Université de Mons
Faculté des Sciences
Département d'Informatique

Représentation "Straight line" d'un graphe planaire par l'algorithme Shift method

Directeur : M^r Jef WIJSEN

Mémoire réalisé par
Youness KAZZOUL

En vue de l'obtention du grade de Master en Sciences Informatiques

Année académique 2022-2023

1 Dédicace

Je dédie ce travail à ma famille, qui a toujours été mon soutien et mon inspiration tout au long de mes études. Leur amour inconditionnel et leurs encouragements constants m'ont permis d'atteindre mes objectifs. Je tiens également à remercier mes amis et mes proches pour leur soutien et leur motivation. Ce travail est dédié à tous ceux qui ont cru en moi et ont été présents à chaque étape de mon parcours académique.

2 Remerciements

Tous mes sincères remerciements à mon directeur de mémoire, Jef WIJSEN, pour sa précieuse guidance, son soutien continu et ses conseils avisés tout au long de la réalisation de ce travail. Sa vision, son expertise et son dévouement ont été essentiels pour mener à bien cette recherche.

Je souhaite également exprimer ma gratitude envers mes professeurs pour leurs enseignements de qualité, leur encadrement et leur disponibilité. Leur expertise et leur soutien ont grandement contribué à mon parcours académique et ont été essentiels à la réussite de ce mémoire.

Enfin, je tiens à exprimer ma reconnaissance envers toutes les personnes qui ont participé de près ou de loin à ce travail et qui ont contribué à son aboutissement.

3 Contenu du travail de fin d'étude

Ce travail de fin d'étude existe en deux parties :

1. Le présent document qui décrit un algorithme pour dessiner un graphe planaire (le code LaTeXest sur le Github).
2. L'implémentation de cet algorithme qu'est disponible sur GitHub à l'adresse suivante : https://github.com/Kazzoul-Youness/StraightLine-ShiftMethod.

Table des matières

Résumé

La représentation en ligne droite des graphes planaires joue un rôle essentiel dans de nombreux domaines tels que les sciences humaines, la médecine, la topologie, l'informatique et bien d'autres encore. L'algorithme de Shift method, développé par Fraysseix, Pach et Pollack, offre une approche efficace pour créer des représentations visuelles claires et ordonnées de ces graphes. Cependant, cette méthode n'a pas été suffisamment étudiée et ses implications restent peu explorées.

Dans cette étude, l'accent est mis sur une exploration détaillée de la Shift method et de son utilisation pour représenter des graphes planaires en ligne droite. L'ordre canonique des sommets, qui joue un rôle crucial dans l'algorithme, est introduit et appliqué pour générer des représentations en ligne droite qui préservent la structure et la connectivité du graphe d'origine.

Les étapes clés de l'algorithme de Shift method sont décrites, mettant en évidence les techniques utilisées pour organiser les sommets de manière linéaire et créer des tracés optimaux. Les avantages et les limites de cette méthode sont discutés, en soulignant ses applications pratiques dans la visualisation et l'analyse de graphes planaires.

Cette étude approfondie de la méthode Shift method vise à améliorer notre compréhension de la représentation en ligne droite des graphes planaires. En mettant en évidence ses fondements théoriques et ses techniques pratiques, elle ouvre de nouvelles perspectives intéressantes pour la visualisation et l'analyse de ces graphes dans divers domaines d'application. Nous espérons que ce travail encouragera la poursuite des recherches dans ce domaine et favorisera l'adoption de l'algorithme de Shift method en tant qu'outil essentiel pour la représentation en ligne droite des graphes planaires.

4 État de l'art

Le but derrière les dessins en ligne droite des graphes planaires est de bien représenter la visibilité des données avec les liens ou les relations de façon lucide et déterminée. Les graphes planaires sont généralement utilisés dans divers domaines, quasiment toutes les sciences humains, politiques, médecines, la topologie, géométrie combinatoire. . .,
En utilisant des lignes droites pour relier les points de données, il est possible de créer des représentations visuelles telles que les cartes de transport (métro, routier, ferroviaire, maritime et aérien), ainsi que des graphiques représentant des processus métaboliques, atomiques, et autres. Ces schémas permettent d'analyser le dynamisme et l'efficacité des modèles étudiés, ou de mettre en évidence des incohérences dans les données. Cette compréhension approfondie est cruciale pour aborder les problèmes de manière adéquate, prendre des décisions éclairées et résoudre des situations complexes.

Pour créer une représentation en ligne droite d'un graphe planaire, diverses méthodes existent. Atteindre une qualité optimale, c'est-à-dire la meilleure représentation possible qui soit claire, précise, facile à comprendre, et sans chevauchements ni enchevêtrements de lignes, nécessite le respect de certaines conditions. Des algorithmes permettent de réaliser ces représentations en temps linéaire en utilisant la méthode de shift, tout en assurant la qualité optimale en respectant les conditions énoncées <Kindermann, 2010>. La méthode de shift, développée par de Fraysseix, Pach et Pollack <De Fraysseix, Pach et Pollack, 1990>, est une approche efficace pour construire des dessins planaires en ligne droite sur grille, où les arêtes ont des coordonnées entières. Elle permet de créer des représentations de graphes planaires avec une hauteur $\theta(n)$ [1] et une largeur $\theta(n)$, ce qui résulte en une superficie de $\theta(n^2)$. La complexité temporelle de cette méthode est linéaire, c'est-à-dire $\theta(n)$, où n représente le nombre de sommets du graphe planaire. L'algorithme de la méthode de shift se concentre sur la création de représentations de graphes planaires qui sont visuellement claires et faciles à comprendre, tout en évitant les chevauchements et les enchevêtrements de lignes <Vismara, 2013>.

L'algorithme de shift method est particulièrement utile pour les applications pratiques, car il permet d'obtenir des dessins de graphes planaires de manière efficace et avec une superficie minimale. Cette méthode utilise une approche itérative pour déterminer la position des nœuds du graphe sur un axe horizontal, en utilisant une technique de décalage pour résoudre les conflits entre les nœuds. Il est important de noter que les améliorations apportées à l'algorithme de shift method ont permis de résoudre certains problèmes liés à la représentation en ligne droite des graphes planaires. En particulier, la surface minimale nécessaire pour obtenir un dessin en ligne droite est un élément crucial pour la visualisation et l'analyse des relations entre les nœuds du graphe <N. Takao, R. Saidur, 2017>.

Plusieurs améliorations et variantes de l'algorithme de shift method ont été proposées dans

1. La complexité temporelle de la méthode de shift utilisée pour créer les représentations de graphes planaires est $\theta(n)$. Cela signifie que la hauteur, la largeur et la superficie de ces représentations augmentent proportionnellement au nombre de sommets du graphe. En d'autres termes, plus le nombre de sommets augmente, plus ces mesures augmentent de manière linéaire. Ainsi, $\theta(n)$ indique que la croissance des mesures de taille et de la complexité temporelle est linéaire par rapport au nombre de sommets du graphe planaire.

la littérature, permettant de réduire la surface minimale nécessaire pour obtenir un dessin en ligne droite tout en maintenant un temps de calcul linéaire en $\theta(n)$. Ces avancées ont renforcé la position de l'algorithme de shift method comme l'une des méthodes les plus efficaces pour obtenir des représentations en ligne droite de graphes planaires. En conclusion, la méthode de shift et ses améliorations successives offrent des solutions performantes pour la visualisation et l'analyse de graphes planaires dans diverses applications pratiques <Kindermann, 2010>.

5 Introduction

Les graphes planaires occupent une place centrale dans diverses disciplines allant des mathématiques à l'informatique, avec une portée qui s'étend bien au-delà. Que ce soit pour l'optimisation des réseaux, la modélisation des structures chimiques, la conception de circuits imprimés, ressources humains, la cartographie et la planification urbaine, voire l'étude des métabolismes humains en médecine, les graphes planaires sont omniprésents. Ils représentent une manière élégante et efficace de décrire des relations complexes. Leur dessin, en particulier lorsqu'il est réalisé en ligne droite sans croisement d'arêtes, est non seulement esthétiquement plaisant, mais aussi d'une grande utilité pratique. C'est cette richesse d'applications et ce défi de création qui m'ont poussé à choisir le sujet des graphes planaires pour mon travail de recherche. Plus précisément, l'objectif de ce mémoire est d'étudier et d'implémenter la méthode de décalage, ou **Shift method**, pour **la représentation en ligne droite des graphes planaires**, tout en soulignant leur importance fondamentale dans une multitude de domaines. Un dessin en ligne droite d'un graphe planaire est un dessin dans lequel chaque arête est dessinée comme un segment de ligne droite sans croisement d'arêtes, comme illustré à la figure suivante :

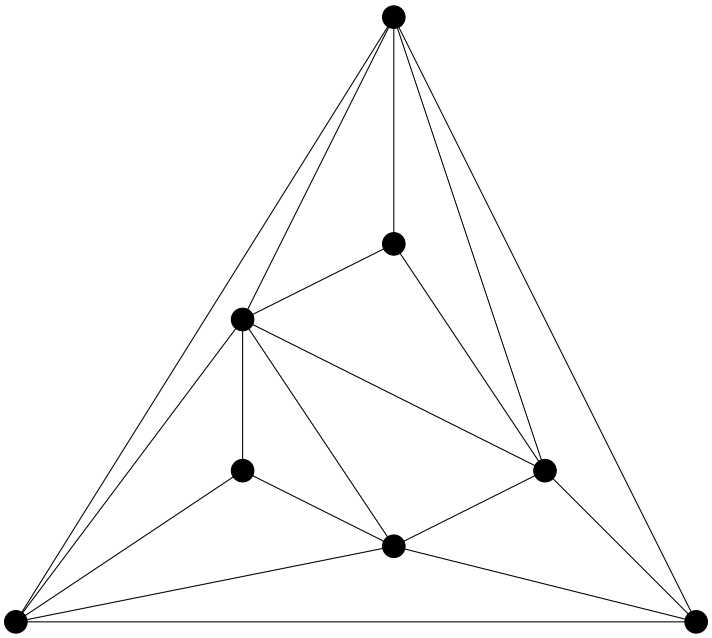


FIGURE 1 – Dessin en ligne droite d'un graphe planaire

Wagner <Wagner, 1936>, Fary <Fary, 1948> et Stein <Stein, 1951> ont indépendamment prouvé que tout graphe planaire G possède un dessin en ligne droite. Leurs preuves donnent immédiatement des algorithmes en temps polynomial pour trouver un dessin en ligne droite d'un graphe planaire donné. Cependant, l'air d'un rectangle entourant un dessin sur une grille d'entiers obtenu par ces algorithmes n'est pas limitée par un polynôme dans le nombre de$n$sommets de G. En fait, l'obtention d'un dessin dont l'aire est limitée par un polynôme est restée un problème ouvert pendant longtemps. En 1990, de Fraysseix et al <De Fraysseix, Pach et Pollack, 1990>. et Schnyder <W. Schnyder, 1990> ont montré par deux

méthodes différentes que tout graphe planaire de $n > 3$ sommets possède un dessin en ligne droite sur une grille d'entiers de taille $(2n-4)\times(n-2)$ et $(n-2)\times(n-2)$, respectivement<N. Takao, R. Saidur, 2017>. Les deux méthodes peuvent être mises en œuvre sous forme d'algorithmes en temps linéaire, et sont bien connues sous le nom de "**Shift method**" et la "**realizer method**", respectivement.

Dans ce travail on va voir la Shift method ou méthode de décalage à la section 8 <N. Takao, R. Saidur, 2017>. De Fraysseix et al. ont montré que, pour chaque $n > 3$, il existe un graphe planair de$n$sommets, *e.g.,* des triangles imbriqués, qui nécessite une grille de taille au moins égale à $\frac{2}{3}(n-1)\times\frac{2}{3}(n-1)$ pour tout tracé de grille <M. Chrobak and S. Nakano, 1998> <De Fraysseix, Pach et Pollack, 1990>. Il a été conjecturé que tout graphe plan de$n$sommets possède un dessin sur une grille de $\frac{2}{3}(n) \times \frac{2}{3}(n)$, mais il s'agit encore d'un problème ouvert. D'autre part, une classe restreinte de graphes possède une grille plus compacte. *e.g.,*, si G est un graphe plan à 4 connexions, alors G a un dessin de grille plus compact<N. Takao, R. Saidur, 2017>.

À la section 11 nous décrivons une preuve constructive du théorème de de Fraysseix et al <De Fraysseix, Pach et Pollack, 1990> selon lequel tout graphe plan G de $n > 3$ sommets possède une grille de lignes droites de taille $(2n-4)\times(n-2)$, et nous présentons une implémentation en temps linéaire d'un algorithme permettant de trouver un tel dessin <M. Chrobak and T. H. Payne, 1995>. Si G n'est pas triangulé, nous obtenons un graphe plan triangulé G' à partir de G en ajoutant des arêtes fictives à G. À partir d'un dessin de grille de lignes droites de G', nous pouvons immédiatement obtenir un dessin de grille de lignes droites de G en supprimant les arêtes fictives. Par conséquent, il suffit de prouver qu'un graphe plan triangulé G de$n$sommets possède une grille de lignes droites de taille $(2n-4)\times(n-2)$. Pour construire un tel dessin, de Fraysseix et al. ont introduit un ordre des sommets appelé "**ordre canonique**" et ont installé les sommets un par un dans le dessin en fonction de l'ordre <N. Takao, R. Saidur, 2017>.

Dans la section 7 nous présentons un ordre canonique, dans la section 9 nous appliquons Shift method, et dans la section 10 nous allons voir l'algorithme Shift method. Nous nous présentons une implémentation en langage Java de l'algorithme dans la section 12.

6 Motivation

Les graphes planaires ont de nombreuses applications pratiques dans divers domaines tels que l'informatique, les mathématiques, la physique, la biologie, etc. Voici quelques exemples d'utilité des graphes planaires :

- **Algorithmes** : Les graphes planaires sont utilisés dans la conception d'algorithmes pour résoudre divers problèmes tels que la recherche de chemins les plus courts, la coloration de graphes, la planification de tournées, peuvent tous être appliqués à des graphes planaires.

- **Conception de circuits imprimés** : Les graphes planaires sont utilisés pour modéliser la disposition des composants électroniques et les connexions entre eux dans la conception de circuits imprimés <Freeman, R. L., 1991>. Les composants peuvent être représentés par des sommets et les connexions par des arêtes.

- **Cartographie et planification urbaine** : Les graphes planaires sont utilisés pour représenter les réseaux de transport, les plans de ville et les cartes routières, ce qui permet de visualiser les connexions et les distances entre les différents points d'intérêt <Di Battista, G., Tamassia, R., 1989>.

- **Modélisation moléculaire** : Les graphes planaires sont utilisés pour modéliser la structure moléculaire des composés chimiques, où les nœuds représentent les atomes et les arêtes représentent les liaisons entre les atomes <Todeschini, R., Consonni, V., 2009>.

- **Réseaux informatiques** : Les graphes planaires sont utilisés pour modéliser les réseaux de communication, ce qui permet de visualiser les connexions entre les différents nœuds et de trouver les chemins les plus courts pour acheminer les données <Wang, 2008>.

- **Reconnaissance de formes** : Les graphes planaires sont utilisés pour représenter des données dans des domaines tels que la reconnaissance de formes et l'analyse d'image <Balakrishnan, A., Ranganathan, K., 2018>...*etc*

En résumé, les graphes planaires sont des outils utiles pour modéliser et résoudre des problèmes dans de nombreux domaines, car ils permettent de représenter visuellement les connexions et les relations entre les différents éléments d'un système.

7 Terminologie

7.1 Graphe

Un graphe est une structure mathématique qui représente deux ensembles : un ensemble de points appelés **sommets** ou **nœuds**, et un ensemble d'**arêtes**. Les arêtes représentent des relations qui relient certains ou tous ces sommets entre eux. *e.g.*,, dans le cas des réseaux sociaux, une arête peut indiquer qu'il y a une relation d'amitié entre deux personnes. Si aucune arête n'existe entre deux sommets, cela signifie qu'il n'y a pas de relation entre eux.

Ces relations peuvent également représenter des liens entre des atomes, des objets, des entités ou des événements, entre autres. Les graphes sont utilisés dans de nombreux domaines tels que les mathématiques, l'informatique, les sciences humaines, sociales, médicales, électroniques et bien d'autres.

Un graphe est défini comme un ensemble de sommets et d'arêtes. En notant V l'ensemble des sommets et E l'ensemble des arêtes, un graphe G peut donc s'écrire sous la forme :

$G = (V, E)$, où chaque élément de E est une paire de $\{v_1, v_2\}$ avec $v_1 \neq v_2$ et $v_1, v_2 \in V$.

7.2 Sommet

Un ensemble appelés noeuds ou sommets sont les éléments du graphe, dans notre exemple figure 2 A, B et C sont les sommets de ce graphe.

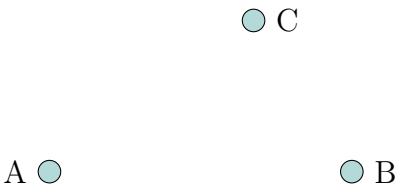


FIGURE 2 – Graphe sans arêtes, nommé stable

7.3 Arête

Un ensemble d'arêtes représente les relations entre certains des sommets d'un graphe <Laforest, 2017>. Ces relations peuvent prendre la forme de segments de droites ou de **courbes simples**, comme illustré dans la figure 3. Le dessin d'un graphe n'est rien d'autre que la représentation visuelle de ces relations entre les différents éléments du graphe.

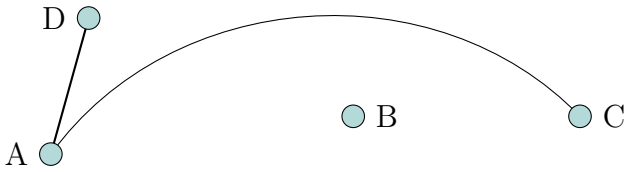


FIGURE 3 – Graphe avec une courbe simple

7.4 Courbe simple

Par courbe simple, on entend un tracé continu sans point double. Des arêtes peuvent se croiser, un sommet n'étant pas indiqué à l'intersection. Mais cela peut être ambigu si le graphe a une allure "exotique" comme la figure 4, l'arête (1,6) coupe l'arête (1,4) par trois fois.

Exemple 1

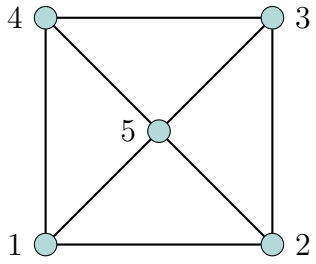


FIGURE 4 – Exemple d'un graphe licite

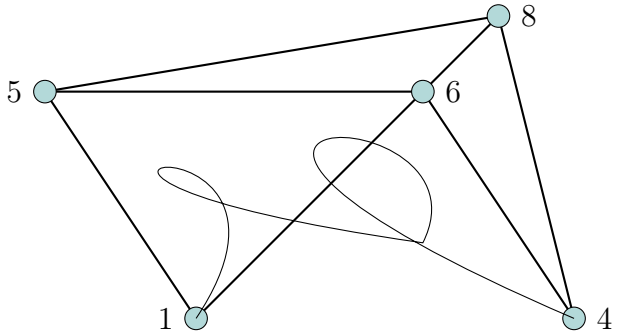


FIGURE 5 – Exemple d'un graphe illicite, avec une arête (1,4) illégale

Le premier graphe 4 entrain une situation licites, mais le deuxième une situation illicites, car l'arête (1,4) possède deux points doubles, c'est illégal ! <Laforest, 2017>

Exemple 2

Dans l'exemple suivant figure 6 on a les sommets : 1,2,3, et 4 et on a des arrêts : $\{1,3\}, \{2,4\}, \{2,3\}$ et $\{3,4\}$.

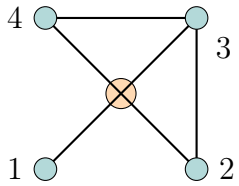


FIGURE 6 – Exemple 1 d'un graphe avec courbe simple

Et on peut le dessiner autrement, *e.g.,* voir la figure 7 <Laforest, 2017>.

Exemple 3

Le dessin suivant représente le même graphe, mais avec les sommets positionnés d'une autre façon différente dans le plan. En conséquence, l'intersection entre les arêtes {1,3} et {2,4} n'est plus présente, comme le montre la figure suivante 7 <Laforest, 2017>.

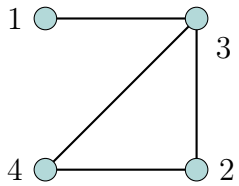


FIGURE 7 – Exemple 2 d'un graphe avec courbe simple

Dans un graphe comme celui-ci, avec cette intersection entre les arêtes {1,3} et {2,4} l'essentiel est de bien déterminer qu'il y a effectivement les quatre arêtes qui sont représentées, que ce soit sur la figure 6 ou sur la figure 7. Il est important que le dessin du graphe ne prête pas à confusion et qu'il n'y ait pas d'ambiguïté quant à la présence ou l'absence de certaines arêtes ou de certains sommets <Laforest, 2017>.

Exemple 4

Ci-dessous une autre représentation graphique du même graphe, potentiellement plus originale, comme illustré dans la figure suivante <Laforest, 2017>.

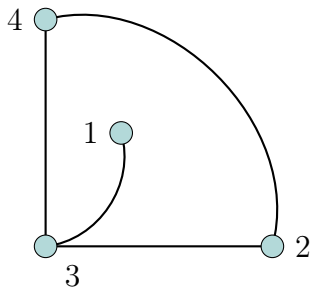


FIGURE 8 – Exemple 3 d'un graphe avec courbe simple

Il s'agit toujours du même graphe, même si les arêtes sont représentées de manière un peu plus tordue. Cela n'a pas d'importance, car ce qui compte avant tout, c'est de savoir quelles sont les relations entre les différents sommets, comme le montre la figure 8 <Laforest, 2017>.

7.5 Ordre

L'ordre d'un graphe est défini comme étant le nombre de sommets qui le composent <B Portie, J-M Menyr>. La disposition spatiale des sommets ainsi que la forme des arêtes ne sont pas des paramètres pertinents pour le calcul de l'ordre d'un graphe.

7.6 Voisin

À partir des deux notions de base vu avant : les **sommets** qui sont les éléments et les **arêtes** qui sont les relations, on va définir d'autres vocabulaires, qui seront bien utiles, sont les notions de **voisin** et de **degré**, <Laforest, 2017>.
On dit qu'un sommet A est un voisin d'un sommet B dans un graphe si et seulement si les deux sommets sont reliés par une arête, notée $\{A, B\}$ ou $\{B, A\}$, conformément à la définition mathématique de la relation de voisinage entre deux sommets.

7.7 Degré

Le degré d'un sommet v dans un graphe G, noté $d_G(v)$, est défini comme étant le nombre d'arêtes de G qui sont incidentes avec ce sommet. Autrement dit, le degré d'un sommet correspond simplement au nombre de ses voisins. Dans le cas particulier d'un graphe simple, le degré d'un sommet représente donc le nombre de voisins de ce sommet. Enfin, un sommet de degré zéro est appelé sommet isolé <J.A. Bondy and U.S.R. Murty, 2008>.
Exemples :

- **Exemple 2 :**

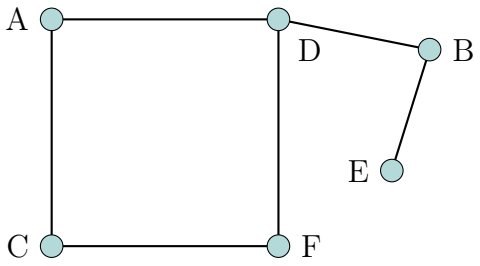


FIGURE 9 – Exemple différents cas de voisinage

Sur le graphe de la figure 9, on dit que :
— les voisins de A, sont D et C puisqu'il a une Arête AD et une arête AC ;

— E et F ne sont pas voisins malgré le fait qu'ils sont proches sur le dessin, ils ne sont pas voisins parce qu'ils ne sont pas reliés par une arête ;

— E il a un seul voisin qu'est B, du coup son degré, c'est 1. <J.A. Bondy and U.S.R. Murty, 2008> ;

— et puis un autre exemple sommet D est de degré 3, puisqu'il a trois voisins qui sont à B, F et A <Laforest, 2017>.

