Calcul exact de la pathlength par branch and bound

*

Rapport de stage de M1

Pierre Pébereau

Encadrants :
David Coudert, COATI, Directeur de Recherche
Nicolas Nisse, COATI, Chargé de Recherche

Contents

1 Introduction

1.1 Motivations

Une décomposition arborescente d'un graphe et une représentation du graphe sous la forme d'un arbre dont les sommets, appelés sacs, sont des séparateurs du graphe (un sommet de l'arbre est un sous ensemble de sommets du graphe donc la suppression déconnecte le graphe). La largeur d'une décomposition arborescente est la taille du plus grand sac de la décomposition, et sa longueur est la plus grande distance entre les sommets d'un sac. Ces décompositions apportent des informations sur la structure et les propriétés métriques des graphes et sont utilisées dans la conception d'algorithmes de programmation dynamique.

La largeur arborescente (treewidth) ainsi que la pathwidth ont été le sujet de plusieurs papiers et implémentations expérimentales. À ma connaissance, il existe une implémentation du calcul exact de la treelength [**4**] et aucune pour la pathlength.

Durant ce stage, nous avons conçu un algorithme exact pour calculer la pathlength des graphes [**5**], c'est-à-dire trouver une décomposition linéaire d'un graphe (cas où la décomposition arborescente est un chemin) de longueur minimum. Mon implémentation permet de calculer en un temps raisonnable (10 minutes sur un ordinateur portable générique) la pathlength de graphes de moins de 40 sommets, et au-delà pour certaines classes de graphes.

1.2 Définitions et résultats folkloriques

Definition 1 (path-decomposition) *Soit $G = (V, E)$ un graphe non orienté et sans boucles.*
On appele path-decomposition de G toute suite $(X_i)_{1\leq i\leq r}$ de sous-ensembles de V telle que :

1. $\bigcup_{1\leq i\leq r} X_i = V$
2. *Pour tout* $1 \leq i \leq j \leq k \leq r, X_i \cap X_k \subseteq X_j$
3. *Pour tout* $(u, v) \in E$, *il existe* $1 \leq i \leq r$ *tel que* $u \in X_i$ *et* $v \in X_i$

On appele largeur de la décomposition la quantité $w(X) := \max_{1\leq i\leq r}(|X_i|) - 1$. La pathwidth du graphe est le minimum de cette quantité sur l'ensemble des path-decompositions de G, notée $pw(G)$.

On appele longueur de la décomposition la quantité $l(X) := \max_{1\leq i\leq r}(diam(X_i))$ La pathlength du graphe est le minimum de cette quantité sur l'ensemble des path-decompositions de G, notée $pl(G)$.

Pour $s \in V$, on notera $X(s)$ le premier sac contenant s dans la décomposition X.

Definition 2 (layout) *Soit $G = (V, E)$ un graphe. On appele layout de G toute permutation de V. Leur ensemble est noté $\mathcal{L}(V)$. Soit $S \subseteq V$, $L = v_1...v_n$ un layout de G.*

- *On appele préfixe toute suite de sommet $v_1...v_i$ pour $1 < i \leq n$.*
- *On appele concaténation la composition de layouts de supports disjoints.*
- *Pour P un préfixe, on note $L_P(V) = \{P \circ Q | Q$ permutation de $V \setminus V(P)\}$ l'ensemble des permutations préfixées par P.*

Definition 3 (Décomposition induite) *Soit $G = (V, E)$ un graphe, L un layout de G. On note $|V| = n$. On appele décomposition induite par L sur G la suite définie par :*
Pour $1 \leq i \leq r, X_i = \{L^{-1}(i)\} \cup V_L(i)$ avec

$V_L(i) = \{u \in V | L(u) \leq i \text{ et } \exists w \in V, L(w) > i \text{ et } uw \in E\}$
X_i *est appelé sac induit par* $L^{-1}(i)$. *On remarque que* $X_i = X(L^{-1}(i))$.

Lemma 1 (Kinnersley) *Soit* $G = (V, E)$ *un graphe,* L *un layout de* G. *Alors la décomposition induite par* L *sur* G *est une path-decomposition. Reciproquement, toute path-decomposition de* G *induit un layout de* G.

Ce résultat est utilisé par Kinnersley pour montrer l'équivalence des problèmes de la pathwidth et de la vertex separation.

En pratique, il nous permet de considérer des layouts, qui sont parfois plus faciles à manipuler que des décompositions.

Le lemme suivant permet d'encore faciliter la manipulation des layouts : les sacs induits par un préfixe ne dépendent pas de l'ordre du suffixe et inversement.

Lemma 2 *Soit $G = (V, E)$ un graphe, $S \subset V$, P un layout de S et Q un layout de $V \setminus S$.*
Alors les sacs induits par les sommets de P ne dépendent que de P et $V \setminus S$ et les sacs induits par les sommets de Q ne dépendent que de Q et S.

Proof.
C'est vrai par définition.

■

En particulier, pour P un préfixe, on notera (abusivement) $l(P)$ le diamètre maximal d'un sac induit par un sommet de P et on parlera de décomposition induite par P pour désigner les sacs induits par les sommets de P.

Theorem 1 (Dragan, Kohler, Leitert, 2-approximation de la pathlength) *Soit $G = (V, E)$ un graphe.*
Soit $(X_i)_{1 \leq i \leq r}$ une décomposition optimale pour la pathlength sur G de length λ.
Soit $s \in X_1$, et $(L_i)_{1 \leq i \leq n}$ un layering du graphe depuis s.
Alors la décomposition $(s, \{s\} \cup L_1, ...L_i \cup L_{i+1}...)$ est de length au plus 2λ.

Lemma 3 (Dourisboure, Gavoille) *Soit $G = (V, E)$ un graphe. Si G admet un cycle de longueur k comme sous-graphe isometrique, alors $pl(G) \geq \lfloor \frac{k}{2} \rfloor$.*

Contrairement au problème de la recherche de plus long cycle et de plus long cycle induit, problèmes NP-complets, la recherche de plus grand cycle isometrique est un problème polynomial d'après Lohkstanov. En particulier, il propose un algorithme dont une première version a été implémentée par Théo Qui.

