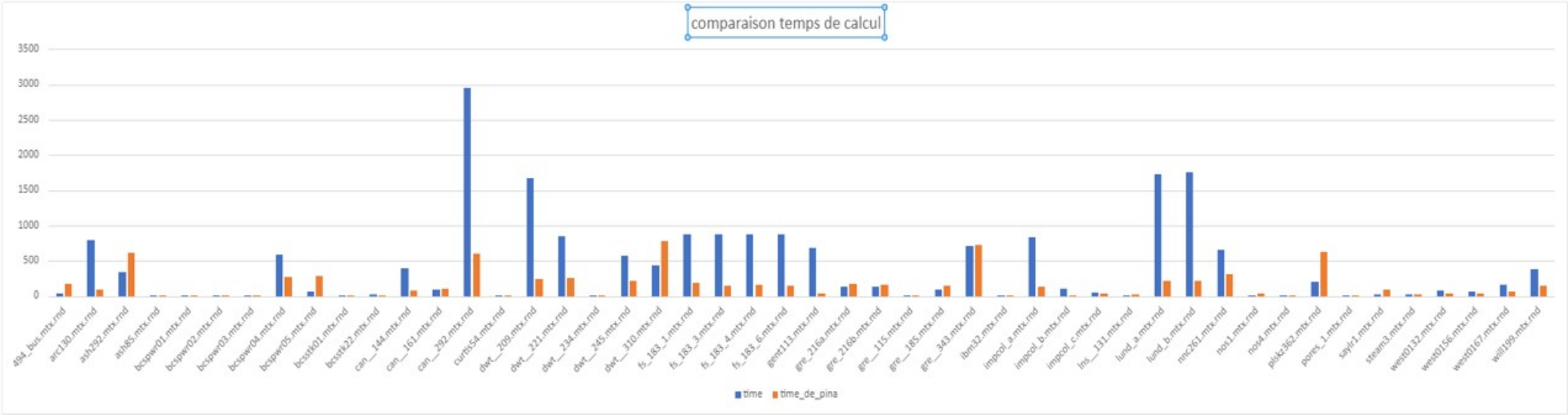

FIGURE 2 – comparaison du temps de calcul

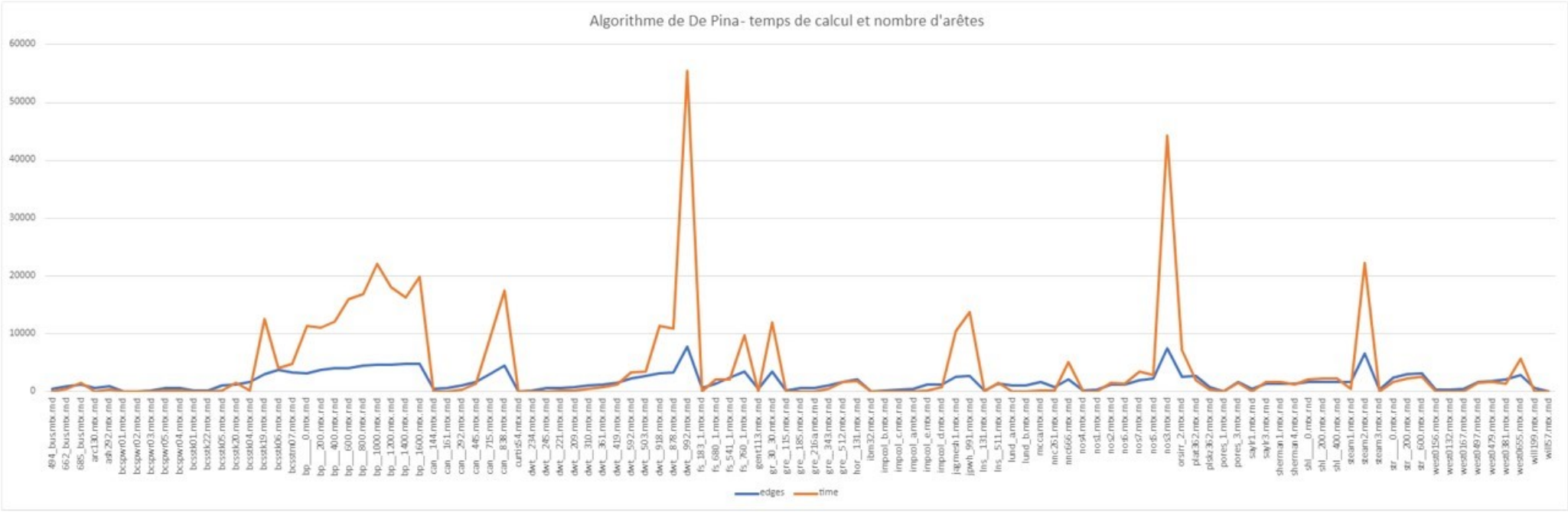

FIGURE 3 – Algorithme de De Pina - temps de calcul sur 109 instances

Références

[1] Eric Monfroy Depres Hugues, Guillaume Fertin. Improved Lower Bounds for the Cyclic Bandwidth Problem. page 14.

[2] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3) :255–259, March 1998.

[3] networkx.algorithms.cycles.cycle_basis — NetworkX 2.5 documentation.

[4] Keith Paton. An algorithm for finding a fundamental set of cycles of a graph. *Communications of the ACM*, 12(9) :514–518, September 1969.

[5] Hugo Tremblay and Fabio Petrillo. The cross cyclomatic complexity : a bi-dimensional measure for program complexity on graphs. *arXiv :2003.00399 [cs]*, February 2020. arXiv : 2003.00399.

[6] Joseph Horton. A Polynomial-Time Algorithm to Find the Shortest Cycle Basis of a Graph. *SIAM J. Comput.*, 16 :358–366, April 1987.

[7] J. C. de Pina. Applications of shortest path methods. December 1995.

[8] Michael Vasquez Otazu. Finding the MCB in Undirected Graph.

[9] Edoardo Amaldi. An improved algorithm for finding minimum cycle bases in undirected graphs. page 28.

[10] Edoardo Amaldi, Claudio Iuliano, Tomasz Jurkiewicz, Kurt Mehlhorn, and Romeo Rizzi. Breaking the O(m2n) Barrier for Minimum Cycle Bases. pages 301–312, September 2009.

[11] Ying Fei Dong, Ding-Zhu Du, and Oscar H. Ibarra. *Algorithms and Computation : 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings*. Springer, December 2009. Google-Books-ID : 48lqCQAAQBAJ.

[12] Telikepalli Kavitha, Kurt Mehlhorn, Dimitrios Michail, and Katarzyna E. Paluch. An $\tilde{O}(m^{2}n)$ Algorithm for Minimum Cycle Basis of Graphs. *Algorithmica*, 52(3) :333–349, November 2008.

[13] Josh Alman and Virginia Vassilevska Williams. A Refined Laser Method and Faster Matrix Multiplication. *arXiv :2010.05846 [cs, math]*, October 2020. arXiv : 2010.05846.

[14] Software for Complex Networks — NetworkX 2.5 documentation.

[15] networkx.algorithms.cycles.minimum_cycle_basis — NetworkX 2.5 documentation.

[16] graph-tool : Efficent network analysis with python.

[17] Kurt Mehlhorn. Cycle Bases in Graphs Structure, Algorithms, Applications, Open Problems. page 65.

[18] Éléments de théorie des graphes BRETTO Alain, FAISANT Alain, HENNECART François.