7.8 Chaîne, longueur d'un chemin

Une chaîne : Dans un graphe, une chaîne est définie comme une suite d'arêtes consécutives, représentant une sorte de promenade le long du graphe. Elle est généralement identifiée par la liste ordonnée des sommets qu'elle relie. Le terme "chaîne" est couramment utilisé en théorie des graphes pour désigner ce type de parcours, bien que les termes "chemin" ou "parcours" soient également courants <Mehl>.

Une **chaîne simple** est une **chaîne** qui ne passe pas deux fois par la même arête.

e.g., : Prenant deux sommets A et B dans ce graphe, et le chemin donc c'est une suite d'arêtes qui permet de relier A à B.
La **longueur d'un chemin** est son nombre d'Arêtes.

Exemple :
Le chemin vert représenté sur la figure 10 est un exemple de chemin dans le graphe considéré, qui est composé de quatre arêtes consécutives. On dit alors que la longueur de ce chemin est égale à 4. Il est important de souligner que ce n'est pas le seul chemin existant dans le graphe : il existe en effet de nombreux autres chemins possibles, avec des longueurs allant de 3 à 8 arêtes.

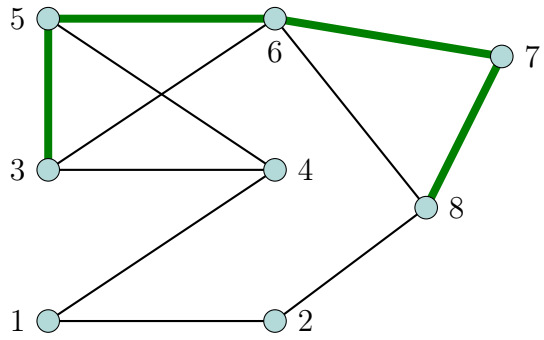


FIGURE 10 – Exemple 1 d'un chemin de longueur 4

7.9 Cycle, longeur d'un cycle

Un cycle est un **chemin/chaîne** dans les deux extrémités sont reliées [2], ou qui commence et se termine au même sommet. [4]
Et la **longueur d'un cycle** est la somme des Arêtes, regardons des exemples :

- **Exemple :**

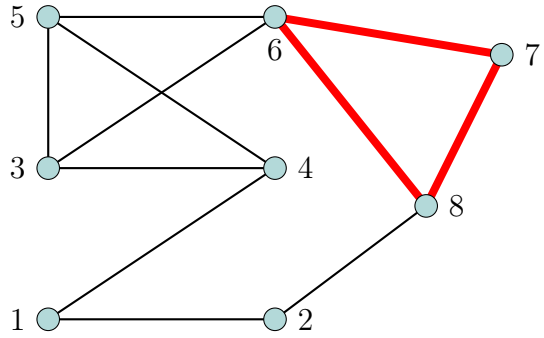


FIGURE 11 – Exemple 1 d'un cycle de longueur 3

Ce cycle rouge est de longueur=3, la longueur d'un cycle, c'est le nombre de ces Arêtes.

7.10 Connexe

Définition : Un graphe $G = (V, E)$ est connexe si pour tout $A, B \in V$ il existe un chemin entre A et B.
Donc on dit si pour chaque paire de sommet, le graphe contient un **chemin** entre ces deux sommets, alors le **graphe est connexes** <Laforest, 2017>.
i.e. il ne doit pas y avoir de sommet isolé, nous allons voir après un exemple dans la section Graphe 2-connexe 8.2.

Exemples :

▶ Exemples 1

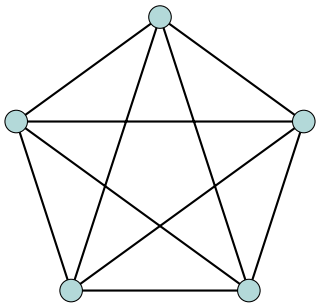


FIGURE 12 – Exemple 1 d'un graphe connexe

Ce graphe nomé K_5 est un graphe complet non orienté de cinq sommets, c'est-à-dire qu'il contient toutes les arêtes possibles entre ses sommets. Par conséquent, chaque sommet de K_5 est adjacent à tous les autres sommets, ce qui signifie qu'il existe un chemin de longueur 1 entre chaque paire de sommets. En d'autres termes, le graphe K_5 est connexe et cette propriété est assurée par sa structure particulière en tant que graphe complet.

▶ Exemples 3

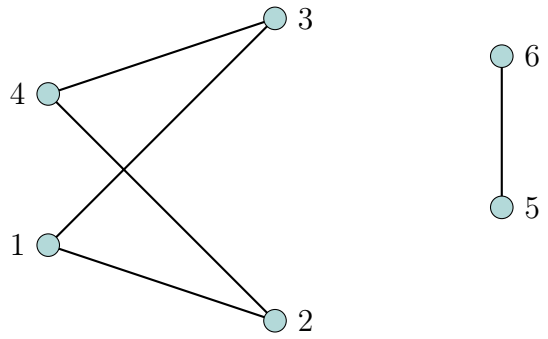


FIGURE 13 – Exemple 2 d'un graphe non-connexe

En revanche, il est intéressant de considérer l'exemple de graphe suivant pour illustrer une propriété importante des graphes : la connexité. Ce graphe est composé de deux morceaux séparés, ce qui le rend non connexe. En effet, il n'existe pas de chemin reliant les sommets 3 et 6, ni entre les sommets 4 et 5 *etc.*, ce qui empêche de se déplacer de l'un à l'autre de ces sommets en suivant les arêtes du graphe.

7.11 Graphes planaires

Définition d'un graphe planaire.
En théorie des graphes, un graphe planaire est un graphe qui peut être incorporé dans le plan, c'est-à-dire qu'il peut être dessiné sur le plan de telle sorte que ses arêtes ne se croisent qu'à leurs extrémités. En d'autres termes, il peut être dessiné de manière à ce qu'aucune arête ne croise une autre. Un tel dessin est appelé graphe planaire ou plongement. Un graphe planaire peut être défini comme un graphe planaire dont chaque nœud est relié à un point d'un plan et chaque arête à une courbe planaire de ce plan, de sorte que les points extrêmes de chaque courbe sont les points reliés à ses nœuds d'extrémité et que toutes les courbes sont disjointes, sauf en leurs points extrêmes.

Exemple : Le graphe figure suivante 14 est planaire si on déplace les sommets <M. Sablik>.

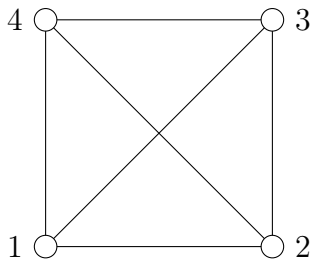


FIGURE 14 – Exemple d'un graphe planaire

Dans cette image, les chemins $< 1, 4, 3 >$ et $< 1, 2, 3 >$ sont distincts, même s'ils partagent les sommets 1 et 3.

7.12 Face d'un graphe planaire

En théorie des graphes, on appelle "face" d'un graphe G planaire tout cycle de G de longueur au moins égale à 3 ne contenant en son intérieur aucun sommet ni arête. Par convention, la frontière extérieure d'un graphe connexe planaire est considérée comme une face, appelée "face extérieure" ou "face infinie" (au sens de non limitée). Cette notion de face est essentielle en théorie des graphes planaires, où elle permet notamment de décrire la structure des graphes planaires et de caractériser leur planarité ou non-planarité à partir de leur nombre de faces et de sommets.

Exemple :

Le graphe ci-dessous possède 5 faces finies : la face 1 adf, la face 2 afb, la face 3 bfc, la face 4 cfd et la face cbe. la face abcd n'en est pas une. Ce graphe connexe contient donc 5

faces si on lui adjoint sa face 6 extérieure.

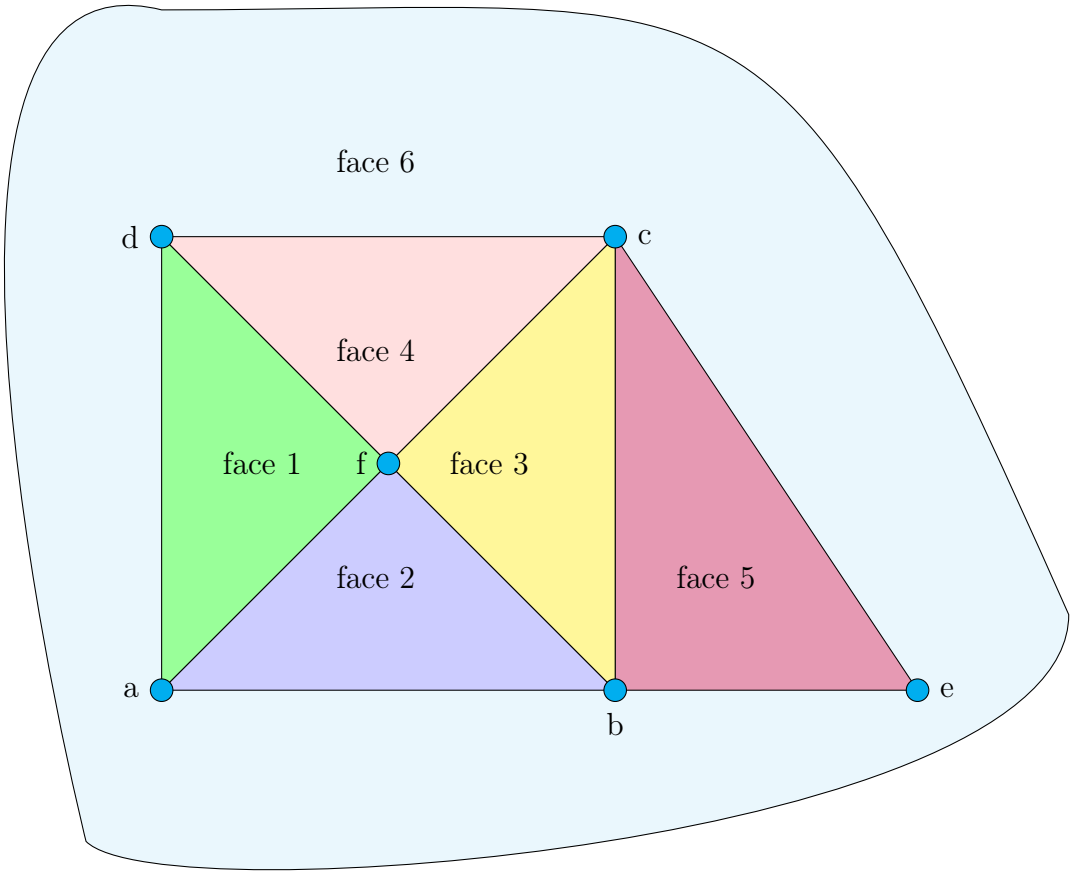


FIGURE 15 – Les faces d'un graphe

<Mehl>

7.13 Frontière

En théorie des graphes planaires, la **frontière** d'une face dans un graphe planaire est le "contour" qui délimite cette face, c'est-à-dire l'ensemble des arêtes qui se trouvent sur le bord de la face. Dans le cas particulier des graphes planaires dessinés avec des arêtes droites, la frontière d'une face correspond au tracé rectiligne qui relie les sommets de la face. La notion de frontière est une notion essentielle en théorie des graphes planaires, car elle permet de caractériser la structure des faces qui composent le graphe planaire, ainsi que leur relation avec les autres faces du graphe. La frontière est également utilisée pour décrire la propriété

de "faces adjacentes" dans un graphe planaire, qui correspond au fait que deux faces ont une partie de leur frontière en commun.

7.14 Corde

Une corde, dans le contexte de la théorie des graphes, est une arête qui relie deux sommets non adjacents d'un cycle. Autrement dit, une corde est une arête supplémentaire qui connecte deux sommets d'un cycle sans passer par les sommets intermédiaires du cycle.
Nous verrons plus tard dans la section du Graphe cordal 8.4, un exemples interessant. [2]

8 Types de graphes

8.1 Sous-graphes et sur-graphes

Un sous-graphe est un graphe qui est obtenu en retirant des sommets et/ou des arêtes d'un graphe initial, tout en conservant les relations entre les sommets restants. Plus précisément, un sous-graphe est **induit par** [3] un ensemble de sommets sélectionnés à partir du graphe initial, ce qui signifie que seuls les sommets choisis et les arêtes qui les relient sont inclus dans le sous-graphe. Ainsi, le sous-graphe conserve la structure de connexion entre ses sommets, mais peut avoir moins de sommets et d'arêtes que le graphe initial.

Étant donné un graphe G. Si e est une arête de G, on peut obtenir un graphe à $m-1$ arêtes en supprimant e de G, mais en laissant les sommets et les autres arêtes intacts. Le graphe ainsi obtenu est noté $G' = G - e$. De manière similaire, si v est un sommet de G, on peut obtenir un graphe à $n-1$ sommets en supprimant de G le sommet v ainsi que toutes les arêtes incidentes à v. Le graphe ainsi obtenu est noté $G - v$, comme illustré dans les deux figures suivates <J.A. Bondy and U.S.R. Murty, 2008>.

2. Introduc graphes `Théorie des graphes sur Wiki`
3. La notion de "induit par" est utilisée pour spécifier qu'un sous-graphe est construit en sélectionnant un ensemble spécifique de sommets et/ou d'arêtes à partir d'un graphe initial.

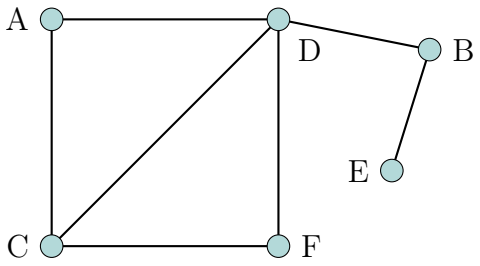


FIGURE 16 – Exemple d'un sur-graphe

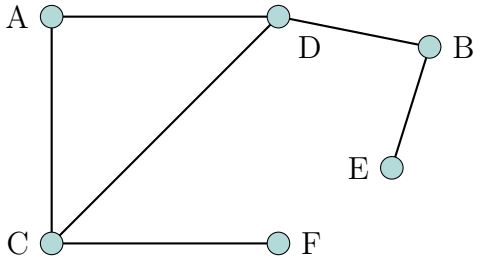


FIGURE 17 – Exemple d'un sous-graphes

8.2 Graphe 2-connexe

Définition :
Un graphe est dit "2-connexe" ou "biconnexe" s'il existe au moins deux chemins complètement distincts entre chaque paire de sommets (c'est-à-dire qu'aucun sommet n'est partagé entre les chemins). Cela implique que si on retire un sommet (et toutes les arêtes qui lui sont connectées), le graphe reste toujours connexe<West, Douglas B., 2001>. En d'autres termes, la suppression de n'importe quel sommet ou n'importe quelle arête permet au graphe de rester connexe, sans le séparer en plusieurs parties non connectées<Belbèze, Christian, 2012>.

Theorem 8.1. *Un graphe G est 2-connexe si et seulement si, pour toute paire de sommets de G, il existe un cycle dans G qui contient ces deux sommets.*

8.3 Graphe 3-connexe

Un graphe 3-connexe est un graphe non orienté dans lequel il existe au moins trois chemins internes distincts entre chaque paire de sommets. Autrement dit, pour chaque paire de sommets, il existe au moins trois chemins différents reliant ces deux sommets qui ne partagent aucun sommet commun autre que les deux extrémités <J.A. Bondy and U.S.R. Murty, 2008>, on dit : Le graphe $G = (V, E)$ est 3-connexe si pour tout $v_1, v_2 \in V$, le graphe **induit par** [3] $V \setminus \{v_1, v_2\}$ est connexe.

8.4 Graphe cordal

En théorie des graphes, on dit qu'un **graphe est cordal** figure 18 si chacun de ses cycles de quatre sommets ou plus possède une **corde**, c'est-à-dire une arête reliant deux sommets non adjacents du cycle. Une définition équivalente est que tout cycle sans corde possède au plus trois sommets.

Exemple

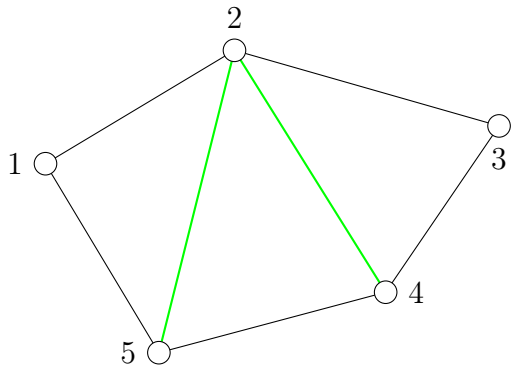


FIGURE 18 – Graphe cordal

Un cycle, en noir, avec deux cordes, en vert. Si l'on s'en tient à cette partie, le graphe est cordal. Supprimer l'une des arêtes vertes *e.g.* {2,5} rendrait le graphe non cordal.
En effet, l'arête {2,4} formerait, avec les trois arêtes noires {1,2}, {1,5} et {5,4}, un cycle de longueur 4 sans corde [4].
Les **graphes complets** et les **arbres** sont des exemples simples de graphes cordaux. <J.A. Bondy and U.S.R. Murty, 2008>.

8.5 Graphe planaire triangulé

Examinons l'exemple présenté dans la figure ci-dessous :

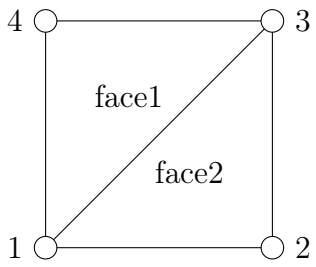


FIGURE 19 – Graphe non-triangulé

4. Graphe cordal sur Wikipedia

Nous avons un graphe non triangulé, car la face extérieure contient 4 arêtes. Comment pouvons-nous alors effectuer une triangulation pour ce graphe ? En ajoutant une arête entre les sommets (1) et (3), nous obtenons le graphe modifié ci-dessous :

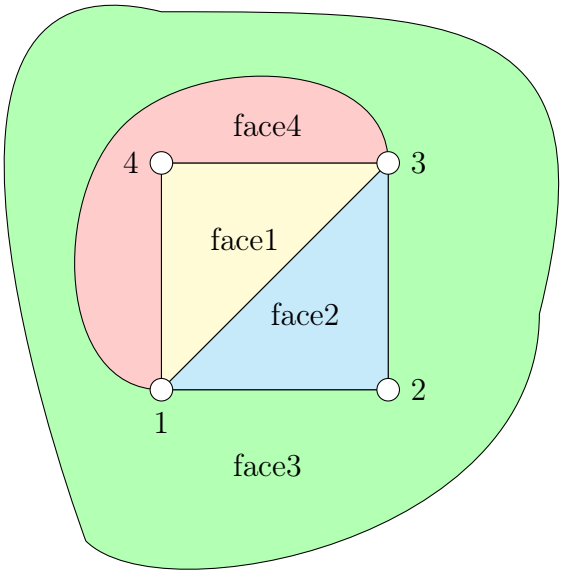


FIGURE 20 – Graphe triangulé

Et maintenant, nous avons notre triangulation, également appelée **graphe planaire avec triangulation interne** <J.A. Bondy and U.S.R. Murty, 2008>.

Definition
Un graphe triangulé est un graphe planaire, simple, connexe, non orienté, où toutes ses faces sont des triangles <J.A. Bondy and U.S.R. Murty, 2008>. Dans la figure précédente, nous voyons que les faces (4) et face3 sont des triangles, car elles contiennent chacune trois arêtes, mais elles n'ont pas la forme d'un triangle classique, comme illustré dans la figure suivante :

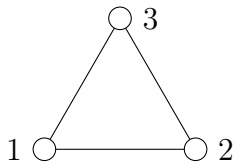


FIGURE 21 – Triangle

8.6 Mineurs de graphes

Un mineur de graphe est un sous-graphe obtenu à partir d'un graphe d'origine en supprimant des arêtes et/ou des nœuds. En d'autres termes, un graphe G' est un mineur du

graphe G si G' peut être obtenu à partir de G en supprimant des arêtes et/ou des nœuds, et en effectuant des contractions d'arêtes.
La contraction d'une arête fait référence à une opération qui consiste à fusionner les deux extrémités d'une arête en un seul nœud, tout en maintenant les arêtes adjacentes à l'arête contractée. Plus précisément, si nous avons un graphe planaire et que nous effectuons la contraction d'une arête, nous supprimons cette arête du graphe et fusionnons les nœuds aux extrémités de l'arête en un seul nœud. Les arêtes adjacentes à l'arête contractée sont préservées et continuent de se connecter au nouveau nœud résultant de la contraction.
Les mineurs de graphes sont largement étudiés en théorie des graphes, en informatique théorique et en mathématiques discrètes, car ils ont de nombreuses applications dans différents domaines, tels que la théorie des réseaux, l'optimisation combinatoire, l'algorithmique et la complexité Jungnickel <2003> R. Diestel <2005>.

Et Whitney <1933> a prouvé que la suppression ou la contraction d'une arête d'un graphe planaire produit toujours un autre graphe planaire. Plus précisément, il a montré que tout graphe planaire est "mineur" d'un autre graphe planaire obtenu par suppression ou contraction d'arêtes.
Cette affirmation est maintenant considérée comme un résultat fondamental de la théorie des graphes planaires et est souvent utilisée dans la preuve d'autres résultats importants dans ce domaine.

Theorem 8.2. ***Théorème de Whitney*** *La suppression ou la contraction d'une arête d'un graphe planaire donne un autre graphe planaire.*

Ainsi nous avons :

Proposition 1. *Les mineurs de graphes planaires sont planaires.*

8.7 Graphes planaires maximaux

Un graphe simple est dit planaire maximal s'il est planaire mais que l'ajout d'une arête (sur l'ensemble de sommets donné) détruirait cette propriété (planarité). Toutes les faces (y compris la face extérieure) sont alors délimitées par trois arêtes, ce qui explique le terme alternatif de triangulation planaire. Les noms alternatifs "graphe triangulaire" <Schnyder, Walter, 1989> ou "graphe triangulé"<Bhasker, Jayaram and Sahni, Sartaj, 1988> ont également été utilisés, mais ils sont ambigus, car ils font plus communément référence au graphe linéaire d'un graphe complet et aux graphes chordaux respectivement.
Tout graphe planaire maximal est au moins 3-connexe. Si un graphe planaire maximal a v sommets avec $v > 2$, alors il a précisément $(3v - 6)$ arêtes et $(2v - 4)$ faces, conformément au **théorème d'Euler** pour les graphes planaires. Ce résultat peut être illustré en utilisant un exemple concret.
Par exemple, si nous considérons le graphe planaire maximal illustré dans la figure 20 avec $v = 4$ sommets, nous pouvons appliquer la formule pour calculer le nombre d'arêtes et de

faces. Ainsi, nous avons $3v - 6 = 3x4 - 6 = 6$ arêtes et $2v - 4 = 2x4 - 4 = 4$ faces. Cette correspondance entre le nombre de sommets, d'arêtes et de faces est vérifiée dans cet exemple spécifique, ce qui confirme la validité de la formule.

9 Dessins en lignes droites des graphe planaires

Il y a beaucoup d'études par des chercheurs sur ce sujet, on a *e.g.,* l'article :

> By far the most agreed-upon edge placement heuristic is to minimize the number of edge crossings in a graph [BMRW98,Har98,DH96,Pur02,TR05,TBB88]. The importance of avoiding edge crossings has also been extensively validated in terms of user preference and performance (see Section 4). Similarly, based on perceptual principles, it is beneficial to minimize the number of edge bends within a graph [Pur02, TR05, TBB88]. Edge bends make edges more difficult to follow because an edge with a sharp bend is more likely to be perceived as two separate objects. This leads to the heuristic of keeping edge bends uniform with respect to the bend's position on the edge and its angle [TR05]. If an edge must be bent to satisfy other aesthetic criteria, the angle of the bend should be as little as possible, and the bend placement should evenly divide the edge.

"The Aesthetics of Graph Visualization" <Bennett, Ryall, Spaltzerholz et Gooch, 2007, 3.2, page 8>.

Dans cet article ils disent que l'heuristique de placement des arêtes la plus reconnue consiste à minimiser le nombre de croisements des arêtes dans un graphe.
Et plus tard ils disent aussi qu'il est bénéfique de minimiser le nombre de courbes d'Arêtes [5], et si nous voulons minimiser quelque chose, dans le cas optimal nous n'avons rien de tout cela ! donc si nous n'avons pas de croisements d'Arêtes alors nous avons un dessin planaire, et si nous n'avons pas d'Arêtes alors nous avons un dessin en ligne droite. alors la question est de savoir **quelle est l'esthétique de nos dessins ?**, et pour cela, dans cette partie, nous nous concentrerons sur les graphes planaires.
Chaque fois qu'on regarde une classe de graphe, il y a quelques questions importantes qu'on doit poser, et les trois les plus importantes sont :

1. pouvons-nous caractériser cette classe ?
2. pouvons-nous reconnaître cette classe ?
3. comment pouvons-nous la dessiner ?