On introduit le graphe Y_k pour $k \in \mathbb{N}$:
Il s'agit d'un arbre dont les trois branches issues de la racine sont des chemins de longueur k.

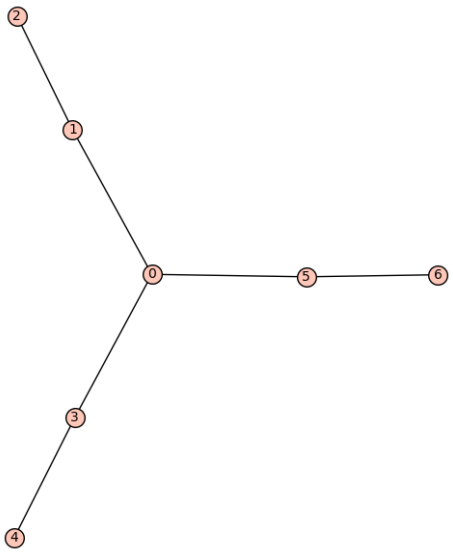

Figure 1: Y_2

Lemma 4 (Dourisboure, Gavoille) *Soit $G = (V, E)$ un graphe. Si G admet Y_k comme sous-graphe isometrique, alors $pl(G) \geq k$.*

2 Résultats

Lemma 5 (Permutations de préfixe) *Soit P,P' deux layouts de $S \subset V$ tq $l(P) \leq l(P')$.*
Alors $\min_{L \in L_P(V)} l(L) \leq \min_{L' \in L_{P'}(V)} l(L')$.

Proof. Soit Q un layout de $V \setminus S$. Montron que $l(P \circ Q) \leq l(P' \circ Q)$.

On note $(X_i)_{1 \leq i \leq n}$ la décomposition induite par $P \circ Q$ et $(X'_i)_{1 \leq i \leq n}$ la décomposition induite par $P' \circ Q$.

Les sacs induits par le suffixe ne dépendent pas de l'ordre du préfixe d'après le Lemme 2, donc :

$$\begin{aligned} l(P \circ Q) &= \max_{1 \leq i \leq n} diam(X_i) \\ &= \max(\max_{1 \leq i \leq r} diam(X_i), \max_{r < i \leq n} diam(X_i)) \end{aligned} \tag{1}$$

De même :
$l(P' \circ Q) = \max(\max_{1 \leq i \leq r} diam(X'_i), \max_{r < i \leq n} diam(X_i))$
Or $\max_{1 \leq i \leq r} diam(X_i) \leq \max_{1 \leq i \leq r} diam(X'_i)$ par hypothèse.
D'où $l(P \circ Q) \leq l(P' \circ Q)$, d'où la conclusion. ■

Lemma 6 (Extension de préfixe 1) *Soit P un layout de $Q \subset V$. Soit $(X_i)_{1\leq i\leq r}$ la décomposition de $G[Q]$ induite par P. Soit F la frontière de P.*

Soit $B \subset V \setminus S$ un sous ensemble de sommets tel que :

- $(N(B) \setminus B) \subset X_r$
- $diam(B \cup F) \leq l(P)$

Alors $\min_{L\in L_{P\circ B}(V)}(l(L)) \leq \min_{L\in L_P(V)}(l(L))$.

Proof.

Notons $B = \{b_1, ..., b_m\}$ avec $m = |B|$ et $S = V \setminus (Q \cup B)$. On note $|S| = r$. Soit L un layout optimal parmi ceux préfixés par P. Il existe des suites de sommets $(S_i)_{1\leq i\leq m+1}$ formant une partition de S telles que
$L = PS_1b_1...S_mb_mS_{m+1}$.
Soit $L' = Pb_1...b_mS_1...S_{m+1}$.
Notons $(X_i)_{1\leq i\leq n}$ les sacs induits par L et $(X'_i)_{1\leq i\leq n}$ les sacs induits par L'. D'apres un lemme précedent, il suffit de considérer les sacs des suffixes.

Soit $j \in [r, n]$.

- Si $L'(j) \in B$:

Alors le sac X'_j ne contient que des sommets de $B \cup X_r$ par sa position dans le layout. Or par hypothèse, $diam(B \cup X_r) \leq l$. Donc $diam(X'_j) \leq l$.

- Si $L'(j) = s \in S$:

Par hypothèse, $N(B) \subset B \cup X_r$, or l'ensemble B a été traité entierement auparavant, donc aucun des sommets de B figure dans un sac du suffixe. Pour tout $b \in B$, il existe $i \leq r$ tel que $b \in X_i$. Or $N(B) \subset B \cup X_r$, donc pour tout $j \geq r + m + 1, B \cap X_j = \emptyset$.
Et par ailleurs, l'ordre des sommets dans les $(S_k)_{k\in[1,m]}$ a été maintenu entre L et L'.
Donc $\forall u \in X'_j, u \in X(s)$ et donc $diam(X'(s)) \leq diam(X(s))$.

En conclusion :
$l(L') = l(P \circ B \circ S) = \max(l(P), \max_{r+m<i\leq n} diam(X'_i))$
et
$l(L) = max(l(P), \max_{r<i\leq n} diam(X_i))$
Or on a montré $\max_{r+m<i\leq n} diam(X'_i) \leq \max_{r<i\leq n} diam(X_i)$.