9.1 Caractérisation

Caractériser les graphes planaires implique d'identifier les propriétés distinctives qui les séparent des autres classes de graphes, en définissant les critères nécessaires pour qu'un graphe appartienne à cette catégorie. La caractérisation des graphes planaires repose sur le théorème de Kuratowski, selon lequel un graphe est planaire si et seulement si ni K_5 (graphe complet à 5 sommets) ni $K_{3,3}$ (graphe biparti complet avec deux ensembles de 3 sommets)

5. courbes d'Arêtes : où une arête change de direction dans une représentation graphique d'un graphe. Dans le contexte de la visualisation de graphes, les courbes d'arêtes peuvent rendre les arêtes plus difficiles à suivre, car ils peuvent donner l'impression que l'arête se compose de plusieurs segments distincts plutôt que d'une seule entité continue.

ne sont présents en tant que mineurs. Cette absence de deux graphes spécifiques en tant que mineurs constitue la base de la caractérisation des graphes planaires.

9.2 Reconnaissance

Étant donné un graphe à savoir si c'est un graphe planaire est appelé test de planarité. Plusieurs algorithmes ont été proposés pour ce problème. Les meilleurs atteignent une complexité en temps linéaire, ce qui est optimal asymptotiquement. Le premier tel algorithme date de 1974, et est dû à **John Hopcroft** et **Robert Tarjan** 1974, qui reconnaît un graphe planaire en temps linéaire. Donc dans le future on suppose que nous avons déjà vérifié si le graphe qu'on veut dessiner est planaire [6].

9.3 Dessin

Dans le domaine des graphes, différents algorithmes ont été développés pour traiter divers types de graphes. L'algorithme de **Tutte**, par exemple, est conçu spécifiquement pour les graphes planaires 3-connexes. D'autres algorithmes existent pour les arbres et les graphes en séries parallèles. Cependant, il n'existe pas d'algorithme universel pour tous les graphes planaires et leurs représentations.
Des théorèmes anciens de **Wagner** (1936), **Fary** (1948) et **Stein** (1951), prouvés indépendamment, affirment que chaque graphe planaire admet une représentation planaire avec des arêtes droites. Bien que cela permette de représenter tous les graphes planaires de cette manière, ces preuves présentent un inconvénient : la taille (ou le nombre d'arêtes) de la région entourée par les arêtes dans une représentation planaire n'est pas bornée par une fonction polynomiale de n, où$n$est le nombre de sommets du graphe. En conséquence, la taille de cette région peut croître au moins de manière exponentielle par rapport à n, ce qui peut rendre la représentation du graphe inefficace ou difficile à réaliser en pratique.

D'abord nous voulons une classe spéciale de **graphes planaires, appelés triangulations** 20, ou toutes les faces doivent être des triangles, et il y a une deuxième sous-classe qui sont les **graphes planaires maximaux** 8.7.

Extension d'un graphe planaire à une triangulation planaire :

Examinons la figure 20 de notre précédent exemple de graphe triangulé et essayons de trouver une arête que nous pouvons ajouter sans compromettre la planarité.
En réalité, nous ne le pouvons pas. Par exemple, si nous ajoutons une arête entre les sommets (2) et (4), cela créerait un croisement avec l'arête {1,3}. Si nous la faisons passer par le face externe, nous avons également un croisement avec l'arête externe {1,3}. Étant donné que chaque face est un triangle et que chaque arête que nous souhaitons ajouter ne peut être

6. Graphe planaire sur Wikipedia

dessinée de manière planaire que si elle se trouve entièrement à l'intérieur d'une face, nous ne pouvons pas ajouter d'arête arbitrairement, car dans un triangle, tous les sommets sont déjà voisins.
Ainsi, nous pouvons observer qu'un graphe planaire maximal 8.7 est exactement une triangulation planaire ; ces deux classes de graphes sont donc identiques.

En se concentrant sur les triangulations planaires, nous pouvons dessiner tous les graphes planaires, car chaque graphe planaire est un sous-graphe d'une triangulation planaire.

Supposons que nous ayons un graphe planaire et qu'il y ait une face trop grande, ayant plus de trois sommets. Comment pouvons-nous étendre ce graphe à un super-graphe qui est une triangulation planaire ? Il y a deux façons :

Réponse 1 :

Prenons un seul sommet, comme illustré dans la figure 22, et connectons-le à tous les sommets de cette face, à l'exception de ses voisins, car il est déjà connecté à eux. Ainsi, nous obtenons un super-graphe dont notre graphe original est le sous-graphe.

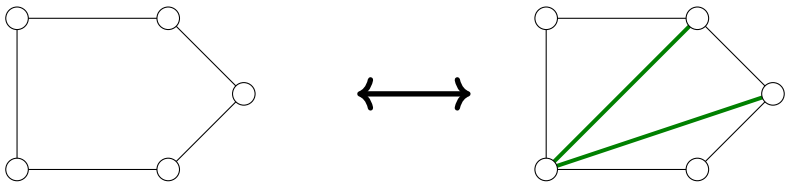


FIGURE 22 – Exemple 1, transformer un sous-graphe en super-graphe

Réponse 2 :

Ajouter un seul sommet dans chaque face et de le connecter à tous les sommets qui se trouvent sur la frontière 7.13 de la face cela fonctionne tant que le graphe d'entrée est déjà 2-connexe, donc toutes nos faces sont des cycles simples si elles ne sont pas des cycle simples, alors nous devons ajouter quelques arêtes supplémentaires mais nous pouvons encore le faire facilement

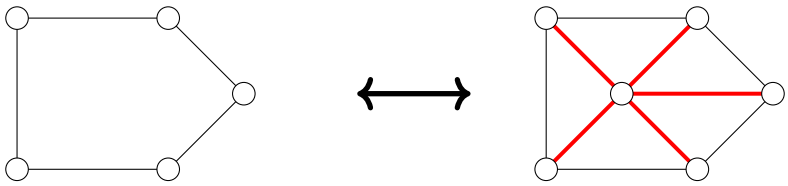


FIGURE 23 – Exemple 2, transformer un sous-graphe en super-graphe

Avec ces deux observations nous pourrons dire que chaque triangulation est 3-connexe, nous pouvons utiliser le théorème 9.1 pour dessiner la triangulation,

Theorem 9.1. *Toute triangulation planaire est au moins 3-connexe et possède un plongement planaire unique.* <*Kindermann, 2010*>

Ce théorème affirme que pour toute triangulation planaire donnée, il existe un unique plongement planaire qui représente graphiquement cette triangulation sans que les arêtes ne se croisent. Le plongement planaire est déterminé de manière unique et ne dépend pas de choix ou de variations arbitraires. De plus, la triangulation planaire est garantie d'être au moins 3-connexe, ce qui signifie qu'il y a au moins trois chemins indépendants reliant tout couple de sommets distincts dans le graphe planaire.

Et si nous avons n'importe quel graphe planaire, nous ajoutons simplement des arêtes pour en faire une triangulation, et pour le dessiner, il existe un **algorithme de Tutte** <Tutte, 1963> qui crée un dessin en ligne droite pour chaque graphe planaire. Et après on supprime à nouveau les arêtes <Kindermann, 2010>.

Algorithmes de dessin pour les graphes planaires des années 1990 :

Il existe deux algorithmes indépendamment découverts vers 1990 qui permettent de générer des dessins de graphes planaires de manière efficace :

1. Le premier par **De Fraysseix, Pach, Pollack** <De Fraysseix, Pach et Pollack, 1990> permet de dessiner une ligne droite simple pour chaque graphe planaire à$n$sommets sur une grille de taille **(2n-4)x(n-2)**.
2. Le seconde par **(Schnyder)** <W. Schnyder, 1990> obtient une liaison légèrement meilleure avec **(n-2)x(n-2)**.

L'idée principale de l'algorithme :
Nous choisissons une arête $\{v_1, v_2\}$ de la face externe du graphe que nous souhaitons dessiner. Ensuite, nous introduisons de nouveaux sommets de manière intuitive pour créer un nouveau sous-graphe, appelé G_i, comme illustré dans la figure suivante.

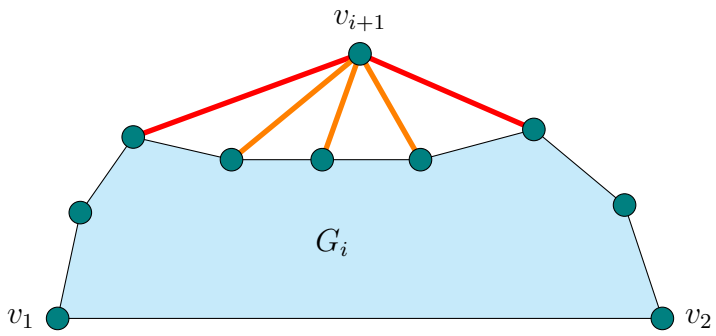


FIGURE 24 – Graphe G_i

Une fois cette étape accomplie, nous sélectionnons un sommet sur la face externe qui n'a pas encore été dessiné. La "face externe" fait référence à la région entourant le graphe, c'est-à-dire l'espace situé à l'extérieur de toutes les autres faces formées par les arêtes du graphe. Ensuite, nous établissons des connexions, c'est-à-dire des arêtes, entre ce sommet sélectionné et les autres sommets du graphe. Ces connexions respectent la propriété de planarité, ce qui signifie qu'elles sont tracées de manière à éviter les croisements d'arêtes. Pour réaliser ces dessins, il est nécessaire d'avoir au moins deux sommets situés sur la frontière du graphe, c'est-à-dire des sommets qui sont à la fois à l'intérieur et à l'extérieur du graphe.
Cette approche progressive nous permet de construire le dessin complet du graphe en ajoutant les sommets un par un, en suivant un schéma prédéfini et en garantissant la planarité du dessin final.

Dans les chapitres suivants nous allons voir à la **méthode Shift** qui prend un **graphe planaire** et son **ordre canonique** et donne le graphe dans cet ordre, cela fonctionne récursivement, mais **c'est quoi l'ordre canonique ?** C'est ce qu'on vas voir dans le chapitre suivant.

10 Ordre canonique

Dans le cadre de notre algorithme de dessin, nous nous concentrons sur l'ajout ordonné des sommets dans un ordre spécifique qui possède des propriétés particulières. Cet ordre, connu sous le nom d'ordre canonique, a été défini par <De Fraysseix, Pach et Pollack, 1990>. L'objectif de cet ordre est de garantir que l'ajout de chaque nouveau sommet ne crée pas de croisements indésirables <Kindermann, 2010, PartII> un exemple d'ordre canonique est illustré dans la figure suivante.

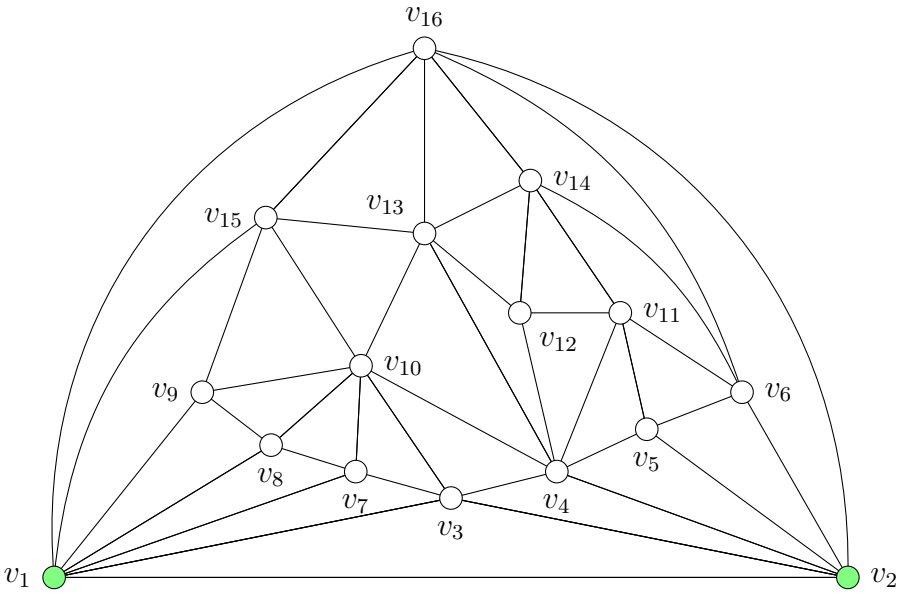


FIGURE 25 – Ordre canonique d'un graphe planaire maximal.

10.1 Conditions

Pour un cycle C dans un graphe, une arête joignant deux sommets non consécutifs de C est appelée une corde 7.14 de C (un exemple de graphe cordal est illustré à la figure 18). Pour un graphe planaire G à deux connexions, nous désignons par $C_0(G)$ le cycle extérieur de G, c'est-à-dire la frontière de la face extérieure de G. Un sommet sur $C_0(G)$ est appelé un sommet extérieur et une arête sur $C_0(G)$ est appelée une arête extérieure. Un graphe plan est intérieurement triangulé si chaque face intérieure est un triangle <N. Takao, R. Saidur, 2017>.

Supposons que nous avons un graphe planaire maximal triangulé $G = (V, E)$, avec n sommets, et soit u_0, u_1, u_2 les sommets externes de G dans le sens inverse des aiguilles d'une montre. **Un ordre canonique** de G (voir Figure) est un ordre $\pi = (v_1, v_2, \ldots, v_k, \ldots, v_n)$ des sommets de G tel que les conditions suivantes sont vérifiées :

— **Condition 1** :
$v_1 = u_1$, $v_2 = u_2$, l'arête (v_1, v_2) appartient à laface extérieure du graphe G voir la figure suivante ;

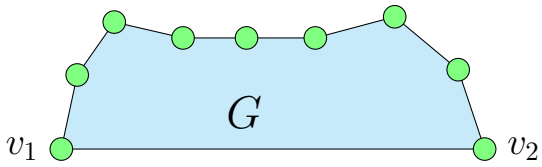


FIGURE 26 – Graphe avec l'arête (v_1, v_2) qui appartient à la face extérieure

— **Condition 2** :
Les sommets $\{v_1, v_2, \ldots, v_k\}$ induisent un graphe :
— **2-connexe** 8.2 c'est-à-dire que si l'on retire n'importe quel sommet, il reste connecté, voir figure 28 ;
— **Triangulé intérieurement**, c'est-à-dire chaque arête intérieure doit appartenir à un triangle, voir figure suivante 27 ;
— **Maximal à l'intérieur**, chaque sous-graphe G_k est maximal à l'intérieur, c'est-à-dire, toutes les faces internes de G_k sont des triangles, voir figure suivante 27.

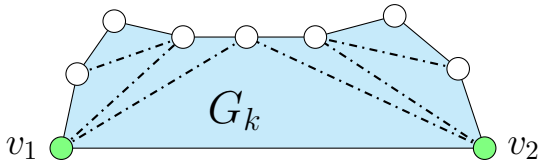


FIGURE 27 – Graphe G_k 2-connexe et triangulé

Pourquoi le graphe G_k doit-il être 2-connexe ?
Considérons l'exemple suivant 28 :

Exemple :

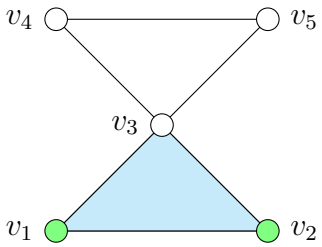


FIGURE 28 – Graphe non 2-connexe

Si nous effectuons la suppression du sommet v_3, le graphe résultant devient disconnexe. En d'autres termes, le graphe se divise en deux parties distinctes qui ne sont

pas connectées entre elles. Ce qui viole la propriété essentielle de 2-connexité 8.2. Aussi, Si nous supprimons le sommet v_5, une situation similaire se produit. Le graphe se fragmente en plusieurs parties disjointes, ce qui indique que le graphe initial ne satisfait pas la condition de 2-connexité. Il est important de noter qu'il existe au moins deux sommets, v_3 et v_5, dont la suppression entraîne une déconnexion du graphe, ce qui est en contradiction avec la propriété fondamentale de 2-connexité 8.2.

— **Condition 3** : Pour $3 \leq k \leq n$, soit G_k le sous-graphe plan de G induit par les sommets $v_1, \ldots, v_k$ et soit C_k la face externe de G_k. Le prochain sommet v_{k+1} que nous souhaitons installer se trouve sur la face C_k <N. Takao, R. Saidur, 2017, chapitre 4, page 46>. De plus, si $k < n$, le sommet v_k a au moins un voisin dans $G \setminus G_k$. Tous ses voisins apparaissent consécutivement [7] sur la frontière, pour $3 \leq k \leq n$ comme le montre bien la figure 29 précédente.

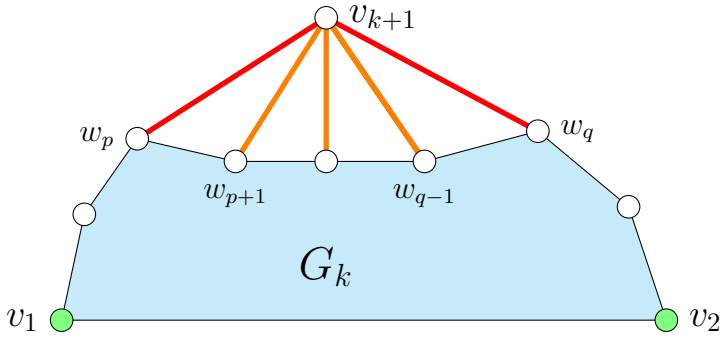


FIGURE 29 – Graphe insértion du sommet v_{k+1}

Ce sont exactement les conditions que l'algorithme de dessin a besoin <Kindermann, 2010> <Vismara, 2013>.

Un ordre canonique de G permet une construction incrémentale du graphe G à partir de l'arête (v_1, v_2). À l'étape k, $3 \leq k \leq n$, le sommet v_k et les arêtes entre v_k et ses voisins dans C_{k-1} sont ajoutés au graphe courant G_{k-1} (illustré sur la figure suivante en gras). Pour chaque $3 \leq k \leq n$, nous notons par $v_1 = w_1, w_2, ..., w_t = v_2$ la séquence des sommets de C_{k-1}, lorsqu'ils sont parcourus dans l'ordre horaire. Pour faciliter l'intuition, nous visualisons $w_2, ..., w_{t-1}$ comme étant disposés de gauche à droite au-dessus de $\{v_1, v_2\}$ dans le plan.

Pour chaque $3 \leq k \leq n$, soit $w_p, ..., w_q$ la sous-séquence des sommets de C_{k-1} qui sont adjacents à v_k (on note que $p + 1$ peut être égal à q). Après que v_k a été ajouté à G_{k-1}, les sommets $w_{p+1}, ..., w_{q-1}$ 29 (s'il y en a) ne sont plus externes ; nous disons que le sommet v_k couvre ces sommets.

7. "Voisins apparaissent consécutivement" signifie que les sommets adjacents (voisins) au sommet en question se présentent les uns à la suite des autres, sans interruption, sur la frontière de la face externe du graphe. Autrement dit, il n'y a pas d'autres sommets entre ces voisins lorsqu'ils sont disposés le long de la frontière.

Un ordre canonique $v_1, ..., v_n$ du graphe G définit un **spanning tree ou arbre couvrant** du graphe $G \setminus \{v_1, v_2\}$, appelé **arbre de couverture**, qui se compose de toutes les arêtes $\{u, v\}$ telles que u couvre v, ce qui signifie que le sommet u est directement relié au sommet v dans l'arbre de couverture du graphe $G \setminus \{v_1, v_2\}$. Cela indique une relation parent-enfant entre les sommets u et v dans l'arbre de couverture. En d'autres termes, u est un prédécesseur immédiat de v dans l'arborescence induite par l'ordre canonique du graphe planaire maximal.

Nous fixons v_n comme racine de l'arbre de couverture. Ainsi, les enfants d'un sommet u dans l'arbre de couverture sont les sommets couverts par u, voir la figure 30. Nous définissons la **forêt** de couverture associée à un ordre canonique comme étant son arbre de couverture avec les arbres à un seul sommet v_1 et v_2 <Vismara, 2013>.

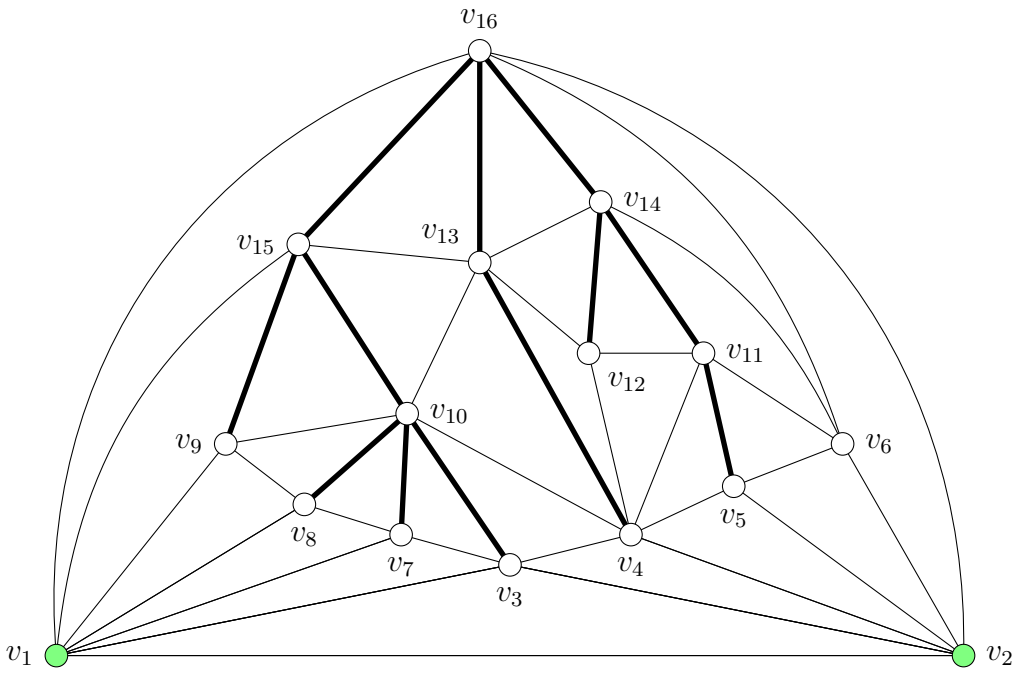


FIGURE 30 – Arbre de couverture induit par un ordre canonique d'un graphe planaire maximal. Les arêtes de l'arbre sont dessinées avec des traits épais noir

10.2 Exemple

Dans cette section, nous nous attacherons à établir un ordre canonique. À chaque étape de cette procédure, nous examinerons soigneusement si les trois conditions établies sont continuellement satisfaites. Pour faciliter la compréhension, nous avons utilisé une codification couleur :
Les sommets de couleur verte ont déjà été ajoutés à l'ordre canonique.
Les sommets de couleur bleue sont ceux qui viennent d'être ajoutés à l'ordre canonique.
Les sommets de couleur orange sont les prochains sommets qui seront ajoutés à l'ordre canonique.
Les arêtes bleues constituent le graphe formé par les sommets qui ont été sélectionnés pour l'ordre canonique.
Les arêtes orange seront incluses dans le graphe à mesure que les sommets orange seront sélectionnés pour l'ordre canonique.

Étape 1 :

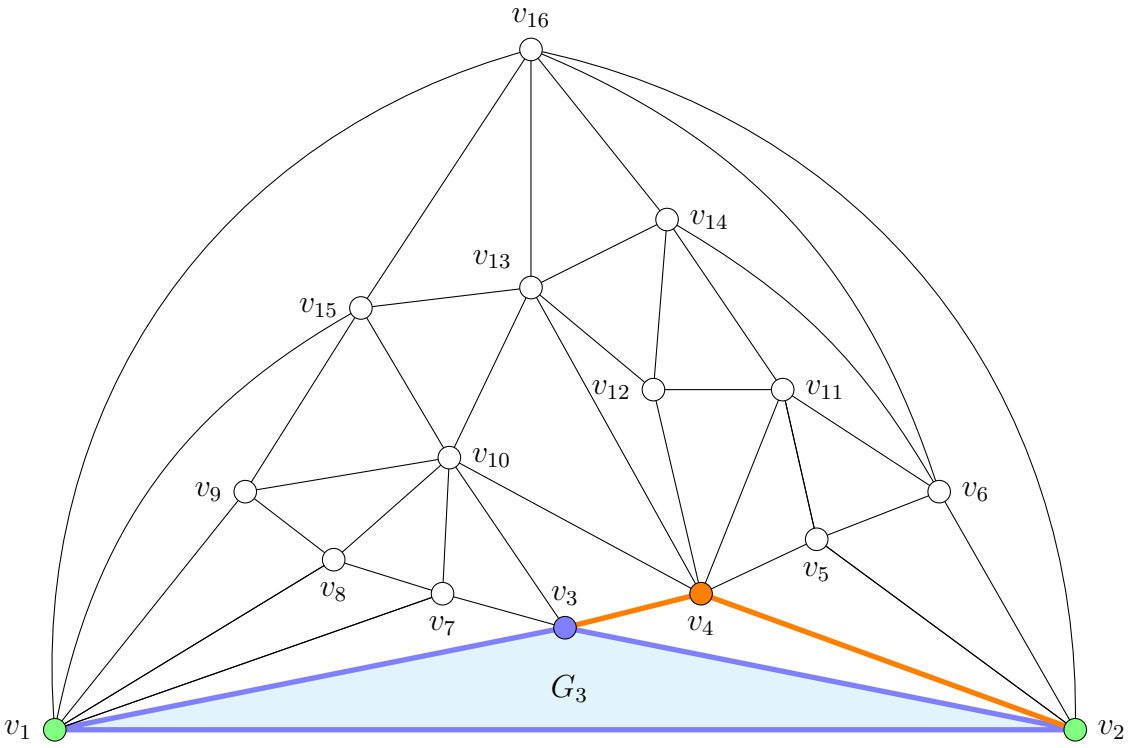


FIGURE 31 – Graphe G_3

Dans la première étape, notre objectif est d'intégrer le sommet v_3 (de couleur mauve) à l'ordre canonique, en le positionnant entre ses voisins consécutifs, v_1 et v_2.
Condition 1 : L'arête (v_1, v_2) appartient à la face extérieure du graphe.
Condition 2 : Les sommets $\{v_1, v_2, v_3\}$ induisent un graphe qui est :

— Biconnexe (ou 2-connexe),nous observons qu'en éliminant n'importe quel sommet, le graphe demeure connecté.

— Triangulé intérieurement : chaque arête interne appartient à un triangle.
— Maximal intérieurement pour chaque sous-graphe de G_3, et toutes les faces internes de sont des triangles.

Condition 3 : Nous avons $3 = k \leq n = 16$, et le prochain sommet $v_{k+1} = v_4$ (de couleur orange) se trouve dans la face extérieure. De plus, tous ses voisins, v_3 et v_2, apparaissent consécutivement sur la frontière du graphe.
Ainsi, nous constatons que les trois conditions sont bien respectées.

Étape 2 :

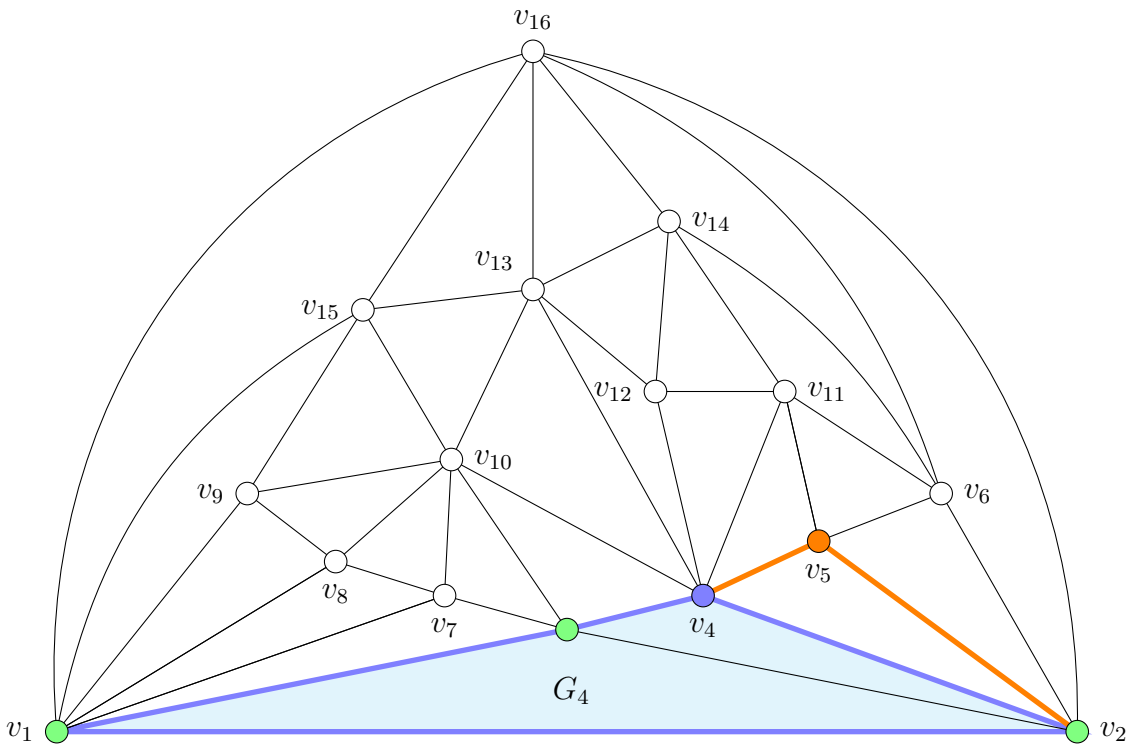


FIGURE 32 – Graphe G_4

Dans la seconde étape, nous voulons insérer le sommet v_4 à l'ordre canonique, en le positionnant entre les voisins consécutifs, v_3 et v_2.
Condition 1 : L'arête (v_1, v_2) appartient à la face extérieure du graphe.
Condition 2 : Les sommets $\{v_1, v_2, v_3, v_4\}$ induisent un graphe qui est :
— Biconnexe (ou 2-connexe),nous observons qu'en éliminant n'importe quel sommet, le graphe demeure connecté.
— Triangulé intérieurement : chaque arête interne appartient à un triangle.
— Maximal intérieurement pour chaque sous-graphe de G_4, et toutes les faces internes de sont des triangles.

Condition 3 : Nous avons $4 = k \leq n = 16$, et le prochain sommet $v_{k+1} = v_5$ se trouve dans la face extérieure. Ses voisins, v_4 et v_2, apparaissent consécutivement sur la frontière du graphe.
Ainsi, nous constatons que les trois conditions sont bien respectées.

Étape 3 :

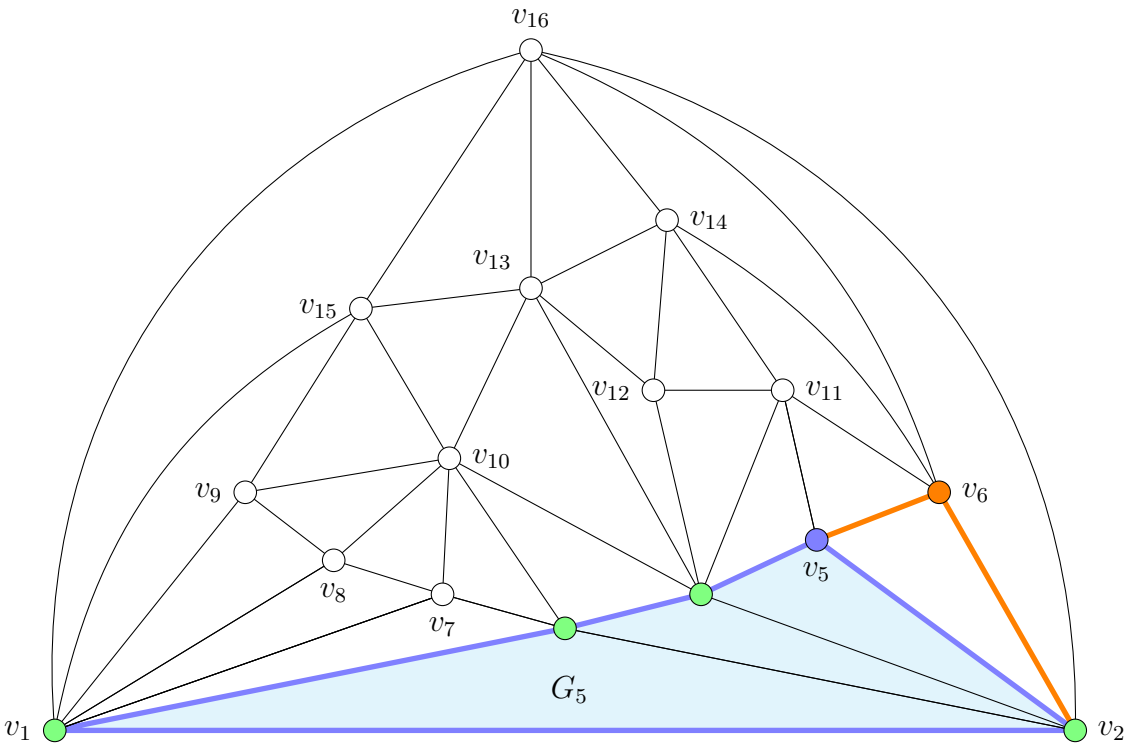


FIGURE 33 – Graphe G_5

Dans la troisième étape, nous voulons insérer le sommet v_5 à l'ordre canonique, en le positionnant entre les voisins consécutifs, v_4 et v_2.
Condition 1 : L'arête (v_1, v_2) appartient à la face extérieure du graphe.
Condition 2 : Les sommets $\{v_1, v_2, v_3, v_4, v_5\}$ induisent un graphe qui est :

- Biconnexe (ou 2-connexe),nous observons qu'en éliminant n'importe quel sommet, le graphe demeure connecté.
- Triangulé intérieurement : chaque arête interne appartient à un triangle.
- Maximal intérieurement pour chaque sous-graphe de G_5, et toutes les faces internes de sont des triangles.

Condition 3 : Nous avons $v_5 = k \leq n = 16$, et le prochain sommet $v_{k+1} = v_6$ se trouve dans la face extérieure. Les voisins, v_5 et v_2, apparaissent consécutivement sur la frontière du graphe.
Ainsi, nous constatons que les trois conditions sont bien respectées.

Étape 4 :

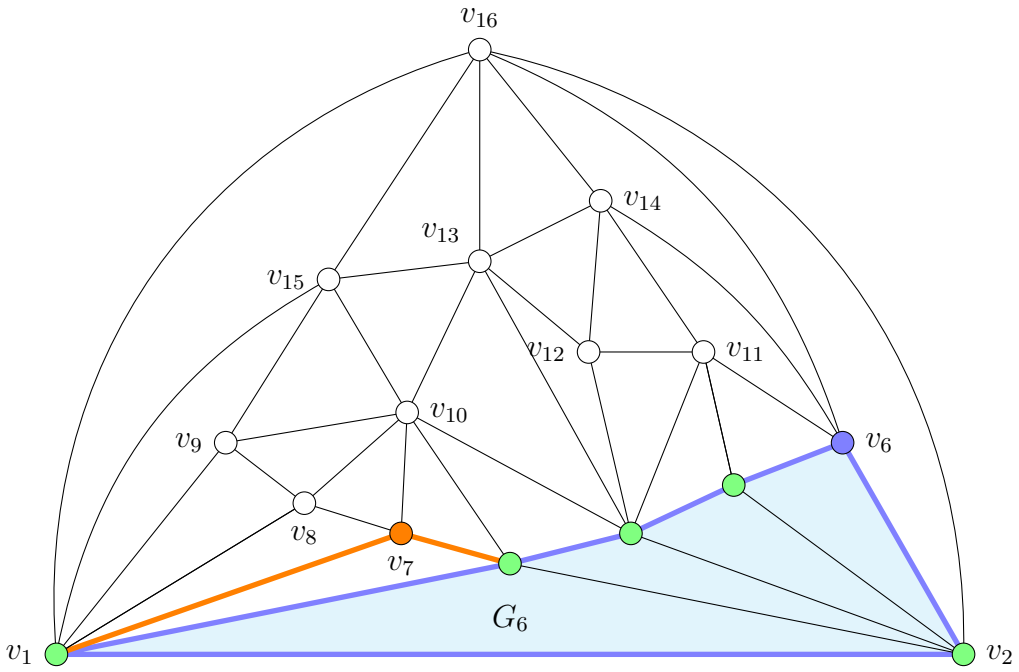


FIGURE 34 – Graphe G_6

Étape 5 :

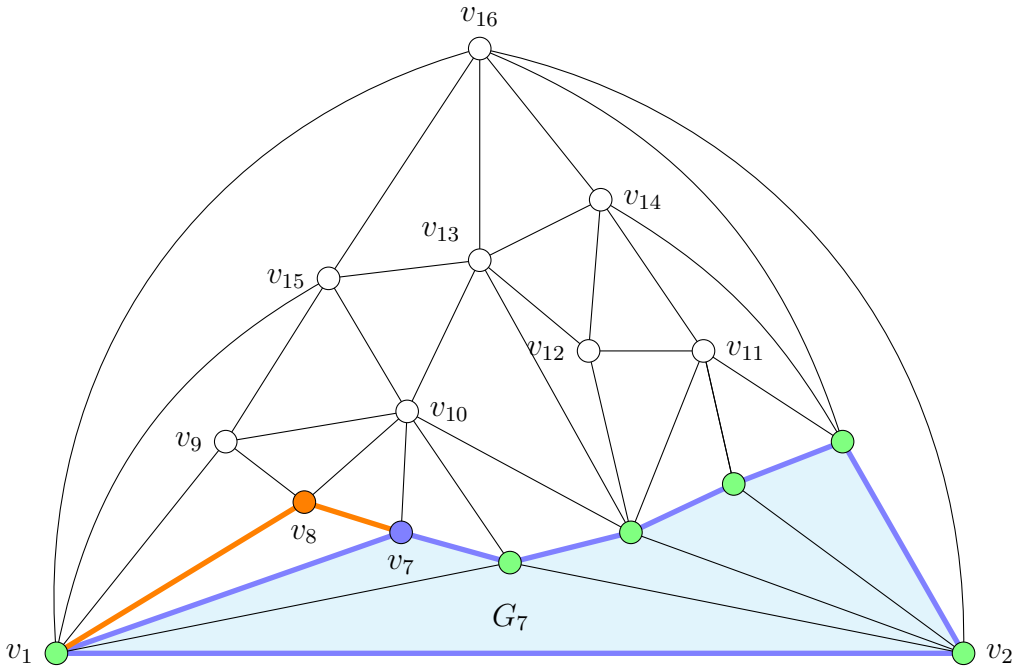


FIGURE 35 – Graphe G_7

Étape 6 :

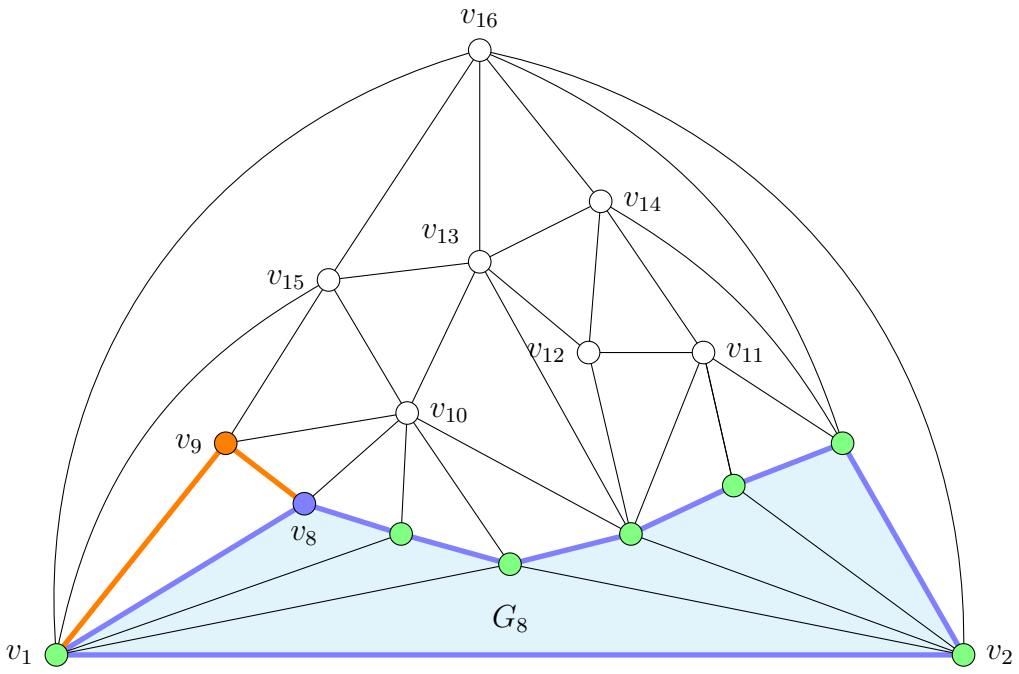


FIGURE 36 – Graphe G_8

Étape 7 :

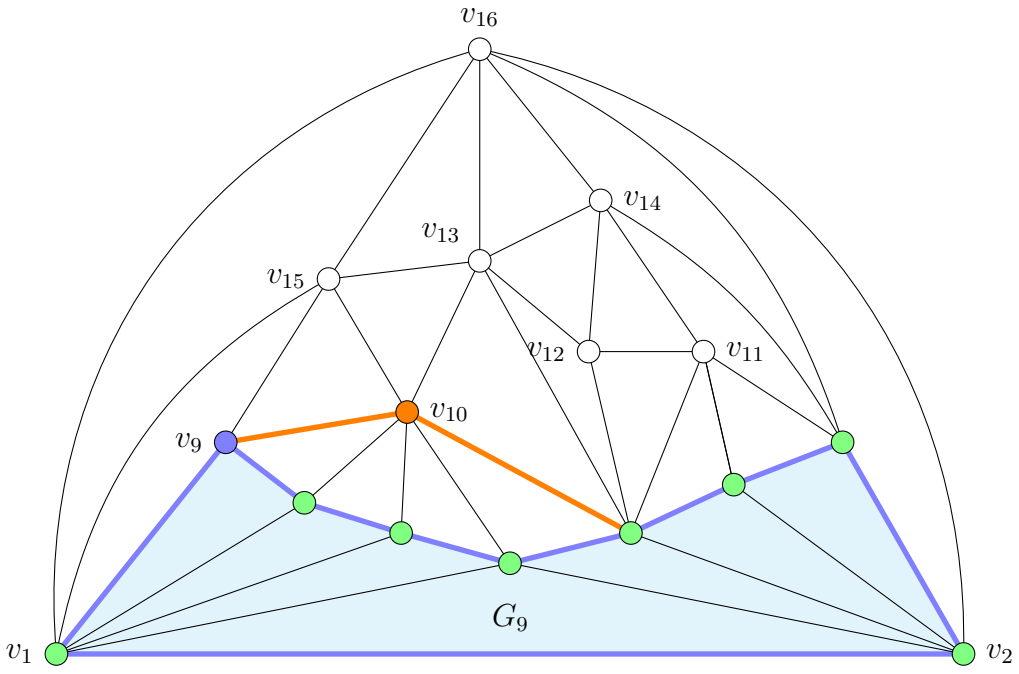


FIGURE 37 – Graphe G_9

Dans la septième étape nous souhaitons insérer le sommet v_9 dans l'ordre canonique en

le positionnant entre les voisins consécutifs v_1 et v_8.
Condition 1 : L'arête (v_1, v_2) appartient à la face extérieure du graphe.
Condition 2 : Les sommets $\{v_1, \ldots, v_9\}$ induisent un graphe qui est :

- Biconnexe (ou 2-connexe),nous observons qu'en éliminant n'importe quel sommet, le graphe demeure connecté.
- Triangulé intérieurement : chaque arête interne appartient à un triangle.
- Maximal intérieurement pour chaque sous-graphe de G_9, et toutes les faces internes de sont des triangles.

Condition 3 : Nous avons $v_9 = k \leq n = 16$, et le prochain sommet $v_{k+1} = v_{10}$ se trouve dans la face extérieure. Et les voisins v_9, v_8, v_7, v_3, v_4 apparaissent consécutivement sur la frontière du graphe.
Ainsi, nous constatons que les trois conditions sont bien respectées.

Étape 8 :

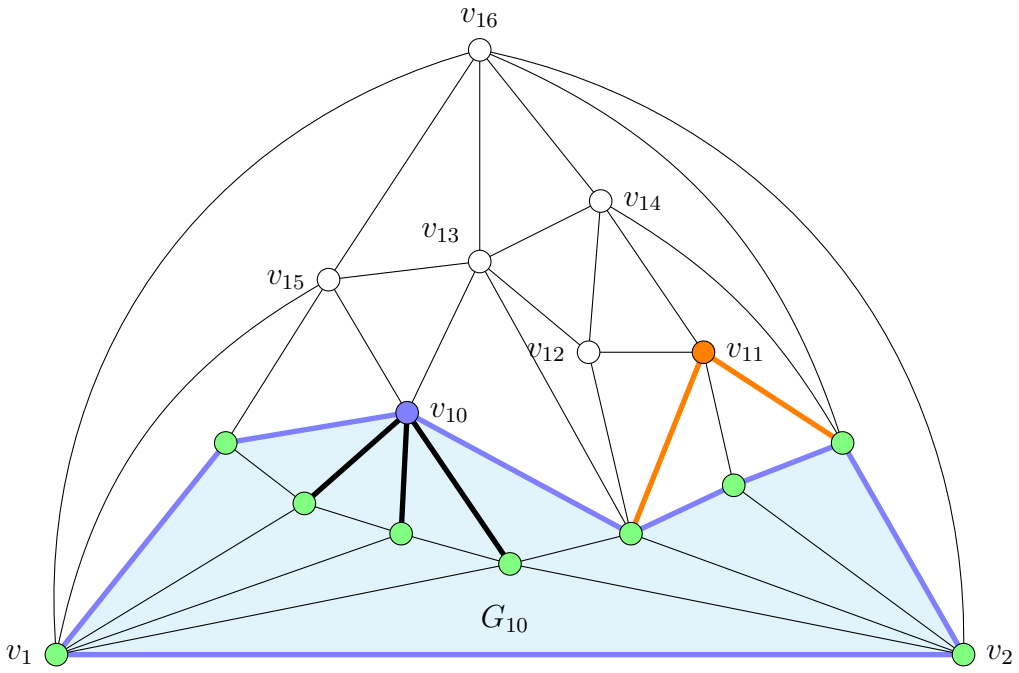


FIGURE 38 – Graphe G_{10}

Nous avons parcouru toutes les étapes précédentes jusqu'à l'étape 8, car elles étaient toutes similaires et ne présentaient aucune particularité. À présent, nous souhaitons insérer le sommet v_{10} à l'ordre canonique, en le positionnant entre les voisins consécutifs, v_9 et v_4.

Condition 1 : L'arête (v_1, v_2) appartient à la face extérieure du graphe.

Condition 2 : Les sommets $\{v_1, \ldots, v_{10}\}$ induisent un graphe qui est :

- Biconnexe (ou 2-connexe),nous observons qu'en éliminant n'importe quel sommet, le graphe demeure connecté.
- Triangulé intérieurement : chaque arête interne appartient à un triangle.

— Maximal intérieurement pour chaque sous-graphe de G_{10}, et toutes les faces internes de sont des triangles.

Condition 3 : Nous avons $v_{10} = k \leq n = 16$, et le prochain sommet $v_{k+1} = v_{11}$ se trouve dans la face extérieure. et les voisins v_4, v_5, v_6 apparaissent consécutivement sur la frontière du graphe.
Ainsi, nous constatons que les trois conditions sont bien respectées.

Observation :
Il convient de noter qu'une observation significative est que le sommet v_{10} remplace les sommets v_8, v_7, v_3 dans la frontière du graphe, ainsi, l'ordre des sommets sur la frontière devient : $\{v_1, v_9, v_{10}, v_4, v_5, v_6, v_2\}$ au lieu de : $\{v_1, v_9, v_8, v_7, v_3, v_4, v_5, v_6, v_2\}$
Donc le sommet v_{10} sera le parent des enfants v_8, v_7 et v_3 dans l'arbre couvrant.