D'où la conclusion. ■

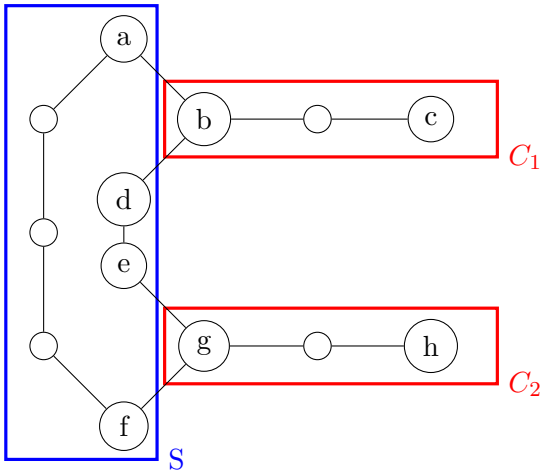

Figure 2: Contre exemple du raisonnement composante par composante

Remarque : Un tel B est une composante connexe de $V \setminus S$. Ce fait peut suggérer qu'il est pertinent de traiter les composantes connexes de $V \setminus S$ les unes après les autres, même sans hypothèses sur leur diamètre. Ce résultat est cependant faux sans plus d'hypothèse : voici un contre exemple.

On suppose que S est l'ensemble des sommets du préfixe. On observe que la situation est entièrement symètrique entre C_1 et C_2. Supposons donc qu'on traite entieremment C_1 avant C_2 : c et f partagerons alors un sac, et la length sera donc au moins 6.
Au contraire, si l'on traite d'abord g puis tout C_1, c et f ne partagent plus de sac et la length est de 5. On a cependant traité C_1 avant de terminer de traiter C_2.

Lemma 7 (Extension de préfixe 2) *Soit $G = (V, E)$ un graphe, $S \subset V$ et P un layout quelconque de S. On note $F = \{u \in S, \exists w \in V \setminus S$ tel que $uw \in E\}$ (frontière de S)*

S'il existe $x \in N(S)$ tel que :

- $diam(\{x\} \cup F) \leq l(P)$
- $\forall v \in (V \setminus S) \cap N(F), d(x, v) \leq \max_{y \in F} d(y, v)$

Alors $\min_{L \in L_{Pox}(V)} l(L) \leq \min_{L \in L_P(V)} l(L)$.

Proof. Soit $L \in L_P(V)$, que l'on note $L = P \circ Q_1 \circ x \circ Q_2$ avec Q_1 et Q_2 deux suites de sommets.
On pose $L' = P \circ x \circ Q_1 \circ Q_2$. On note $(X_i)_{1 \leq i \leq n}$ les sacs induits par L et $(X'_i)_{1 \leq i \leq n}$ les sacs induits par L'.

Montrons que $l(L') \leq l(L)$.
On remarque que $X'(x) = F \cup \{x\}$, donc $diam(X'(x)) \leq l(P)$ par la première hypothèse.
Les sacs induits par des sommets de Q_2 ne changent pas, donc les sacs à considérer sont engendrés par des sommets de Q_1.
Soit $q \in Q_1$. Par position de q dans les layouts, $X'(q) \setminus \{x\} \subset X(q)$.
Soit $v \in X'(q) \setminus \{x\}$. Seul le couple (x, v) pour un tel v est susceptible de faire apparaître un diamètre supérieur à $l(L)$ d'après la remarque précédente.
$x \in N(F)$, donc il existe $y \in F$ tel que $y \in X(q)$ et par définition $d(v, y) \leq l(L)$.
Soit v' le dernier sommet d'un plus court chemin de v à y tel que v' n'appartienne pas à F. Un tel sommet existe car $y \in F$ et $v \notin F$, éventuellement confondu avec v. De plus $v' \in N(F)$.
Alors $d(v, y) = d(v, v') + d(v', y)$, or par hypothèse $d(v', y) \leq d(v', x)$.
Donc $d(v, x) \leq d(v, v') + d(v', x) \leq l(L)$.

D'où $l(L') \leq l(L)$. ■

Ce lemme n'as pas été utilisé dans l'implémentation courante.

Conjecture 1 *Soit $G = (V, E)$ un graphe. Alors il existe un layout atteignant la pathlength et dont le premier sommet est un sommet d'excentricité maximale.*

Nous avons montré qu'une telle approche permet au moins une 2-approximation de la pathlength.

Lemma 8 *Soit $G = (V, E)$ un graphe connexe, $X = (X_i)_{1 \leq i \leq n}$ une path-decomposition de G de length λ. Alors il existe une path-decompoition X' de G dont le premier sac contient un sommet d'excentricité maximale et tel que $l(X') \leq 2\lambda$.*

Proof. Notons premierement que contrairement à ce qu'on a vu précédemment, on raisonne ici sur des path-decompositions et non des layouts.

Soit s, t un diamètre du graphe tel que s apparait dans X dans un sac X_i antérieur au premier sac contenant t. On défini X' par :
Pour $j < i$: $X'_j = X_j \cup \{s\}$
Pour $j \geq i$: $X'_j = X_j$

Ces sacs forment toujours une path-decomposition.

Montrons que ces premiers sacs ne peuvent avoir pour diamètre qu'au plus 2λ. Toute paire de sommet qui partageait déjà un sac dans X ne peut être à une distance de plus que λ par

définition. On considère donc une paire (x, s) : G étant connexe, il existe un chemin de x à t. Or x figure dans un sac antérieur au sac X_i et t dans un sac ultérieur, donc le sac X_i contient un séparateur de x et t par définition d'une path-decomposition. Notons c un sommet d'un plus court chemin de x à t appartenant à ce séparateur.
s, t est un diamètre, donc $d(x, t) \leq d(s, t)$.
Or $d(x, t) = d(x, c) + d(c, t)$ car c appartient à un plus court chemin entre ces deux sommets.
et $d(s, t) \leq d(s, c) + d(c, t)$ par inégalité triangulaire.
Donc en soustrayant, $d(x, c) \leq d(s, c)$.
Or s et c appartiennent tout deux au sac X_i, donc $d(s, c) \leq \lambda$ donc $d(x, c) \leq \lambda$
Et $d(x, s) \leq d(x, c) + d(c, s)$ par inégalité triangulaire. D'où $d(x, y) \leq 2d(c, s) \leq 2\lambda$.