Étape 9 :

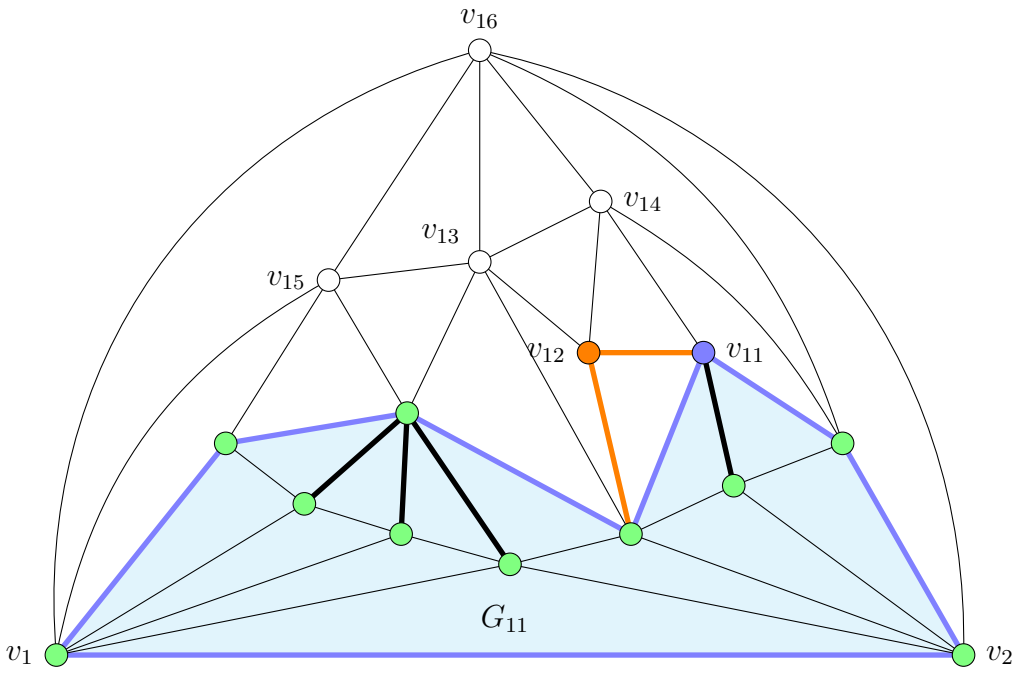


FIGURE 39 – Graphe G_{11}

Situation particulière :

Dans la neuvième étape nous souhaitons insérer le sommet v_{11} à l'ordre canonique, imaginons que nous voudrions le positionnant entre les voisins, v_{12} et v_6, On a un problème !
Car d'abord v_{12} n'apparaît pas sur la frontière ! (condition 3)

Le Graphe n'est pas 2-connexe (condition 2) : Imaginons si on supprime ce sommet v_{11} et en travers le graphe en partant de v_1 direction v_8, v_9, v_4, v_{12} on revient vers v_4, puis on

supprime v_4 on auras le point v_{12} isolé ! et c'est situation que nous nous voulons pas !

L'idéal est de le positionnant entre (deux voisins qui appartient à la frontière) comme les voisins, v_4 et v_6.

Condition 1 : L'arête (v_1, v_2) appartient à la face extérieure du graphe.

Condition 2 : Les sommets $\{v_1, \ldots, v_{11}\}$ induisent un graphe qui est :
- Biconnexe (ou 2-connexe),nous observons qu'en éliminant n'importe quel sommet, le graphe demeure connecté.
- Triangulé intérieurement : chaque arête interne appartient à un triangle.
- Maximal intérieurement pour chaque sous-graphe de G_{11}, et toutes les faces internes de sont des triangles.

Condition 3 : Nous avons $v_{11} = k \leq n = 16$, et le prochain sommet v_{k+1}=v_{12} se trouve dans la face extérieure. De plus, tous ses voisins, $v_4, v_1 1$, apparaissent consécutivement sur la frontière du graphe.
Ainsi, nous constatons que les trois conditions sont bien respectées.

Observation :
Une remarque à souligner : le sommet v_{11} prend la place de sommet v_5 dans la délimitation du graphe. De ce fait, la séquence des sommets se modifie pour devenir : $\{v_1, v_9, v_{10}, v_4, v_{11}, v_6, v_2\}$ au lieu de la séquence initiale : $\{v_1, v_9, v_{10}, v_4, v_5, v_6, v_2\}$
Donc le sommet v_{11} sera le parent d'enfant v_5 dans l'arbre couvrant.

Étape 10 :

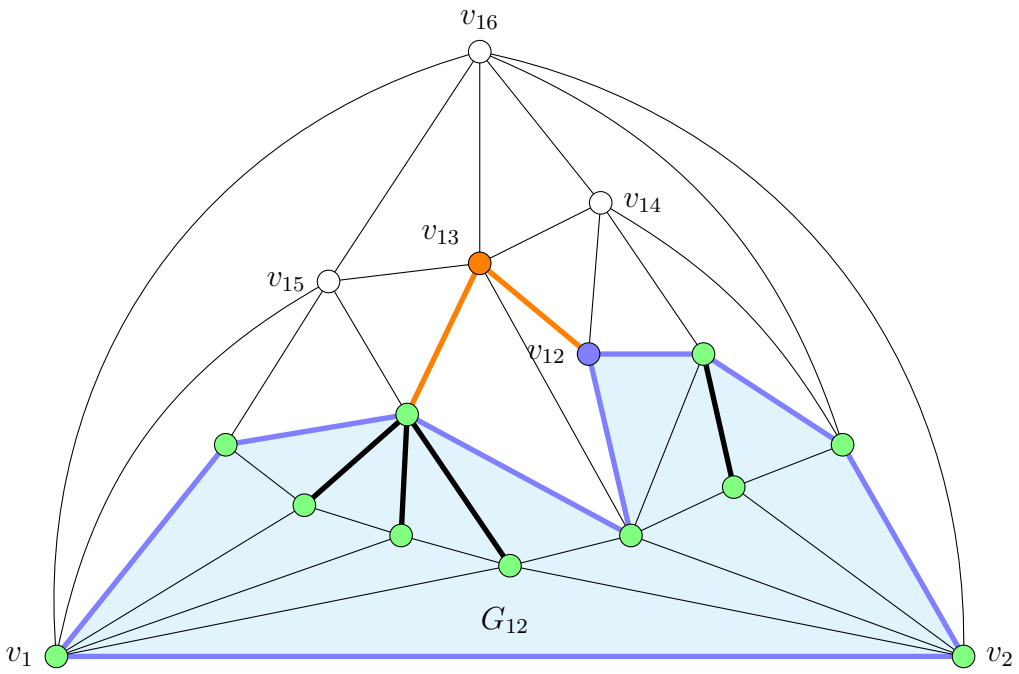


FIGURE 40 – Graphe G_{12}

Dans la dixième étape nous souhaitons insérer le sommet v_{12} à l'ordre canonique, en le positionnant entre les voisins consécutifs, v_4 et v_{11}.

Condition 1 : L'arête (v_1, v_2) appartient à la face extérieure du graphe.

Condition 2 : Les sommets $\{v_1, \ldots, v_{12}\}$ induisent un graphe qui est :
- Biconnexe (ou 2-connexe),nous observons qu'en éliminant n'importe quel sommet, le graphe demeure connecté.
- Triangulé intérieurement : chaque arête interne appartient à un triangle.
- Maximal intérieurement pour chaque sous-graphe de G_{12}, et toutes les faces internes de sont des triangles.

Condition 3 : Nous avons $v_{12} = k \leq n = 16$, et le prochain sommet v_{k+1}=v_{13} se trouve dans la face extérieure. Tous ses voisins v_{10}, v_4, v_{12} apparaissent consécutivement sur la frontière du graphe.
Ainsi, nous constatons que les trois conditions sont bien respectées.

Observation :
Une remarque : Le sommet v_{12} n'a pas d'enfant dans l'arbre couvrant, de ce fait, la séquence des sommets se modifie pour devenir : $\{v_1, v_9, v_{10}, v_4, v_{12}, v_{11}, v_6, v_2\}$ au lieu de la séquence initiale : $\{v_1, v_9, v_{10}, v_4, v_{11}, v_6, v_2\}$.

Étape 11 :

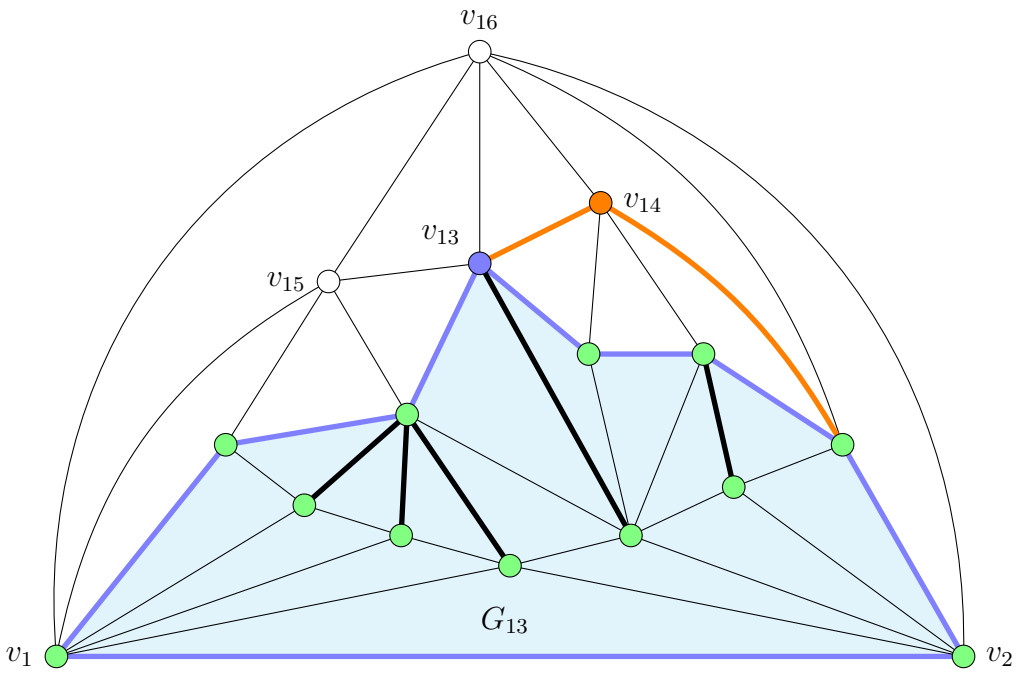


FIGURE 41 – Graphe G_{13}

Dans la onzième étape nous souhaitons insérer le sommet v_{13} à l'ordre canonique, en le positionnant entre les voisins consécutifs, v_{10} et v_{12}.

Condition 1 : L'arête (v_1, v_2) appartient à la face extérieure du graphe.

Condition 2 : Les sommets $\{v_1, \ldots, v_{13}\}$ induisent un graphe qui est :
- Biconnexe (ou 2-connexe),nous observons qu'en éliminant n'importe quel sommet, le graphe demeure connecté.
- Triangulé intérieurement : chaque arête interne appartient à un triangle.
- Maximal intérieurement pour chaque sous-graphe de G_{13}, et toutes les faces internes de sont des triangles.

Condition 3 : Nous avons $v_{13} = k \leq n = 16$, et le prochain sommet v_{k+1}=v_{14} se trouve dans la face extérieure. Et tous ses voisins $v_{13}, v_{12}, v_{11}, v_6$ apparaissent consécutivement sur la frontière du graphe.
Ainsi, nous constatons que les trois conditions sont bien respectées.

Observation :
Une remarque à souligner : le sommet v_{13} sera le parent d'enfant v_4 dans l'arbre couvrant, ainsi la séquence des sommets se modifie pour devenir : $\{v_1, v_9, v_{10}, v_{13}, v_{12}, v_{11}, v_6, v_2\}$ au lieu de la séquence initiale : $\{v_1, v_9, v_{10}, v_4, v_{12}, v_{11}, v_6, v_2\}$.

Étape 12 :

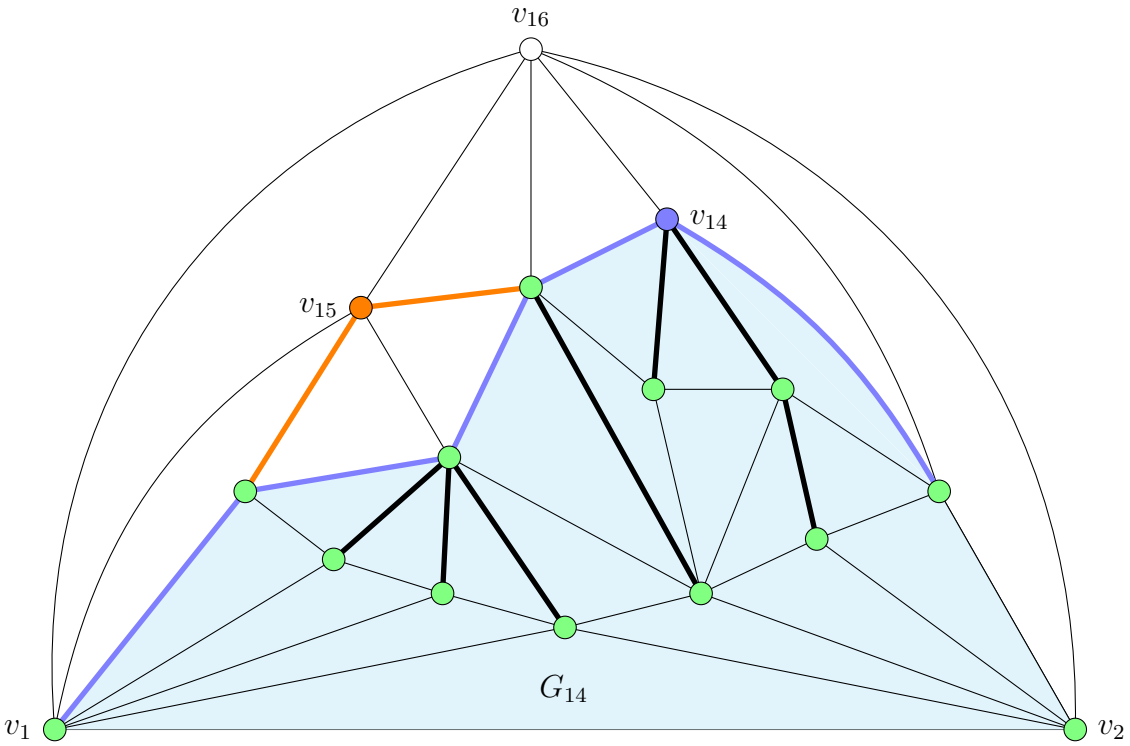


FIGURE 42 – Graphe G_{14}

Dans la douzième étape nous souhaitons insérer le sommet v_{14} à l'ordre canonique, en le positionnant entre les voisins consécutifs, v_{13} et v_6.

Condition 1 : L'arête (v_1, v_2) appartient à la face extérieure du graphe.

Condition 2 : Les sommets $\{v_1, \ldots, v_{14}\}$ induisent un graphe qui est :
- Biconnexe (ou 2-connexe),nous observons qu'en éliminant n'importe quel sommet, le graphe demeure connecté.
- Triangulé intérieurement : chaque arête interne appartient à un triangle.
- Maximal intérieurement pour chaque sous-graphe de G_{14}, et toutes les faces internes de sont des triangles.

Condition 3 : Nous avons $v_{14} = k \leq n = 16$, et le prochain sommet $v_{k+1}{=}v_{15}$ se trouve dans la face extérieure. De plus, tous ses voisins, v_9, v_{10}, v_{13} apparaissent consécutivement sur la frontière du graphe.
Ainsi, nous constatons que les trois conditions sont bien respectées.

Observation :
Une remarque à souligner : le sommet v_{14} sera le parent des enfants v_{12}, v_{11} dans l'arbre couvrant, ainsi la séquence des sommets se modifie pour devenir : $\{v_1, v_9, v_{10}, v_{13}, v_{14}, v_6, v_2\}$ au lieu de la séquence initiale : $\{v_1, v_9, v_{10}, v_{13}, v_{12}, v_{11}, v_6, v_2\}$.

Étape 13 :

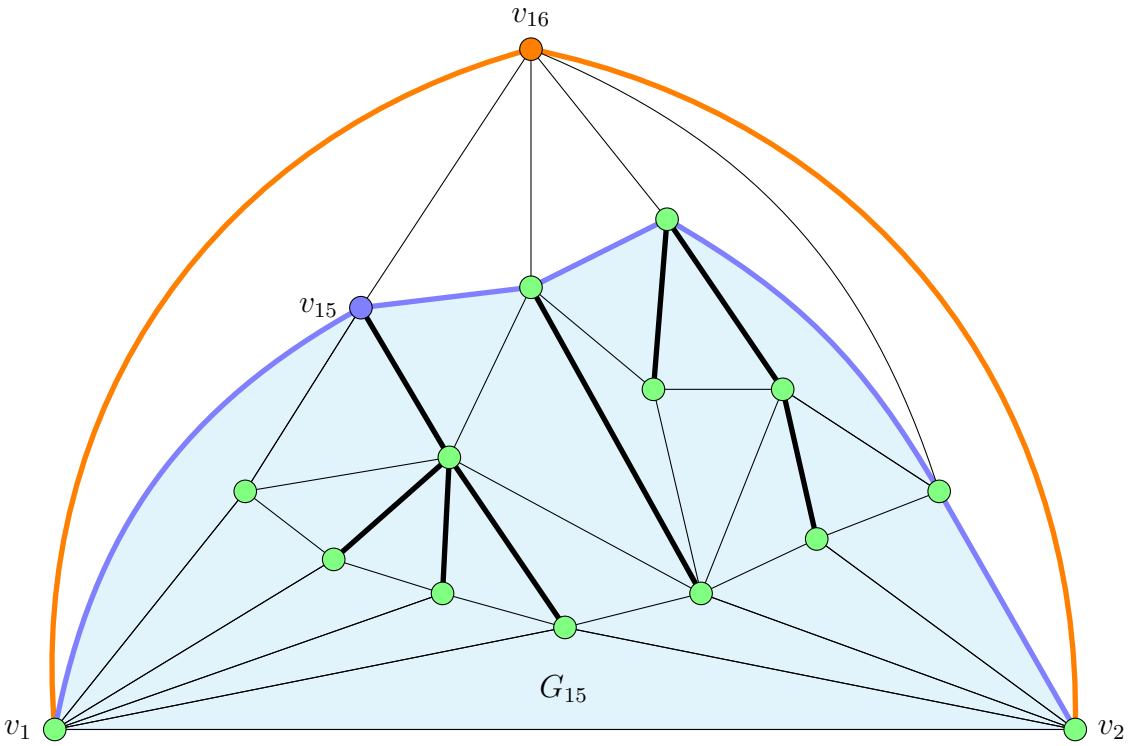


FIGURE 43 – Graphe G_{15}

Dans la treizième étape nous souhaitons insérer le sommet v_{15} à l'ordre canonique, en le positionnant entre les voisins consécutifs, v_1 et v_{13}.

Condition 1 : L'arête (v_1, v_2) appartient à la face extérieure du graphe.

Condition 2 : Les sommets $\{v_1, \ldots, v_{15}\}$ induisent un graphe qui est :
- Biconnexe (ou 2-connexe),nous observons qu'en éliminant n'importe quel sommet, le graphe demeure connecté.
- Triangulé intérieurement : chaque arête interne appartient à un triangle.
- Maximal intérieurement pour chaque sous-graphe de G_{15}, et toutes les faces internes de sont des triangles.

Condition 3 : Nous avons $v_{15} = k \leq n = 16$, et le prochain sommet v_{k+1}=v_{16} se trouve dans la face extérieure. Et tous ses voisins v_{15}, v_{13}, v_{14}, apparaissent consécutivement sur la frontière du graphe.
Ainsi, nous constatons que les trois conditions sont bien respectées.

Observation :
Une remarque à souligner : le sommet v_{15} sera le parent des enfants v_9, v_{10} dans l'arbre couvrant, ainsi la séquence des sommets se modifie pour devenir : $\{v_1, v_{15}, v_{13}, v_{14}, v_6, v_2\}$ au lieu de la séquence initiale : $\{v_1, v_9, v_{10}, v_{13}, v_{14}, v_6, v_2\}$.

Étape 14 :

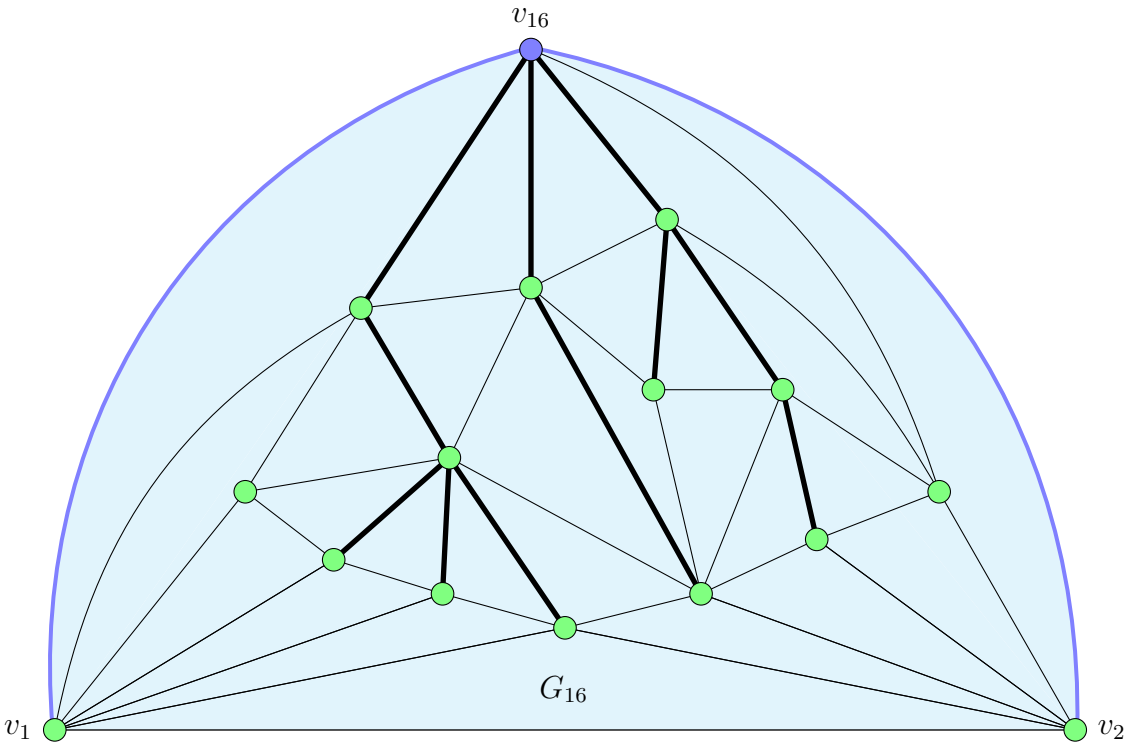


FIGURE 44 – Graphe G_{16}

Dans la treizième étape nous souhaitons insérer le sommet v_{16} à l'ordre canonique, en le positionnant entre les voisins consécutifs, v_1 et v_2.

Condition 1 : L'arête (v_1, v_2) appartient à la face extérieure du graphe.

Condition 2 : Les sommets $\{v_1, \ldots, v_{16}\}$ induisent un graphe qui est :
- Biconnexe (ou 2-connexe),nous observons qu'en éliminant n'importe quel sommet, le graphe demeure connecté.
- Triangulé intérieurement : chaque arête interne appartient à un triangle.
- Maximal intérieurement pour chaque sous-graphe de G_{16}, et toutes les faces internes de sont des triangles.

Condition 3 : Nous avons $v_{16} = k \leq n = 16$, nous n'avons pas de prochain sommet.
Ainsi, nous constatons que les trois conditions sont bien respectées.

Observation :
Une remarque à souligner : le sommet v_{16} sera le parent des enfants v_{15}, v_{13}, v_{14} dans l'arbre couvrant, ainsi la séquence des sommets se modifie pour devenir : $\{v_1, v_{16}, v_2\}$ au lieu de la séquence initiale : $\{v_1, v_{15}, v_{13}, v_{14}, v_6, v_2\}$.

10.3 Pseudo code de l'Ordre Canonique

L'ordre canonique est un concept essentiel en théorie des graphes qui permet de déterminer un ordre linéaire des sommets d'un graphe planaire. Cela permet de représenter graphiquement la structure du graphe de manière ordonnée, ce qui facilite son analyse et sa compréhension. Les algorithmes de l'ordre canonique sont conçus pour être efficaces, garantissant un temps d'exécution linéaire et les rendant adaptés aux graphes de grande taille. Ils utilisent des marqueurs de sommets, des sommets externes et des corde pour établir l'ordre correct des sommets et maintenir la cohérence planaire du graphe. En résumé, l'ordre canonique permet d'obtenir une représentation structurée et systématique des sommets d'un graphe planaire, favorisant ainsi une meilleure analyse et compréhension de sa connectivité.

L'algorithme suivant <N. Takao, R. Saidur, 2017> permet de calculer un ordre canonique pour un graphe plan triangulé $G = (V, E)$. Pour chaque sommet v, nous utilisons les variables suivantes :

1. mark(v) : indique si v a été ajouté à l'ordre (true) ou non (false).
2. out(v) : indique si v est un sommet extérieur du graphe plan actuel (true) ou non (false).
3. corde(v) : le nombre de cordes du cycle extérieur dont v est l'extrémité.

Algorithm 1: Pseudo code de l'Ordre Canonique

Entrée: (Graphe $G = (V, E)$, (v_1, v_2, v_n))
Sortie: (Ordre canonique des sommets)

1 **Début**
  //Initialisation
2 **Pour** chaque sommet v dans V **faire**
3 corde(v) $\leftarrow 0$; //Initialiser le nombre de corde du sommet v à 0
4 out(v) $\leftarrow$ false ; //Marquer le sommet v comme n'étant pas un sommet externe
5 mark(v) $\leftarrow$ false ; //Marquer le sommet v comme n'ayant pas été ajouté à l'ordre canonique
6 mark($v1$) $\leftarrow$ true ; mark($v2$) $\leftarrow$ true ; out($v1$) $\leftarrow$ true ; out($v2$) $\leftarrow$ true ;
7 out(vn) $\leftarrow$ true ;
  //Construction de l'ordre canonique
8 **Pour** k de n à 3 **faire**
9 choisir v tel que mark(v) = false, out(v) = true et corde(v) = 0;
10 $v_k \leftarrow v$, mark(v) $\leftarrow$ true ;
  //Détermination des voisins
11 $w1 = v1, w2, \ldots, wt - 1, wt = v2$;
12 Notation : $wp, \ldots, wq$ désigne les voisins de v_k;
13 mark(w_i) = false ;
  //Mise à jour des sommets et de leurs voisins
14 **Pour** chaque sommet wi avec $p < i < q$ **faire**
15 out(wi) $\leftarrow$ true ; //Marquer le sommet wi comme un sommet externe
16 Mettre à jour le nombre de corde pour wi et ses voisins;

Explication

la ligne 6 : Marquer les sommets v_1, v_2, v_n comme true pour mark(v) et out(v), car ce sont les sommets externes du graphe initial.
la ligne 9 : Cela garantit que nous sélectionnons un sommet qui n'a pas encore été ajouté à l'ordre canonique et qui est un sommet externe sans corde.
La ligne 10 : Assigner $v_k = v$ et marquer v comme true pour indiquer qu'il a été ajouté à l'ordre canonique.
La line 11 : Définir $w1 = v1, w2, \ldots, wt - 1, wt = v2$ comme les sommets délimitant le bord de G_{k-1}. Ces sommets serviront de référence pour trouver les voisins de v_k.
La ligne 12 et 13 : Trouver w_p, w_{p+1}, ..., w_q qui sont les voisins de v_k et ont mark(w_i) = false. Ces sommets sont les voisins non ajoutés de v_k et se trouvent entre les sommets w_1 et w_t dans l'ordre canonique précédent.
La dernière boucle à la ligne 14 pour chaque sommet w_i :
La ligne 15 : Marquer le sommet w_i comme un sommet externe
La ligne 16 : Mettre à jour le nombre de corde pour w_i et ses voisins. Cela peut nécessiter de parcourir les arêtes du graphe pour vérifier les connexions entre les sommets et mettre à jour les valeurs de corde en conséquence. Cette mise à jour permet de garder une trace du nombre de corde pour chaque sommet externe.

11 Shift method

11.1 Conditions pour dessiner des invariants

Nous allons maintenant examiner la méthode de **Shift** qui prend en entrée un **graphe planaire** et l'**ordre canonique** que nous avons vu dans la section précédente (bien qu'il existe plusieurs types d'ordres canoniques tels que leftist, rightist, *etc* <Eppstein, D. and Gansner, Emden R., 2010>). Cette méthode produit en sortie un graphe selon cet ordre. Cela fonctionne de manière inductive, donc nous souhaitons préserver certains invariants de dessin. Plus précisément, à chaque étape k, on place le sommets v_k, avec $k = \{3, 4, 5, \ldots, n\}$, et nous veillons à ce que notre graphe G_{k-1} soit dessiné de telle sorte que v_1 se situe exactement à l'origine et v_2 se situe aux coordonnées $(2k-6, 0)$. Ainsi, il est dessiné sous la forme d'un segment horizontal, voir la figure suivante ;

FIGURE 45 – Le graphe G_{k-1} dessiné comme un segement

La frontière du graphe G_{k-1} à l'exception de l'arête $\{v_1, v_2\}$ est dessiné x-monotone, signifie que pour chaque arête de la frontière de G_{k-1}, lorsque nous la suivons de gauche à droite le long de l'axe des abscisses, elle ne se croise pas avec une autre arête et ne change pas de direction horizontale. Cette propriété garantit que les arêtes de la frontière de G_{k-1} sont dessinées de manière ordonnée et cohérente le long de l'axe des abscisses.
De plus, chaque arête est dessinée avec une pente ± 1 . C'est ainsi que pourrait se présenter la frontière de G_{k-1}.
Supposons maintenant que nous disposions d'un dessin de G_{k-1} respectant toutes ces conditions, voir la figure suivante.

Conditions

1. v_1 se trouve sur (0, 0) et v_2 sur (2k-6, 0) ;
2. Le frontière de G_{k-1} (sauf l'arête $\{v_1, v_2\}$) est dessiné x-monotone ;
3. Chaque arête de la frontière du G_{k-1} (sauf l'arête $\{v_1, v_2\}$) est dessinée avec des pentes de ± 1.

Placement de sommet v_k

À l'étape k, nous voulons placer notre nouveau sommet v_k comme illustré dans la figure suivante 46. Nous avons le voisin le plus à gauche w_p et le voisin le plus à droite w_q, et nous devons nous assurer que ces conditions sont toujours respectées.

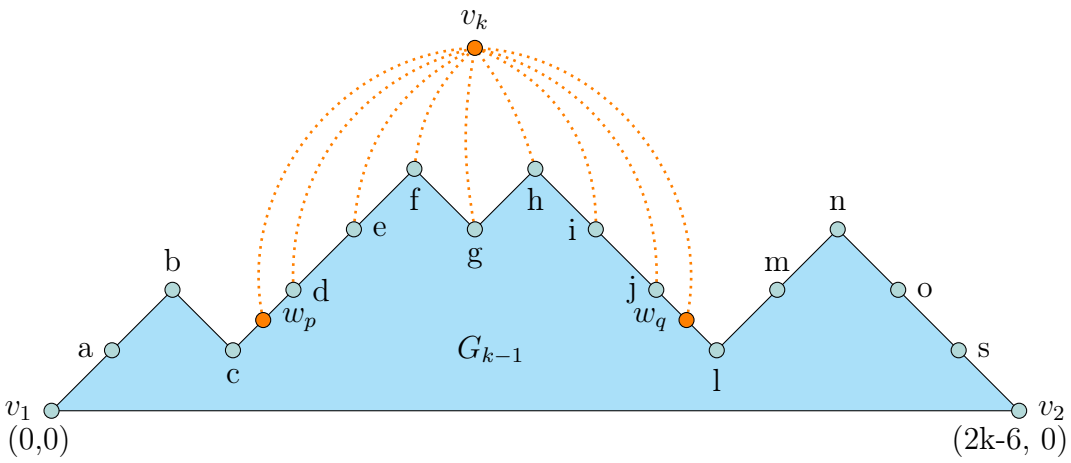


FIGURE 46 – Le graphe G_{k-1} à l'étape $k-1$, avant de placer le sommet k

Problème de superposition d'arêtes

La position de v_k est déterminée par l'intersection de la droite passante par w_p avec une pente de $+1$ et de la droite passante par w_q avec une pente de -1, comme le montre la figure suivante 47.

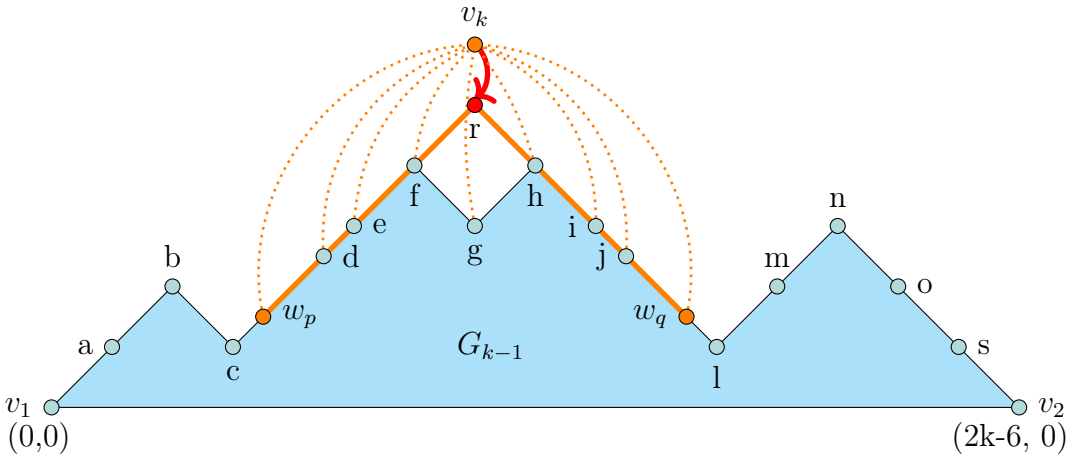


FIGURE 47 – La position du sommet v_k sur le graphe G_{k-1} est à l'intersection de la droite passante par wp avec une pente $+1$ et de la droite passante par w_q avec une pente -1

Par conséquent, nous devons placer le sommet à cette position r, et là on a un problème, les arêtes $\{v_k, h\}$, $\{v_k, i\}$, $\{v_k, j\}$, ... seront superposées, donc le dessin n'est plus planaire. Cependant, un avantage de cette situation est que lorsqu'on passe de G_{k-1} à G_k, cela nous permet d'obtenir deux coordonnées x supplémentaires. Ainsi, nous avons la possibilité de déplacer les sommets.

Découpages le graphe

Et donc ce que nous pouvons faire c'est de créer deux coupures : une entre le voisin le plus à gauche et son voisin, et une autre entre le voisin le plus à droite et son voisin, et maintenant, nous déplaçons tous les sommets qui se trouvent à gauche d'une x-coordonnée à gauche w_p et tous les sommets qui se trouvent à droite de cette même coordonnée à droite w_q, voir la figure suivante.

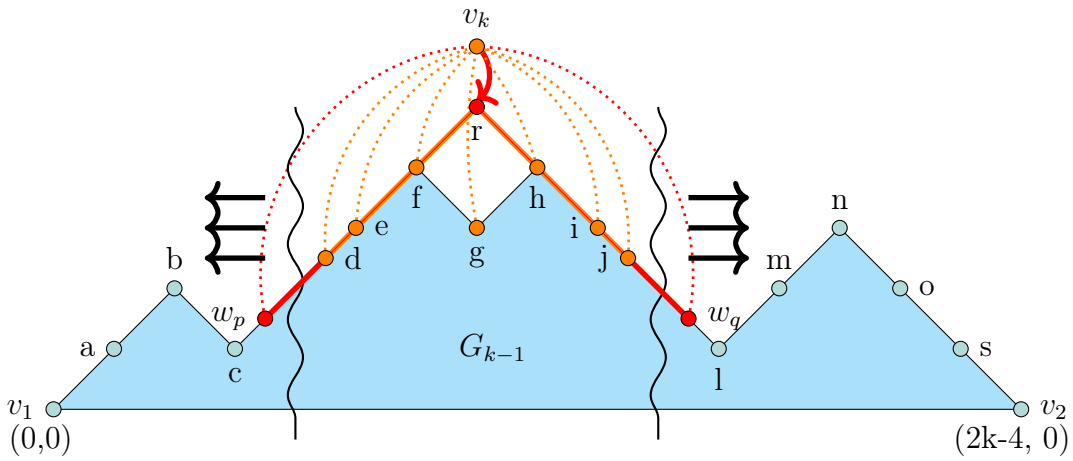


FIGURE 48 – Illustration de l'idée de décaler la partie gauche vers la gauche, et la partie droite vers la droite du G_{k-1}, après avoir découpé le graphe

Reconnexion des arêtes $\{w_p, v_k\}$ et $\{w_q, v_k\}$ découpées avant

Après cela, dans la figure suivante, nous procédons à la reconnexion des deux arêtes $\{w_p, v_k\}$ et $\{w_q, v_k\}$. La reconnexion des arêtes est réalisée en les reliant au sommet v_k, comme illustré dans la Figure 49. Les résultats de cette reconnexion sont prometteurs, montrant une nette amélioration par rapport à l'état précédent. Cependant, il convient de mener davantage d'analyses et de tests pour fournir une évaluation complète de cette méthode.

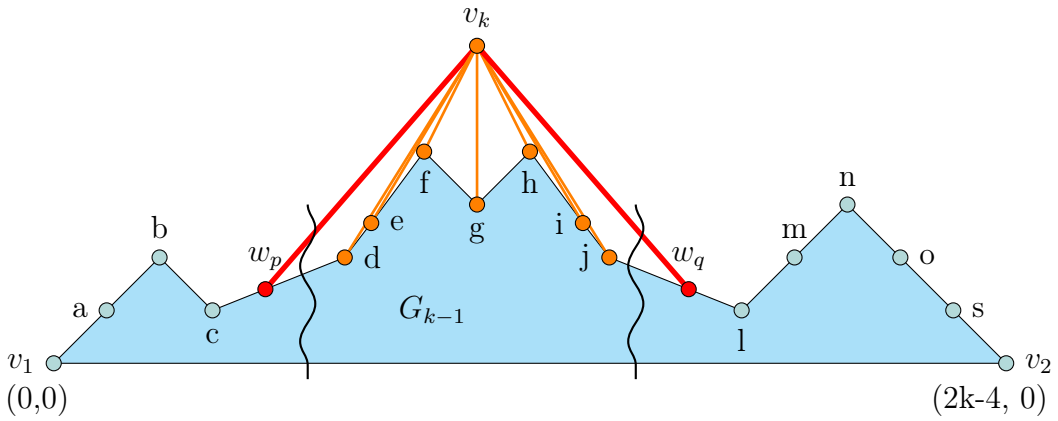


FIGURE 49 – Insertion du sommet v_k dans le graphe G_{k-1} après le décalage

Eh bien, une dernière chose que nous devons corriger est que nous avons déplacé tout vers la gauche, donc cela serait (-1,0), mais nous pouvons simplement déplacer l'ensemble du graphe vers la droite de 1, et ensuite les conditions 1, 2 et 3 (11.1) sont clairement satisfaites.

Position du sommet v_k sur la grille

Y a des questions que nous devons se poser, pouvons-nous placer v_k dans cette position d'intersection de la droite passante par wp avec une pente $+1$ et de la droite passante par w_q avec une pente -1 ? Pourrait-il se trouver quelque part au milieu entre les lignes de la grille ? comme illustré dans la figure suivante 50. Ou sur une ligne verticale (horizontale) de la grille, mais pas sur une ligne horizontale (verticale) de la grille ? Ou l'inverse ? Cependant, nous pouvons montrer qu'il se trouve réellement sur une ligne de la grille. Cela provient de condition 3.

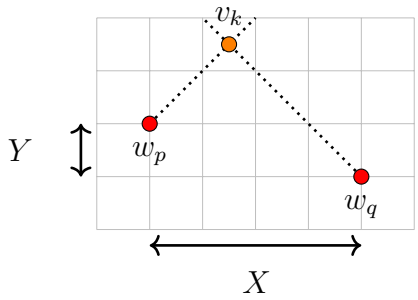


FIGURE 50 – Exploration d'une possible position du sommet v_k sur la grille (à vérifier)

De plus, pour cela, nous voulons examiner **la distance de Manhattan**, c'est ce que nous allons voir sur la section suivante.

11.2 Distance de Manhattan

Et on peut déplacer v_k sur cette nouvelle position, nous pouvons montrer qu'il se trouve vraiment sur une ligne de la grille. De plus, cela découle de la condition 3 11.1.
La **distance de Manhattan** entre X et Y, c'est simplement la somme de la distance en X et la distance en Y, voir l'exemple de la figure suivante :
Example, D = $|x_2$ - $x_1|$ + $|y_2$ - $y_1|$ = |4 - 1| + |2 - 3| = 3 + 1 = 4

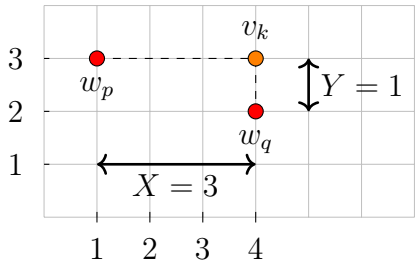


FIGURE 51 – Distance de Manhattan entre w_p et w_q

Démonstration que l'intersection de la droite passant par wp avec une pente $+1$ et de la droite passant par wq avec une pente -1 se situe sur une ligne horizontale et une ligne verticale.

Démonstration que l'intersection se situe sur une ligne horizontale/verticale. Et à partir des pentes, il s'ensuit que la distance de Manhattan est paire entre w_p et w_q, entre w_p et v_k, ainsi qu'entre w_q et v_k. Donc, si nous nous déplaçons du sommet w_p jusqu'à v_k et suivons simplement les segments, à chaque étape, on a une pente de $+1$. Nous parcourons donc le même nombre de coordonnées vers la droite que vers le haut. Donc, entre chaque paire de sommets consécutifs sur la frontière, nous parcourons toujours la même distance en coordonnée y et la même distance en coordonnée x. De même lorsque nous nous déplaçons du sommet v_k jusqu'à w_q nous parcourons aussi la même distance en coordonnée y et la même distance en coordonnée x. La distance est également paire entre w_p et w_q.

Soit $w_p(x_p, y_p)$ et $w_q(x_q, y_q)$ les coordonnées de deux points distincts.
La droite passante par w_p avec une pente $+1$ peut être exprimée par l'équation suivante :

$$y = x - (x_p - y_p)$$

La droite passante par w_q avec une pente -1 peut être exprimée par l'équation suivante :

$$y = -x + (x_q + y_q)$$

Pour prouver que l'intersection de ces deux droites se situe sur une ligne horizontale et une ligne verticale de la grille, nous devons montrer que les coordonnées de l'intersection satisfont cette propriété.

Pour trouver l'intersection, nous égalons les deux équations :

$$x - (x_p - y_p) = -x + (x_q + y_q)$$

En simplifiant cette équation, nous obtenons :

$$2x = (x_q + y_q) + (x_p - y_p)$$

Cela signifie que la coordonnée x de l'intersection est la moitié de la somme des coordonnées $x_q + y_q$ et $x_p - y_p$.

Maintenant, pour montrer que l'intersection se situe sur une ligne verticale, nous devons montrer que les coordonnées x sont égales. Pour ce faire, nous égalons les deux équations de départ :

$$x - (x_p - y_p) = -x + (x_q + y_q)$$

En simplifiant cette équation, nous obtenons :

$$2x = (x_q + y_q) + (x_p - y_p)$$

Divisons maintenant cette équation par 2 :

$$x = (x_q + y_q + x_p - y_p)/2$$

Cela montre que les coordonnées x de l'intersection sont égales à la moyenne des coordonnées $x_q + y_q$ et $x_p - y_p$

De même, pour montrer que l'intersection se situe sur une ligne horizontale, nous devons montrer que les coordonnées y sont égales. Pour ce faire, nous substituons la valeur de x trouvée dans l'une des équations de départ. Par exemple, prenons l'équation de la droite passante par w_p :

$$y = x - (x_p - y_p)$$

En substituant x, nous obtenons :

$$y = (x_q + y_q + x_p - y_p)/2 - (x_p - y_p)$$

En simplifiant cette équation, nous obtenons :

$$y = (x_q + y_q - x_p + y_p)/2$$

Cette équation montre que les coordonnées y de l'intersection sont égales à la moyenne des coordonnées $x_q + y_q$ et $x_p - y_p$.

Ainsi, en montrant que les coordonnées x et y de l'intersection sont égales à la moyenne des coordonnées des points donnés, et que la distance de Manhattan est paire, alors l'intersection des droites passantes par w_p avec une pente $+1$ et par w_q avec une pente -1 se situe à la fois sur une ligne horizontale et une ligne verticale de la grille.

■

12 Exploration de la Shift method

Dans cette section on va voir comment obtenir des dessins de lignes droites des graphes planaires sur une grille grâce à la mise en place de la Shift method, basé sur l'ordre canonique.

12.1 Dessin avec Shift method

Prenons l'exemple suivant de l'ordre canonique d'un graphe planaire maximal qu'on a vu précédemment 52, tous les sommets seront toujours situés sur une grille.

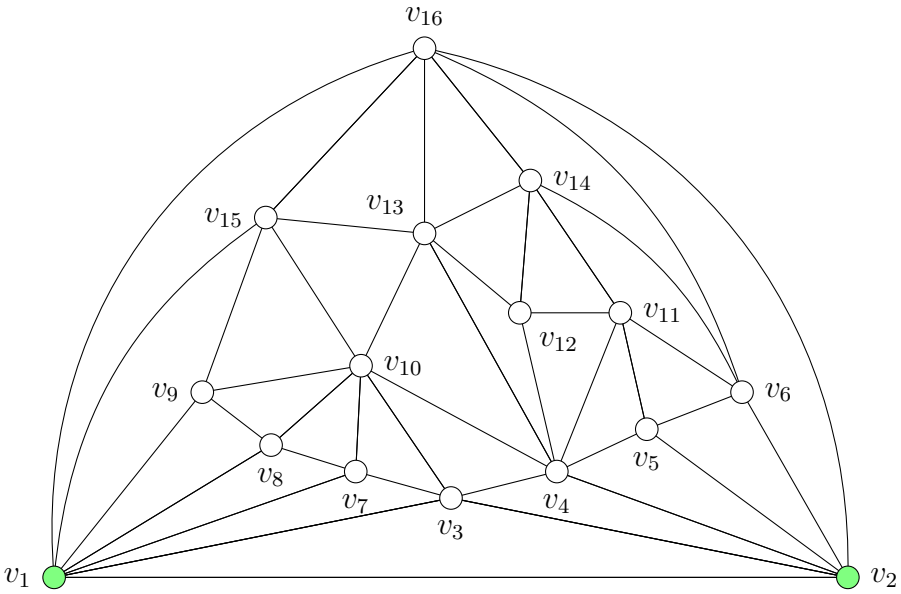


FIGURE 52 – Ordre canonique d'un graphe planaire maximal de la figure 25

Nous voulons insérer les sommets v_k, avec $k = 1, 2, 3....n$, donc à l'étape k on insère le sommet v_k. et pour les étapes 1, 2 et 3, nous avons déjà les trois points qui composent notre graphe, regardons l'étape k=3.

Étape k=3 :
Donc pour le graphe G_3 on place le sommet v_3 à (1,1), et on a v_1 à l'origine (0,0), et pour v_2 est à (2,0) selon la formule $(2k-4, 0)$ pour k=3 : $(2 \times k) - 4 = 6 - 4 = 2$, illustré à la figure 54.