■

Definition 4 *Soit $G = (V, E)$ un graphe, $L = v_1...v_n$ un layout de V. On dit que L est connecté si $\forall 1 < i \leq n, v_i \in \bigcup_j < iN(j)$.*

Conjecture 2 *Soit $G = (V, E)$ un graphe. Alors il existe un layout connecté atteignant la pathlength.*

3 Implémentation

L'implémentation que j'ai réalisée s'est d'abord appuyée entierement sur Sagemath, puis s'en légerement détachée au passage à Cython. L'algorithme se fait en deux temps : une phase de pré-calculs et une phase de branch and bound.

3.1 Pré-calculs

‘ La première étape de précalcul est de convertir le graphe en entrée en un format pratique : à savoir une matrice de distance. On calcule au passage les excentricités de tout les sommets. La seconde est l'obtention de bornes inférieures et supérieures pour le branch and bound.

La borne supérieure initiale est donnée par la 2-approximation du théorème 1. On obtient une première borne inférieure (généralement mauvaise) en divisant cette valeur par 2.

Le choix d'une bonne borne inférieure est plus délicat. La méthode la plus efficace est la recherche de cycle isométrique (en conséquence du lemme 3), dont l'état de l'art est la méthode de Lohkstanov. Cette recherche est polynomiale. Une première implémentation en sage a été réalisée par Théo Qui, mais contenait quelques erreurs qui empiraient la compléxité théorique. J'en propose une version corrigée qui est encore suboptimale sur le calcul de la puissance de graphe mais qui doit se rapprocher nettement de la complexité asymptotique.

Une deuxième solution pour trouver une borne inférieure est inspirée du Lemme 4 de Douris-boure et Gavoille : il s'agit de chercher un 'spider-graph' comme sous-graphe isométrique. Cependant, la recherche exhaustive d'un tel sous graphe n'est pas un problème polynomial, j'en propose donc une solution partielle : on explore tout les diamètres du graphe et on cherche des spider-graphs dont ces diamètres sont des sous-graphes.

L'amélioration de la borne inférieure est essentielle car elle permet d'interrompre le branch and bound le plus tôt possible.

3.2 Branch and Bound

Le branch and bound s'appuie sur les lemmes présentés en section 2 et l'intuition des conjectures présentées à la fin de ces mêmes sections.

En particulier, l'exploration d'une branche se résume à l'exploration d'un layout des sommets préfixés par un premier sommet. Le choix de ce premier sommet est guidé par la conjecture 1 : on trie les sommets par ordre d'excentricité décroissante, et on explore les layouts préfixés par un sommet d'excentricité maximale en premiers.

Pour comprendre la logique de l'algorithme, qui peut paraître désarticulée, voici le raisonnement que j'ai employé (il se peut que cette idée ne soit pas la meilleure, il faudrait alors articuler les étapes différement) : les étapes coûteuses sont exécutées après celles qui permettent une coupure : on s'épargne ainsi quelques calculs inutiles.

On met à jour la borne supérieure courante à chaque fois qu'on atteint une feuille meilleure que l'optimal antérieur . On maintient un ensemble de préfixes déjà explorés : une permutation d'un tel préfixe n'est pas toujours pertinente d'après le lemme 5.

Un noeud du branch and bound correspond à un préfixe P, un layout d'un sous ensemble $S \subset V$. La première étape dans un noeud est la recherche d'un préfixe déjà exploré et stocké qui soit une permutation de P. Si un tel préfixe existe et avait une length courante inférieure ou égale à celle de P, on peut cesser l'exploration de la branche courante.

Si ce n'est pas le cas, on procède aux étapes suivantes. On calcul la frontière F du préfixe.

On énumère ensuite les sacs hypothétiques : il s'agit de $F \cup \{v\}$ pour $v \in V \setminus S$: on en calcule les diamètres et on les trie par diamètre croissant. On note qu'on peut éliminer tout sommet dont le diamètre du sac hypothétique est supérieur à la borne supérieure.

Si à cette étape il n'y a pas de candidat valide, on interrompt l'exploration de la branche courante.

Sinon, on cherche un candidat au lemme 6 parmi les composantes connexes de $V \setminus S$. On note qu'il n'est pas nécessaire de recalculer ces composantes à chaque itération : si l'on a ajouté un sommet au préfixe, seule sa composante à l'itération précédente à une chance d'être changée. Si l'on a ajouté une composante entière à l'itération précédente, les autres n'ont pas évoluées.

Si aucune composante n'est un candidat satisfaisant, on explore récursivement les branches $P \circ v$ dans l'ordre donné par les diamètres des sacs. Après avoir exploré le reste de la branche, on ajoute P à la liste des préfixes à condition qu'il soit satisfaisant. On limite le nombre de préfixes stockés.

3.3 Pseudocode

Algorithm 1 Branch and bound with greedy exploration steps for pathlength

Input: Graph $G = (V, E)$, prefix P, current length D, upper bound U, lower bound L, set $prefix\_storage$
Output: $K \in \mathbb{N}$ the pathlength of G.

```
if |P| = n then
    if l(P) < U : then
        return U
    else
        return K
    end if
end if
if P ∈ prefix_storage then
    return K
end if
F ← { Frontier of P}
L ← [ ]
for v ∈ V \ V(P) do
    d ← diam(F ∪ {v}
    if d ≤ K then
        L.append((d, v))
    end if
end for
for c ∈ connected_components(V \ V(P)) do          ▷ Extension based on lemma 6
    if diam(c ∪ F) ≤ K then
        return Algorithm 1 (G, P ∘ c, D, U, L, prefix_storage)
    end if
end for
Sort L in order of increasing diameters
for (d, v) ∈ L do                                  ▷ Greedy exploration
    if d ≤ U then
        U ← Algorithm 1(G, P ∘ v, D, U, L, prefix_storage)
    end if
end for
if D ≤ U and |P| ≤ max_prefix_length and |prefix_storage| < max_prefix_number then
    prefix_storage.add(P)
end if
return U
```

4 Utilitaires

J'ai developpé quelques fonctions utilitaires pour me faciliter la vie, peut-être seront-elles utiles à quelqu'un d'autre.