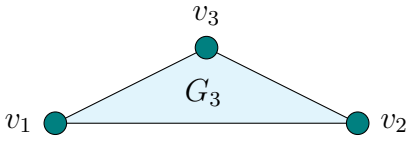


FIGURE 53 – Ordre canonique du graphe G_3

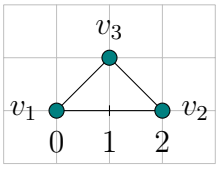


FIGURE 54 – Insertion du sommet v_3 avec Shift method

Dans le code Java figure en annexe, nous initialisons les valeurs suivantes :

v_1	v_2	v_3
(0,0)	(2,0)	(1,1)

Étape k=4 :
On passe au sommet v_4, il faut s'assurer que v_1 est à l'origine, et v_2 est à $2k-6$, et c'est le cas : *i.e,* $2\times 2-6=2$. Le sommet v_4 il a deux voisins, le v_3 et le v_2, qui sont montré sur le dessin, on doit d'abord se déplacer selon l'axe des abscisses, les sommets v_1 et v_3 se trouvent sur la première partie à gauche ils ne bougent pas, et le sommet v_2 se trouve dans la troisième partie à droite, il est donc déplacé de 2 unités vers la droite, voir figure suivante 68,

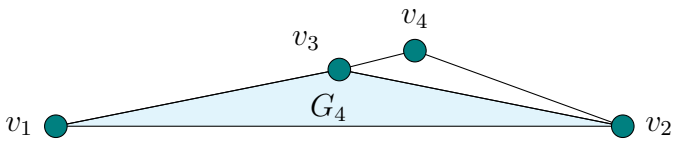


FIGURE 55 – Ordre canonique du graphe G_4

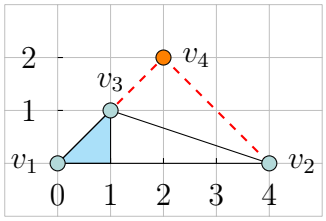


FIGURE 56 – Diagramme

Et on place le nouveau sommet à l'intersection de ces diagonales, le premier avec une pente de $+1$ passante par v_3 et le deuxième avec une pente de -1 passante par v_2, et comme ça nous aurons notre dessin G_4.
Les valeurs suivantes sont obtenues à partir du code Java présenté en annexe :

v_1	v_2	v_3	v_4
(0,0)	(4,0)	(1,1)	(2,2)

Étape k=5 :
Pour le sommet v_5 on l'insère entre les deux nouveaux voisins v_4 et v_2, et donc le v_3 et v_4 se trouvent à gauche, donc tout ce qui est à sa gauche n'est pas déplacé, mais le v_2 est déplacé de 2 unités vers la droite, et nous plaçons le v_5 sur l'intersection des deux droites le premier avec une pente de $+1$ passante par v_4 et le deuxième avec une pente de -1 passante par v_2, voir la figure suivante :

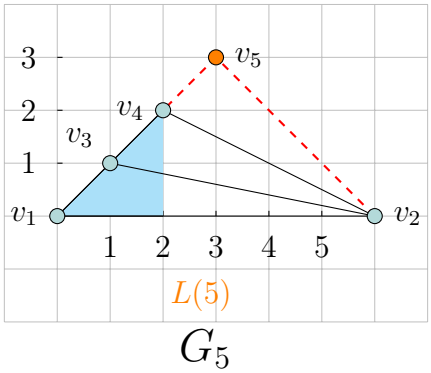
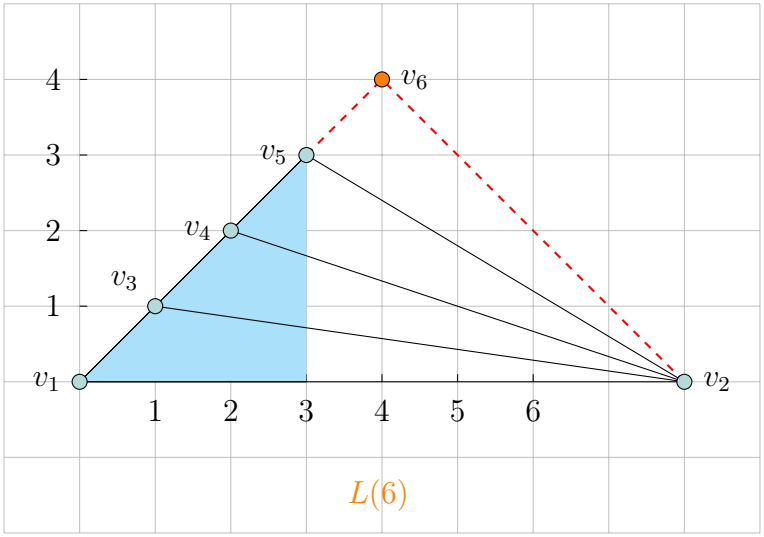


FIGURE 57 – Diagramme

Les valeurs suivantes sont obtenues à partir du code Java présenté en annexe :
Les points créés sont :

v_1	v_2	v_3	v_4	v_5
(0,0)	(6,0)	(1,1)	(2,2)	(3,3)

Étape k=6 :
On passe au sommet suivant est le v_6 il a deux voisins, le v_5 et le v_2, on doit d'abord se déplacer selon l'axe des abscisses, les sommets v_1, v_3, v_4 et v_5 se trouvent sur la première partie à gauche, ils ne bougent pas, et le sommet v_2 se trouve dans la troisième partie à droite, il est donc déplacé de 2 unités vers la droite, voir figure suivante,
et nous plaçons le v_6 sur l'intersection des deux droites la première avec une pente de $+1$ passante par v_4 et la deuxième avec une pente de -1 passante par v_2, voir la figure suivante :



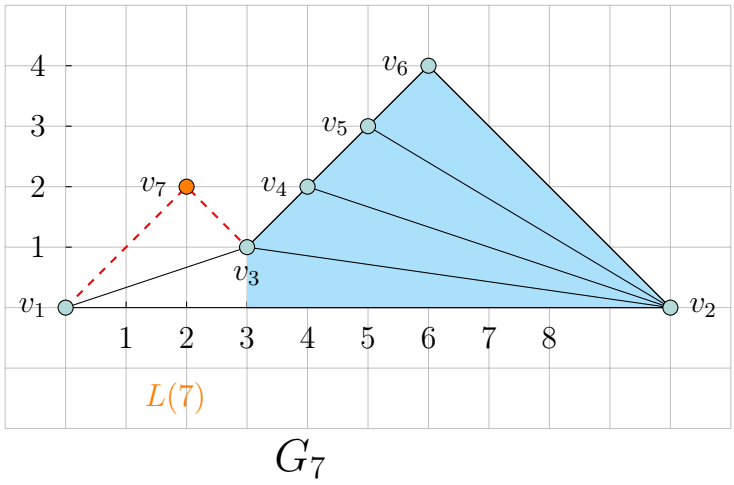
G_6

FIGURE 58 – Diagramme

Les valeurs suivantes sont obtenues à partir du code Java présenté en annexe :
Les points créés sont :

v_1	v_2	v_3	v_4	v_5	v_6
(0,0)	(8,0)	(1,1)	(2,2)	(3,3)	(4,4)

Étape k=7 :
Pour le sommet suivant v_7 il a deux voisins, le v_1 et le v_3, on doit d'abord se déplacer selon l'axe des abscisses, le sommet v_1 se trouve sur la première partie à gauche, il ne bouge pas, et les sommets v_2, v_3, v_4, v_5 et v_6 se trouvent dans la troisième partie à droite, ils sont déplacés de 2 unités vers la droite, voir figure suivante,
et nous plaçons le v_7 sur l'intersection des deux droites : la première avec une pente de $+1$ passante par v_1 et la deuxième avec une pente de -1 passante par v_3, voir la figure suivante :



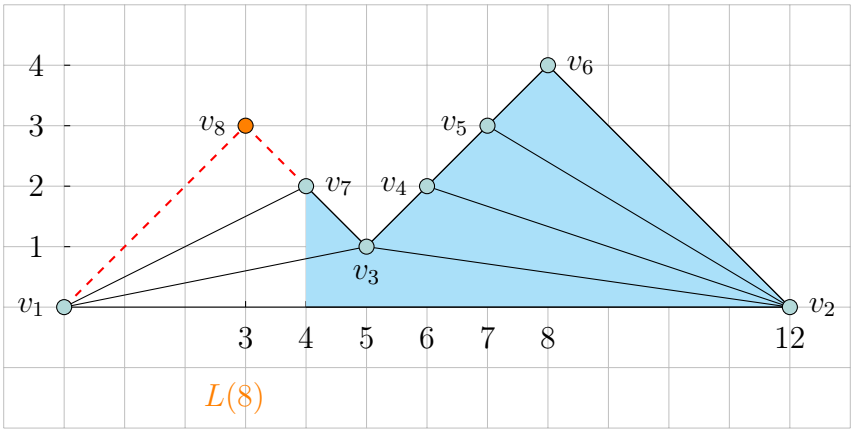
G_7

FIGURE 59 – Diagramme

Les valeurs suivantes sont obtenues à partir du code Java présenté en annexe :
Les points créés sont :

v_1	v_2	v_3	v_4	v_5	v_6	v_7
(0,0)	(10,0)	(3,1)	(4,2)	(5,3)	(6,4)	(2,2)

Étape k=8 :
Pour le sommet suivant v_8 il a deux voisins, le v_1 et le v_7, on doit d'abord se déplacer selon l'axe des abscisses, le sommet v_1 se trouve sur la première partie à gauche, il ne bouge pas, et les sommets v_2, v_3, v_4, v_5, v_6 et v_7 se trouvent dans la troisième partie à droite, ils sont décalés de 2 unités vers la droite, voir figure suivante,
et nous plaçons le v_8 sur l'intersection des deux droites : la première avec une pente de $+1$ passante par v_1 et la deuxième avec une pente de -1 passante par v_7, voir la figure suivante :



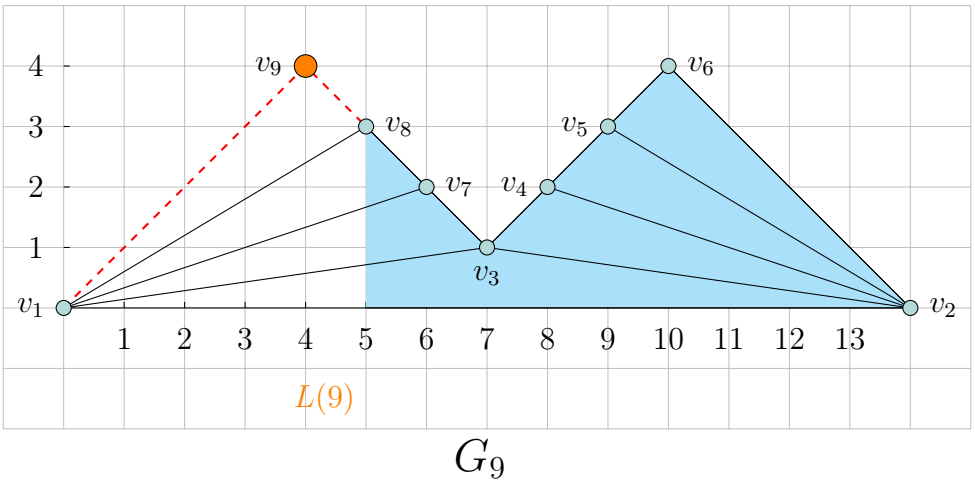
G_8

FIGURE 60 – Diagramme

Les valeurs suivantes sont obtenues à partir du code Java présenté en annexe :
Les points créés sont :

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
(0,0)	(12,0)	(5,1)	(6,2)	(7,3)	(8,4)	(4,2)	(3,3)

Étape k=9 :
Pour le sommet suivant v_9 il a deux voisins, le v_1 et le v_8, on doit d'abord se déplacer selon l'axe des abscisses, le sommet v_1 se trouve sur la première partie à gauche, il ne bouge pas, et les sommets v_2, v_3, v_4, v_5, v_6, v_7 et v_8 se trouvent dans la troisième partie à droite, ils sont décalés de 2 unités vers la droite, voir figure suivante,
et nous plaçons le v_9 à l'intersection des deux droites : la première avec une pente de $+1$ passante par v_1 et la deuxième avec une pente de -1 passante par v_8, voir la figure suivante :



G_9

FIGURE 61 – Diagramme

Les valeurs suivantes sont obtenues à partir du code Java présenté en annexe :
Les points créés sont :

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9
(0,0)	(14,0)	(7,1)	(8,2)	(9,3)	(10,4)	(6,2)	(5,3)	(4,4)

Étape k=10 :
Pour le sommet suivant v_{10} il a deux voisins, le v_9 et le v_4, on doit d'abord se déplacer selon l'axe des abscisses, le sommet v_1 et v_9 se trouvent sur la première partie à gauche, il ne bouge pas, les sommets v_3, v_7 et v_8 sont au milieu ils bougent de $+1$ unité vers la droite, et les sommets v_2, v_4, v_5 et v_6 se trouvent dans la troisième partie à droite, ils sont décalés de 2 unités vers la droite, voir figure suivante,
et nous plaçons notre v_{10} sur l'intersection des deux droites : la première avec une pente de $+1$ passante par v_9 et la deuxième avec une pente de -1 passante par v_4, voir la figure suivante :

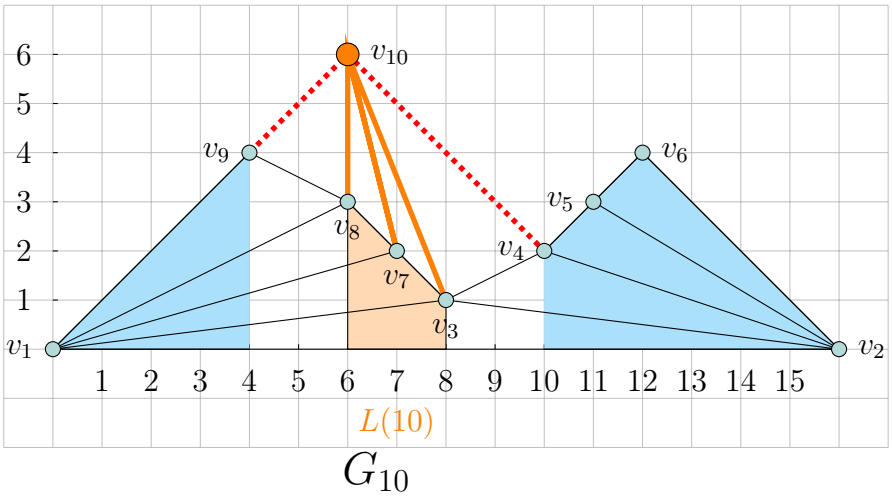


FIGURE 62 – Diagramme

Les valeurs suivantes sont obtenues à partir du code Java présenté en annexe :
Les points créés sont :

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}
(0,0)	(16,0)	(8,1)	(10,2)	(11,3)	(12,4)	(7,2)	(6,3)	(4,4)	(6,6)

Étape k=11 :
Pour le sommet suivant v_{11} il a deux voisins, le v_4 et le v_6, on doit d'abord se déplacer selon l'axe des abscisses, les sommets v_1, v_3, v_4, v_7, v_8, v_9 et v_{10} se trouvent sur la première partie à gauche, il ne bouge pas, le sommet v_5 est au milieu il bouge de $+1$ unité vers la droite, et les sommets v_2 et v_6 se trouvent dans la troisième partie à droite, ils sont décalés de 2 unités vers la droite, voir figure suivante,
et nous plaçons notre v_{11} sur l'intersection des deux droites : la première avec une pente de $+1$ passante par v_4 et la deuxième avec une pente de -1 passante par v_6, voir la figure suivante :

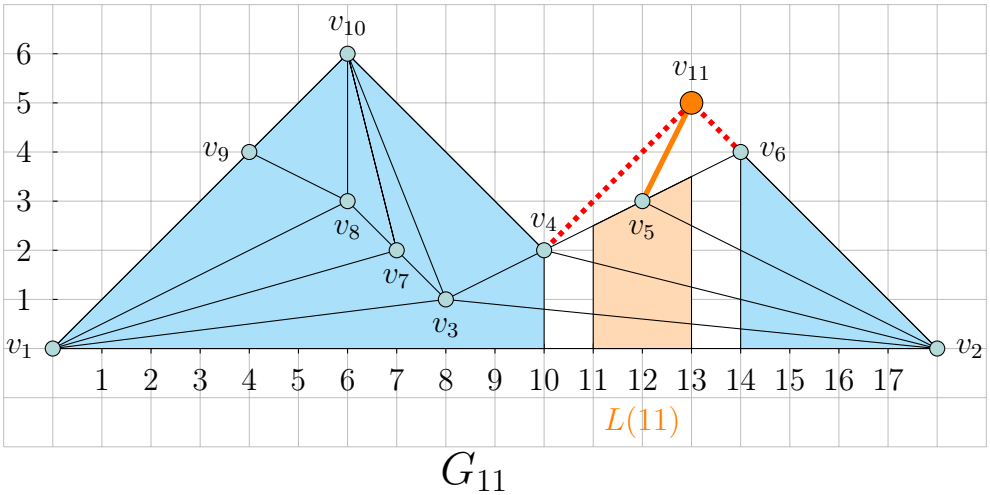
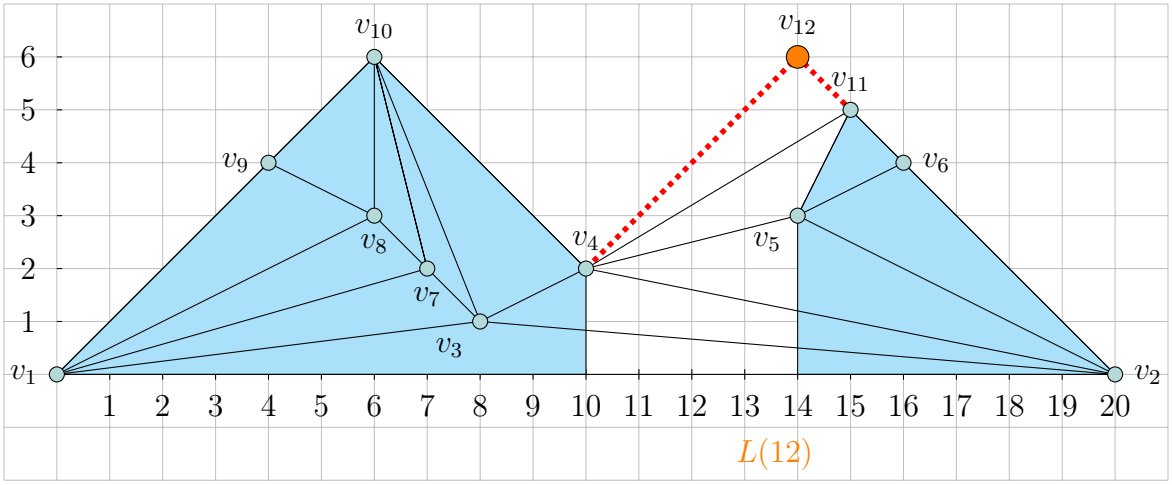


FIGURE 63 – Diagramme

Les valeurs suivantes sont obtenues à partir du code Java présenté en annexe :
Les points créés sont :

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}
(0,0)	(18,0)	(8,1)	(10,2)	(12,3)	(14,4)	(7,2)	(6,3)	(4,4)	(6,6)	(13,13)

Étape k=12 :
Pour le sommet suivant v_{12} il a deux voisins, le v_4 et le v_{11}, on doit d'abord se déplacer selon l'axe des abscisses, les sommets v_1, v_3, v_4, v_7, v_8, v_9 et v_{10} se trouvent sur la première partie à gauche, il ne bouge pas, et les sommets v_2, v_5, v_6 et v_{11} se trouvent dans la troisième partie à droite, ils sont décalés de 2 unités vers la droite, voir figure suivante,
et nous plaçons notre v_{12} sur l'intersection des deux droites : la première avec une pente de $+1$ passante par v_4 et la deuxième avec une pente de -1 passante par v_{11}, voir la figure suivante :



G_{12}

FIGURE 64 – Diagramme

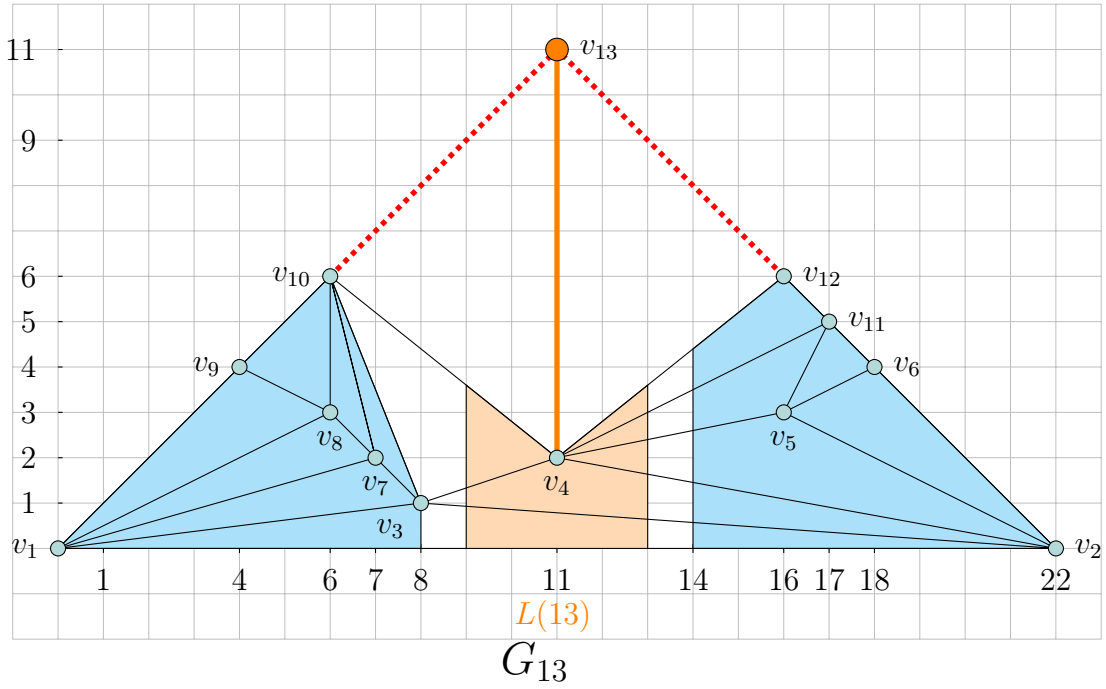
Les valeurs suivantes sont obtenues à partir du code Java présenté en annexe :
Les points créés sont :

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}
(0,0)	(20,0)	(8,1)	(10,2)	(14,3)	(16,4)	(7,2)	(6,3)	(4,4)	(6,6)	(15,5)	(14,6)

Étape k=13 :

Pour le sommet suivant v_{13} il a deux voisins, le v_{10} et le v_{12}, on doit d'abord se déplacer selon l'axe des abscisses, les sommets v_1, v_3, v_7, v_8, v_9 et v_{10} se trouvent sur la première partie à gauche, il ne bouge pas, le sommet v_4 au milieu bouge de $+1$ unité vers la droite, et les sommets v_2, v_5, v_6, v_{11} et v_{12} se trouvent dans la troisième partie à droite, ils sont décalés de 2 unités vers la droite, voir figure suivante,

et nous plaçons notre v_{13} sur l'intersection des deux droites : la première avec une pente de $+1$ passante par v_{10} et la deuxième avec une pente de -1 passante par v_{12}, voir la figure suivante :



G_{13}

FIGURE 65 – Diagramme

Les valeurs suivantes sont obtenues à partir du code Java présenté en annexe :

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
(0,0)	(22,0)	(8,1)	(11,2)	(16,3)	(18,4)	(7,2)	(6,3)

v_9	v_{10}	v_{11}	v_{12}	v_{13}
(4,4)	(6,6)	(17,5)	(16,6)	(11,11)

Étape k=14 :

Pour le sommet suivant v_{14} il a deux voisins, le v_{13} et le v_6, on doit d'abord se déplacer selon l'axe des abscisses, les sommets v_1, v_3, v_4, v_7, v_8, v_9, v_{10} et v_{13} se trouvent sur la première partie à gauche, il ne bouge pas, les sommets v_5, v_{11} et v_{12} sont au milieu donc bougent de $+1$ unité vers la droite, et les sommets v_2 et v_6 se trouvent dans la troisième partie à droite, ils sont décalés de 2 unités vers la droite, voir figure suivante,

et nous plaçons notre v_{14} sur l'intersection des deux droites : la première avec une pente de $+1$ passante par v_{13} et la deuxième avec une pente de -1 passante par v_6, voir la figure suivante :