Dans pathlength.pyx

- UnionFind, une classe cython qui implémente la structure de donnée Union-Find.
- layout_to_decomp, une fonction qui prend en entrée un graphe et un layout des sommets et renvoie la path-decomposition induite.
- length2, une fonction qui prend en entrée un graphe et un layout, et qui renvoie la length de la path-décomposition induite .
- is_path_decomposition, une fonction qui prend en entrée un graphe et une liste de listes de sommets et vérifie que cette derniere constitue une path decomposition.
- decomp_to_layout, une fonction qui prend en entrée un graphe et une path decomposition et qui renvoie le layout induit.

Dans test.sage

- randomOuterPlanar, une fonction qui prend en entrée un entier n et renvoie un graphe outerplanar 2-connexe à n sommets. Le graphe est construit en ajoutant de cordes aléatoirement à un cycle.
- randomConnectedGNP, une fonction qui prend en entrée un entier n et renvoie un graphe connecté généré par GNP.

Il y a quelques fonctions triviales qui sont de simples traductions en C++ de methode python (notamment vflatten qui est l'équivalent de flatten pour des vectors C++) ou des parsers.

5 Résultats expérimentaux

Les fonctions présentées précédemment ont été testées sur des graphes générés aléatoirement par les méthodes GNP, randomOuterPlanar, et enfin les graphes de la collection ROME. Les figures suivantes répertorient les temps d'exécutions de différentes méthodes. On trouve notamment à part les temps d'exécutions des fonctions de précalculs qui soulignent leurs coûts respectifs.

5.1 Pré-calculs

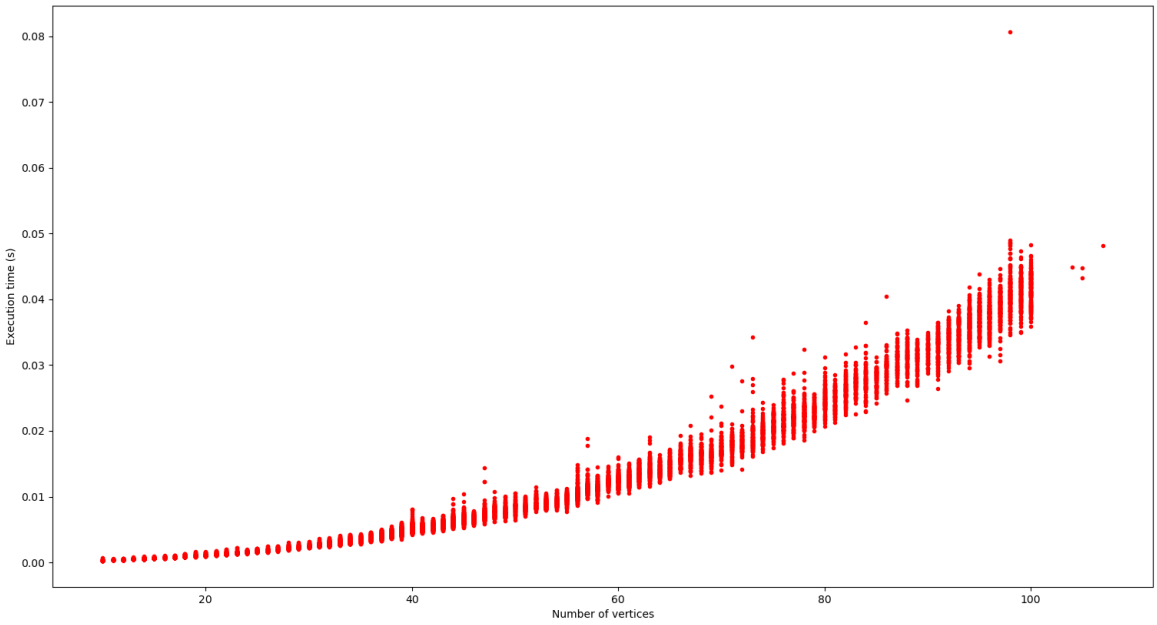

Figure 3: Calcul de la 2-approximation sur les graphes ROME

La principale piste d'amélioration du temps de pré-calcul est le calcul d'un plus long cycle isométrique. La partie coûteuse de l'algorithme est le calcul d'une puissance de graphe, qui est actuellement extrêmement naïve (car ce n'est pas un bottleneck comparable au calcul de la pathlength). On pourra se référer à l'article de Lohkstanov pour plus de détails, mais une implémentation par puissance matricielle est prometteuse.

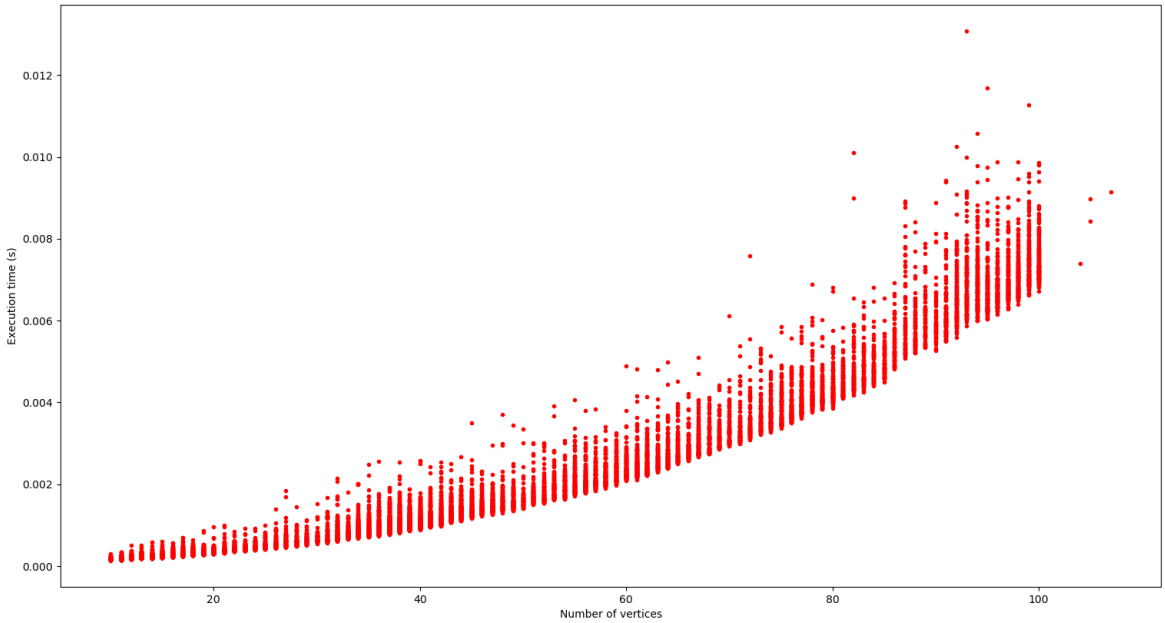

Figure 4: Calcul de la borne inférieure par spider-graph sur les graphes ROME

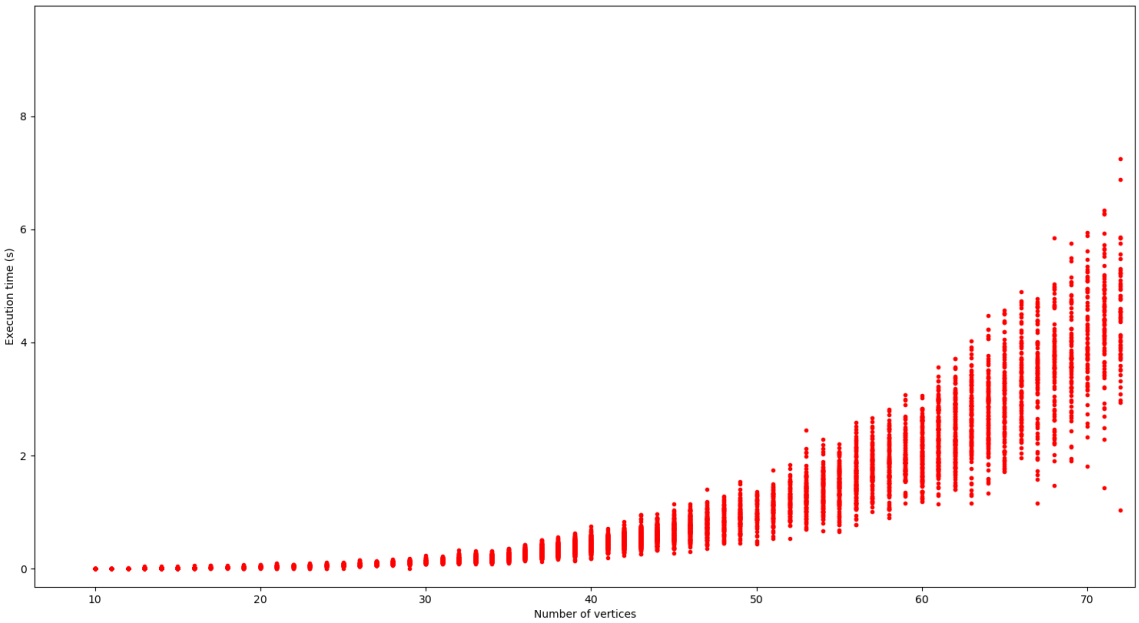

Figure 5: Calcul de la borne inférieure par plus grand cycle isométrique sur les graphes ROME

5.2 Calcul de la pathlength

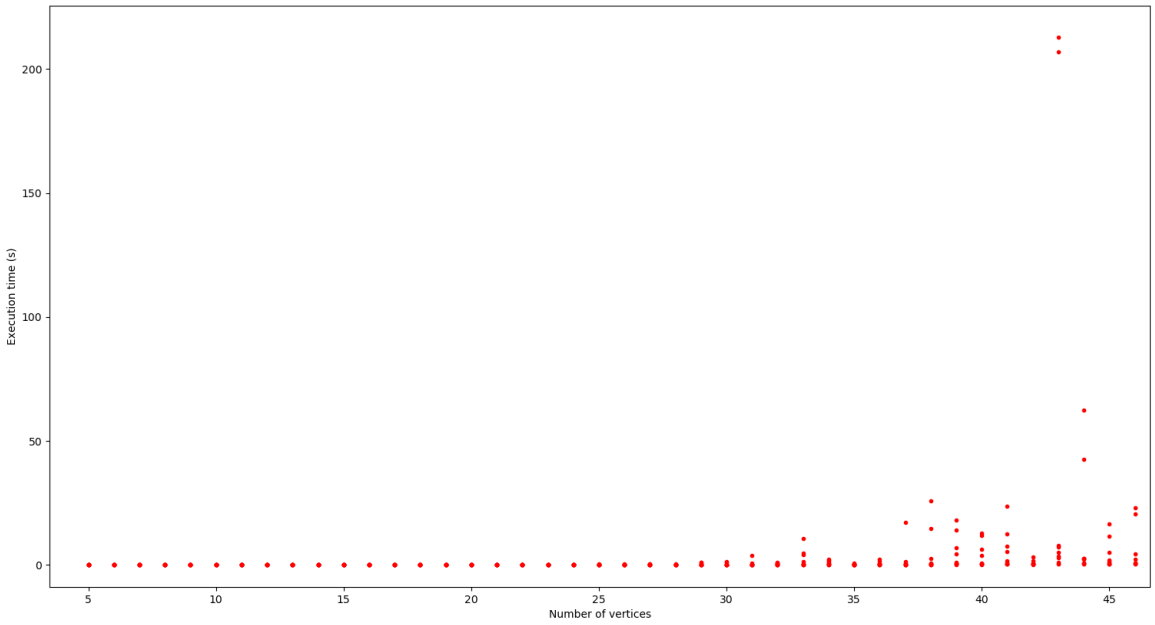

Figure 6: Calcul de la pathlength sur des graphes outerplanar générés aléatoirement

Pour les outerplanar (Figure 6), 1 graphe sur les 420 a eu un temps de calcul supérieur à 10 minutes et a été ignoré. On a étudié les temps de calcul pour des graphes de 5 à 47 sommets, avec 10 graphes pour chaque taille.
Pour ROME (Figure 7), 32 graphes sur les 4312 de taille inférieure ou égale à 40 ont eu un temps de calcul supérieur à 10 minutes et ont été ignorés.

De manière générale, il y a 2 catégories de graphes : ceux pour lesquels le précalcul permet de trouver immédiatement la solution : la borne inférieure est égale à la borne supérieure, et ceux pour lesquels ce n'est pas le cas. Dans la première catégorie, la solution est donc trouvée en temps polynomial. Des exemples de tels graphes sont les cycles ou les graphes contenant un cycle isomètrique de taille égale au double de la pathlength.

Dans cette seconde catégorie, on peut encore raffiner la distinction : ceux pour lesquels la borne inférieure est égale à la pathlength, et ceux ou ce n'est pas le cas. Dans le premier cas, il est essentiel de trouver vite une bonne solution : on peut alors conclure l'exploration le plus tôt possible. Des exemples de tels graphes sont les grilles. Dans le second cas, ca ne fait pas autant de différence : on va se voir obliger d'explorer toutes les branches (même si beaucoup de noeuds seront coupés).

On observe donc deux régimes distincts dans une représentation graphique des temps d'éxecutions : une branche polynomiale et une branche exponentielle.

Malgré la quantité réduite de données pour les graphes outerplanars, on note que l'exécution est comparativement beaucoup plus rapide sur cette classe de graphes que pour des graphes quelconques (générés par la méthode GNP ou choisi parmi les graphes ROME).

On note qu'il reste plusieurs endroits dans l'exécution de l'algorithme de branch and bound où l'ordre initial des sommets (induit par les labels initiaux) peut avoir une influence : c'est

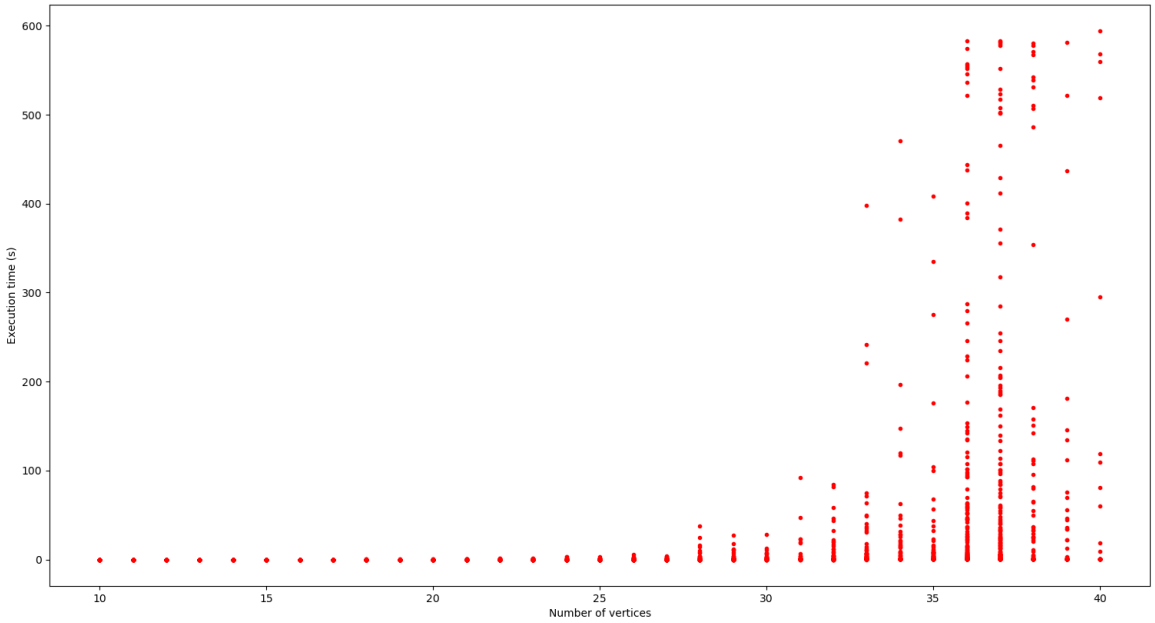

Figure 7: Calcul de la pathlength sur les graphes ROME

les cas où il faut rompre l'ambiguïté, par exemple lorsque les eccentricités de deux sommets sont égales ou le diamètre de leurs sacs induits dans un préfixe sont égaux. Cela implique que certaines permutations initiales peuvent ralentir ou accélerer l'exécution (et ce phénomène a été observé en pratique) : il serait pertinent de trouver des critères pour rompre cette dépendance à l'ordre initial.

À titre de comparaison, j'ai représenté en figure 8 et 9 les temps de calculs pour une fonction de branch and bound allégée : on n'utilise plus de lemmes d'extensions, seulement l'heuristique gloutonne pour le choix du sommet suivant du préfixe (et le stockage de préfixes). Dans le cas des graphes outerplanar, on teste bien sûr sur les mêmes graphes. Dans les deux cas, on teste avec la même permutation initiale des sommets.

On ne note pas de différence fondamentale entre les deux méthodes. Cela suggère que le surcoût entraîné par les calculs de composantes connexes sont compensés par la quantité de noeuds coupés. Pour autant, cela ne permet pas de prédire la façon dont cette proximité se prolongera ou non avec de plus grandes instances.

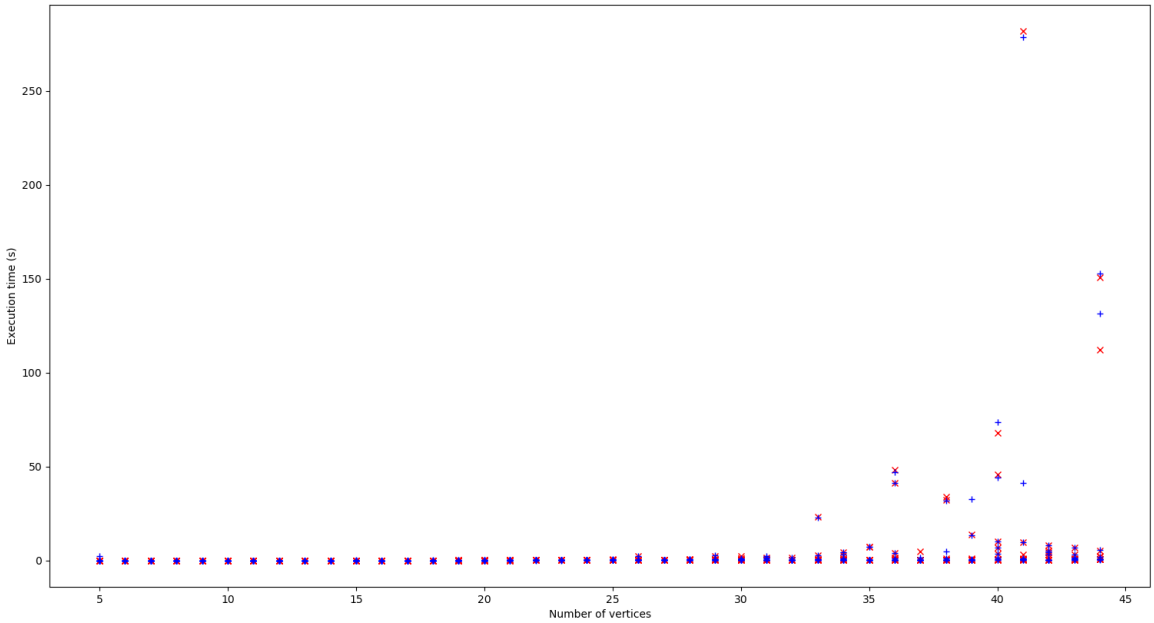

Figure 8: Calcul de la pathlength sur les graphes Outerplanar - en bleu l'algorithme alégé

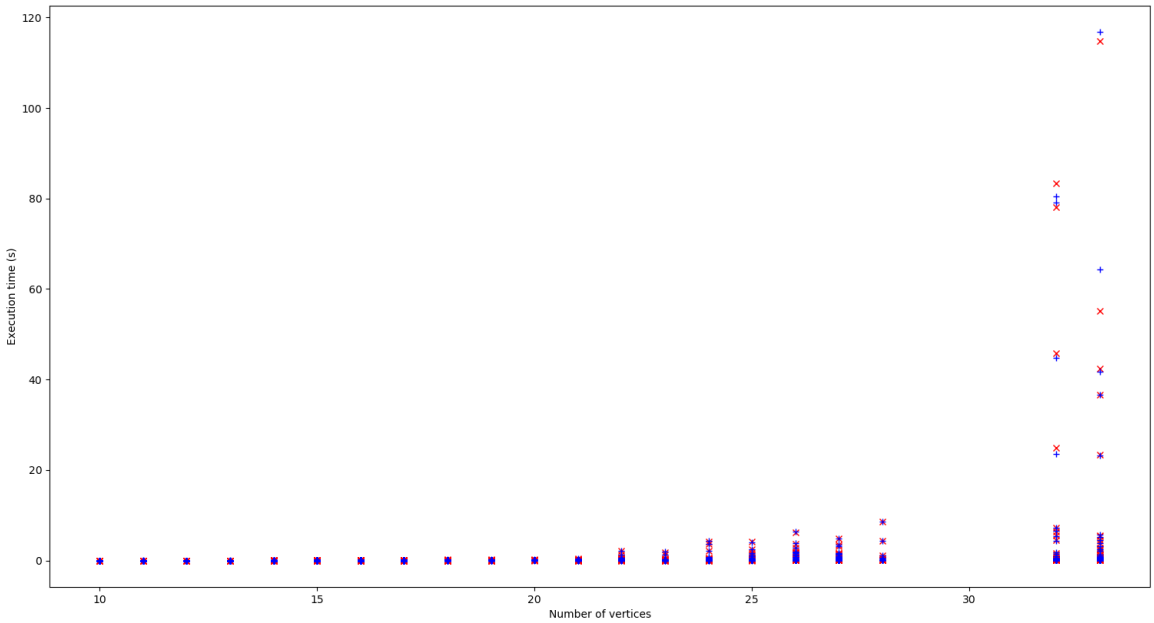

Figure 9: Calcul de la pathlength sur les graphes ROME - en bleu l'algorithme alégé

Pour résumer, les pistes à explorer sont les suivantes :

- Les deux conjectures à valider ou invalider
- Optimisation des précalculs
- Nouvelles bornes inférieures (en particulier, enrichir la recherche de spider-graphs)
- Retirer toute dépendance à l'ordre initial des sommets
- Simplifier chaque noeud du branch and bound : il faudrait être capable de ne pas recalculer les diamètres naïvement.

6 Bibliographie

[0] David Coudert, Dorian Mazauric, Nicolas Nisse. Experimental Evaluation of a Branch and Bound Algorithm for Computing Pathwidth and Directed Pathwidth. ACM Journal of Experimental Algorithmics, Association for Computing Machinery, 2016
[1] Yon Dourisboure and Cyril Gavoille. Tree-decompositions with bags of small diameter. Discrete Math, 2005
[2] Daniel Lohkstanov. Finding the longest isometric cycle in a graph. Discrete Applied Math, 2009
[3] Nancy G. Kinnersley. The Vertex Separation Number of a Graph equals its Path-Width. Inf. Process. Lett. 42(6): 345-350 (1992)
[4] Tuukka Korhonen. Finding Optimal Tree Decompositions, MSc Thesis, (2020) [5] https://github.com/pir2/pathlength