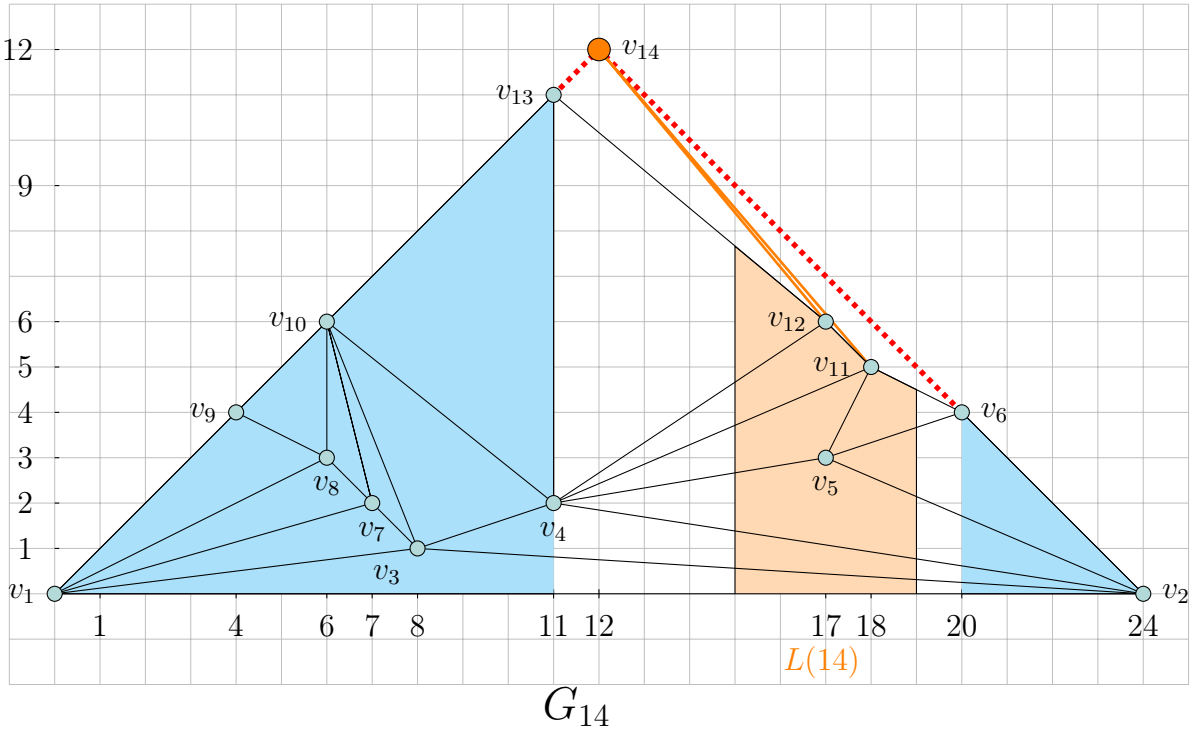


FIGURE 66 – Diagramme

Les valeurs suivantes sont obtenues à partir du code Java présenté en annexe :

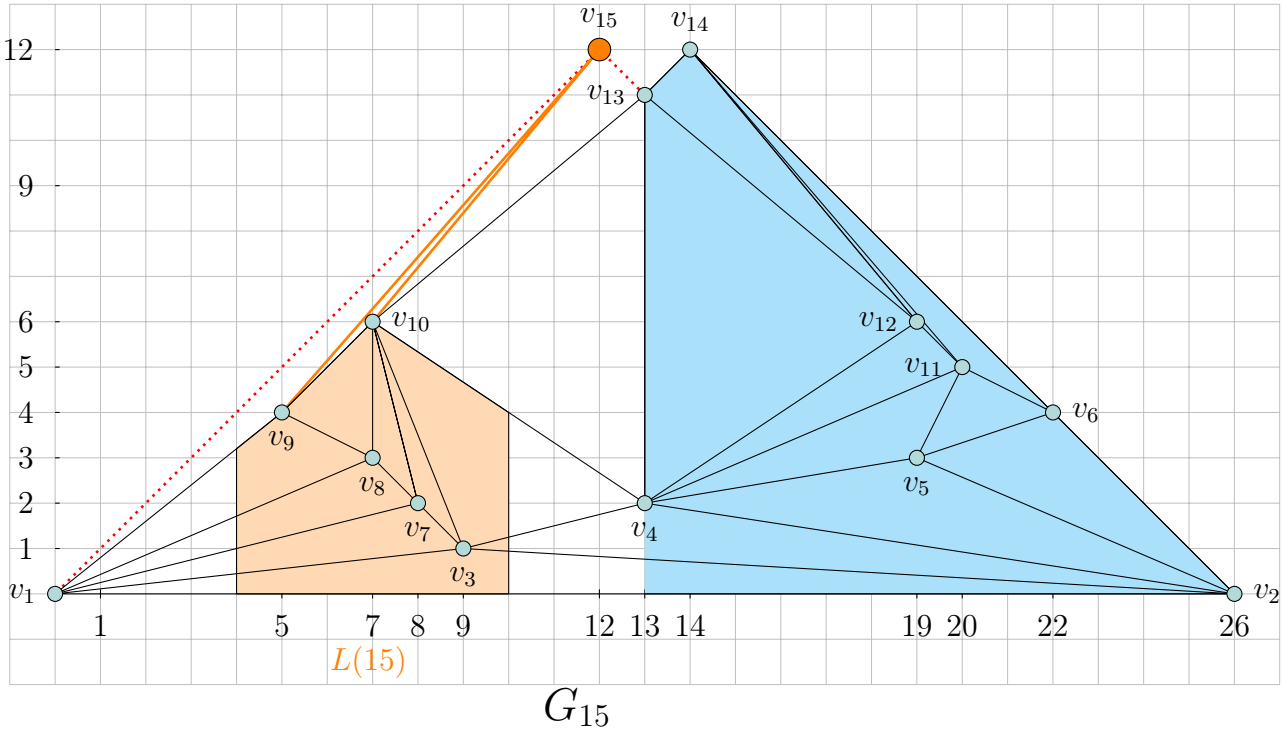
v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
(0,0)	(24,0)	(8,1)	(11,2)	(17,3)	(20,4)	(7,2)	(6,3)

v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}
(4,4)	(6,6)	(18,5)	(17,6)	(11,11)	(12,12)

Étape k=15 :

Pour le sommet suivant v_{15} il a deux voisins, le v_1 et le v_{13}, on doit d'abord se déplacer selon l'axe des abscisses, le sommet v_1 se trouve sur la première partie à gauche, il ne bouge pas, les sommets v_3, v_7, v_8 v_9 et v_{10} sont au milieu donc bougent de $+1$ unité vers la droite, et les sommets v_2, v_4, v_5, v_6, v_{11}, v_{12}, v_{13} et v_{14} se trouvent dans la troisième partie à droite, ils sont décalés de 2 unités vers la droite, voir figure suivante,

et nous plaçons notre v_{15} sur l'intersection des deux droites : la première avec une pente de $+1$ passante par v_1 et la deuxième avec une pente de -1 passante par v_{13}, voir la figure suivante :



G_{15}

FIGURE 67 – Diagramme

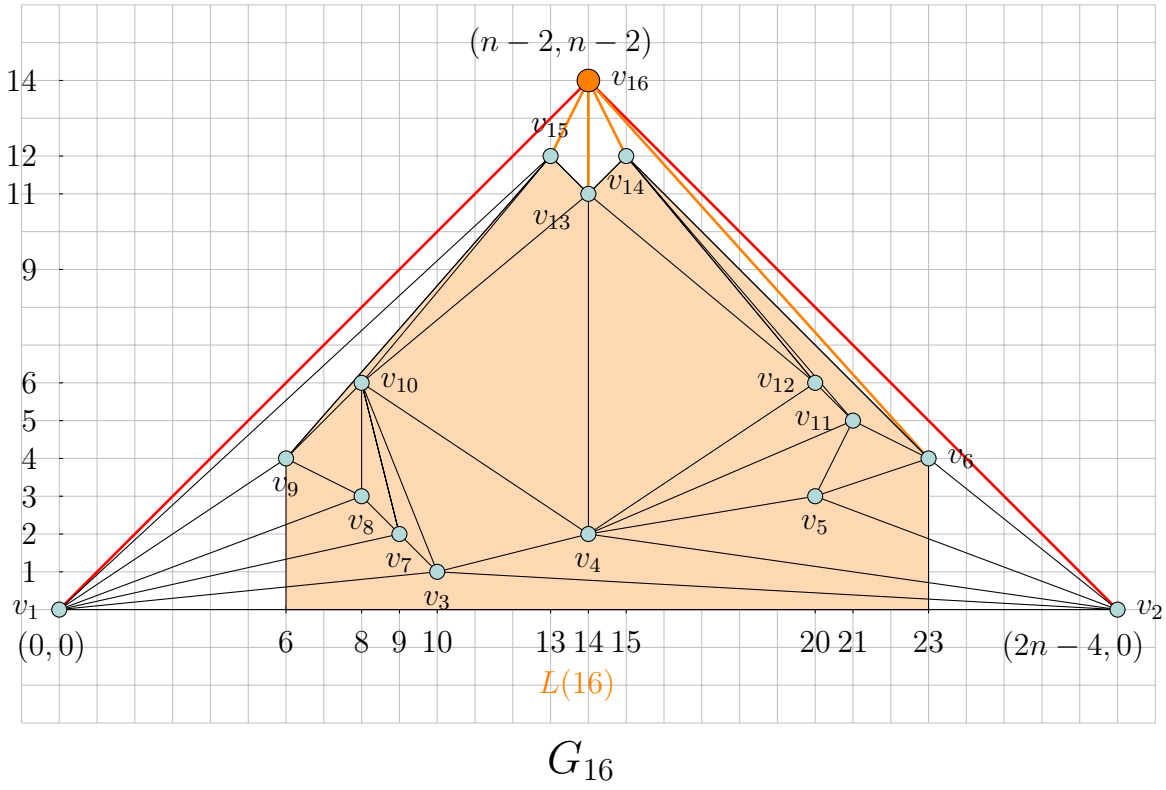
Les valeurs suivantes sont obtenues à partir du code Java présenté en annexe :

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
(0,0)	(26,0)	(9,1)	(13,2)	(19,3)	(22,4)	(8,2)	(7,3)

v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}
(5,4)	(7,6)	(20,5)	(19,6)	(13,11)	(14,12)	(12,12)

Étape k=16 :

Pour le sommet suivant v_{16}, il devient un peu plus grand, il a deux voisins, le v_1 et le v_2, on doit d'abord se déplacer selon l'axe des abscisses, le sommet v_1 se trouve sur la première partie à gauche, il ne bouge pas, les sommets v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}, v_{14} et v_{15} sont au milieu donc bougent de $+1$ unité vers la droite, et le sommet v_2 se trouve dans la troisième partie à droite, il est décalés de 2 unités vers la droite, voir figure suivante,
et nous plaçons notre v_{16} sur l'intersection des deux droites : la première avec une pente de $+1$ passante par v_1 et la deuxième avec une pente de -1 passante par v_2, voir la figure suivante :
Donc on peut avoir des arbres, qui contiennent tous les sommets de notre zone orange,



G_{16}

FIGURE 68 – Diagramme

Les valeurs suivantes sont obtenues à partir du code Java présenté en annexe :

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
(0,0)	(28,0)	(10,1)	(14,2)	(20,3)	(23,4)	(9,2)	(8,3)

v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}
(6,4)	(8,6)	(21,5)	(20,6)	(14,11)	(15,12)	(13,12)	(14,14)

Et voila ainsi nous avons dessiné notre graphe avec Shift method,

12.2 Analyse de la planarité

Dans la section précédente, nous avons vu un exemple avec toutes les étapes d'un dessin d'un graphe avec la **Shift method**.

Avons-nous un dessin d'un graphe planaire ?

Mais avant de répondre à la question, on va examiner un exemple pour nous faire une idée, et pour cela, nous allons marquer la partie au milieu en couleur orange et l'appeler l'ensemble (L) de v_k, on note L(v_k).

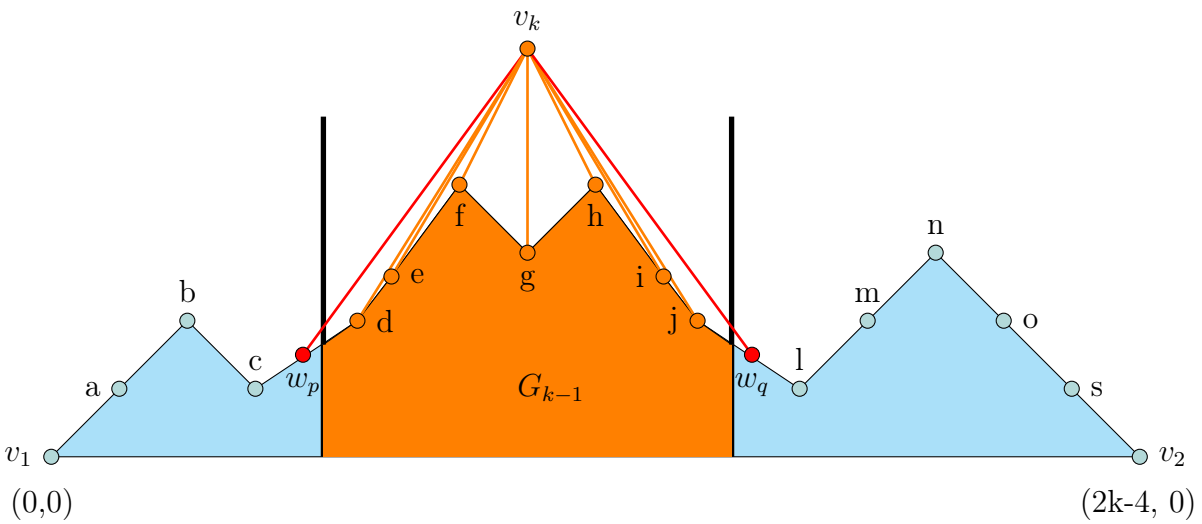


FIGURE 69 – Exemple d'insertion du sommet v_k avec marquage de l'ensemble L(v_k)

Et quand on fait notre décalage au lieu de se déplacer vers la gauche et la droite, on va juste déplacer tout ce qui est à l'intérieur de +1 unité vers la droite, et tout ce qui est à droite de l'intérieur de +2 unités vers la droite, voir la figure suivat 70.

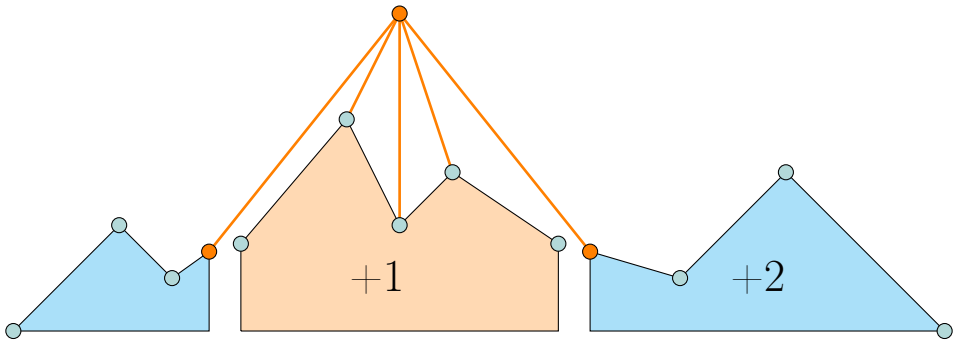


FIGURE 70 – Illustration du décalage dans la méthode shift

Preuve de planarité

Supposons que nous ayons un graphe G_{k-1}, auquel nous ajoutons un nouveau sommet v_k sur la face externe. Dans G_{k-1}, tous les sommets w_i existent avec $i \in \{1, 2, \dots t\}$ tel que $1 < \cdots < p < \cdots < q < \cdots < t$. Nous avons ainsi les sommets v_1, v_2, ..., ainsi que les voisins de gauche $w_p, w_{p+1}, \dots, w_q, \dots$ et ainsi de suite.
Dans ce contexte, certains sommets qui étaient précédemment sur la face externe ne le sont plus maintenant. Nous appelons ces sommets "couverts". Lorsqu'un sommet est retiré de la face externe, cela ne peut se produire qu'une seule fois, ce qui signifie que chaque sommet interne est couvert exactement une fois.
Si nous examinons la relation entre le sommet couvrant v_k et les sommets couverts (entourés en bas dans la figure illustrative 71), nous remarquons que ces derniers peuvent couvrir d'autres sommets en dessous. Cependant, chaque sommet n'est couvert qu'une seule fois, et chaque sommet a donc une seule arête entrante. Ainsi, cette relation de couverture définit un arbre au sein de notre graphe.
Dans les sous-graphes de notre graphe, c'est une **forêt**. Le sommet v_k couvre ceux qui sont colorés en orange, et chacun de ces derniers peut couvrir d'autres sommets. Cependant, les sommets qui ont un indice inférieur à w_p, ainsi que les sommets qui ont un indice supérieur à w_q, ne sont pas couverts par v_k.
Cela donne lieu à plusieurs **arbres** distincts, formant une forêt, comme le montre la figure illustrative 71. Néanmoins, les sommets dont l'indice est inférieur à w_p et ceux dont l'indice est supérieur à w_q ne sont pas couverts par v_k.

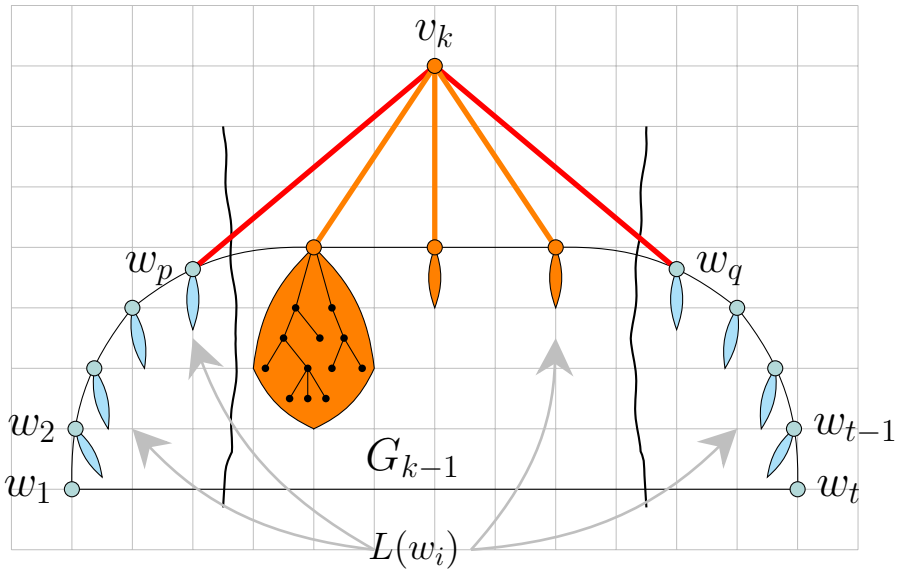


FIGURE 71 – Illustration de la relation de couverture et de la forêt de sous-graphes

que devons-nous faire, pour être sûrs que notre dessin est planaire ?
Après avoir effectué notre découpe, nous devons garantir que les nouvelles arêtes n'intersectent aucune des anciennes structures présentes, et que ces anciennes structures demeurent

disposées de manière planaire.
Pour la première étape, les éléments en orange sont clairement planaires. En effet, si l'on observe la figure 69, les arêtes oranges présentent une pente de ± 1. Les trois successeurs, c'est-à-dire les sommets d, e et f montant de w_p, avaient une pente de $+1$ précédemment, et ils ont été déplacés. Par conséquent, nous ne pouvons pas avoir d'intersection en v_k. De même, les prédécesseurs, à savoir les sommets h, i et j descendant de w_q, avaient une pente de -1, et ils ont également été déplacés. Par conséquent, nous ne pouvons pas non plus avoir d'intersection en v_k.
En ce qui concerne les autres arêtes à l'intérieur de la structure, elles présentent une pente plus importante, soit supérieure à $+1$, soit inférieure à -1. Par conséquent, nous ne pouvons pas avoir de croisements au sein de la partie orange.

Observations

1. Chaque sommet interne est couvert exactement une fois.
2. La relation de couverture définit un arbre dans G
3. Une forêt dans G_i, $1 \leq i \leq n-1$.

Pour cela, on utilise une proposition général, c'est-à-dire on examine tous les ensembles de recouvrement $L(w_i)$, que nous avons définis avant, et prouver que nous pouvons les déplacer d'une certaine manière, et tout reste toujours planaire, donc on va prouver le théorème suivant :

Theorem 12.1. *Supposons que G_k soit dessiné de façon planaire.*
Soit $0 < \delta_1 < \delta_2 < ... < \delta_t \in \mathbb{N}$.
Si nous décalons chaque $L(w_i)$ de δ_i vers la droite, alors le dessin reste planaire <*Kindermann, 2010*>.

Nous avons quelques variables croissantes qui vont de δ_1 à δ_t, donc δ_1 est égale au moins zéro, δ_2 est égale au moins 1, δ_3 est égale au moins 2, et ainsi de suite, et maintenant si on prends tous les $L(w_i)$, et que nous décalons chacun d'entre eux, de δ_i vers la droite alors le dessin reste planaire,

Exemple : On décale w_1 de 0, w_2 de 1, le suivant de 2, v_k et tous les oranges au milieu par 3, et touts les suivants par 4, w_t aussi par 4, alors c'est planaire.
Dans notre étape inductive, on a $(0 < \delta_1 < \delta_2 < ... < \delta_t)$ uniquement pour les 0, 1 et 2, mais il nous faut une forme générale, pour la preuve inductive,

Démonstration du théorème 12.1

Démonstration du théorème 12.1. Initialement, nous avons G_3 et, par suite, nous obtenons un graphe. On distingue précisément trois ensembles. Tant que nous procédons à des dépla-

cements selon la méthode décrite, la planarité est préservée. Par conséquent, nous pouvons supposer par induction que si le lemme est vrai pour G_{k-1}, il l'est aussi pour G_k.

Comment cela se manifeste-t-il ? Si pour G_k, nous obtenons un nombre correspondant à tous les sommets sur la face externe (les sommets de w_1 à w_p, et les sommets de w_q à w_t), nous pouvons alors appliquer un décalage à G_{k-1}. Plus précisément, nous décalons tous les sommets avant w_p et après w_q dans le dessin de G_{k-1} par exactement ce nombre $(0 < \delta_1 < \delta_2 < ... < \delta_t)$. De plus, nous décalons tous les sommets du milieu par le nombre que nous avons obtenu pour v_k. Par induction, notre graphe demeure donc un dessin planaire.

Qu'en est-il de G_k ? Nous avons appliqué le décalage à tous les sommets du graphe et il ne reste plus qu'à déplacer également v_k vers la droite. Toutefois, ce déplacement est effectué avec tout ce qui se trouve en dessous (la partie orange du milieu), garantissant que rien ne change. Par conséquent, nous ne pouvons pas introduire de nouveaux croisements dans la partie orange du milieu.

Quant aux deux arêtes $(v_k,\ w_p)$ et $(v_k,\ w_q)$, elles demeurent clairement à l'extérieur du dessin central, assurant ainsi que l'ensemble reste planaire. ■

En conclusion, après avoir démontré le théorème 12.1, nous sommes en mesure d'affirmer que nous pouvons appliquer nos décalages à chaque étape sans introduire de nouveaux croisements. Par conséquent, l'ensemble de la méthode nous fournit un dessin planaire.

13 Analyse de complexité et implémentation d'algorithme Shift method

Nous avons prouvé que la méthode du décalage nous donne un dessin sur planaire simple, sur une grille de taille polynomiale, maintenant on a besoin de connaître la durée d'exécution.

13.1 Pseudo-code

Pour connaître la durée d'exécution, nous allons jeter un coup d'œil au pseudo-code.

Algorithm 2: Pseudo-code de Shift method

Entrée: (Graphe, son ordre canonique)
Sortie: (Graphe décalé par la méthode shift)

1 **Début**
  //Soit $v_1, \ldots, v_n$ l'ordre canoniquedu graphe G
2 **Pour** (i :=1 **jusqu'à** 3) **faire**
3 L(v_i) ← $\{v_i\}$
4 P(v_1) ← (0, 0) ; P(v_2) ← (2, 0) ; P(v_3) ← (1, 1) ;
5 **Pour** (i :=4 **jusqu'à** n) **faire**
  //Soit w_1=v_1, w_2, ..., w_{t-1}, w_t=v_2 désigne la frontière de G_{i-1}, et notons w_p, ..., w_q les voisins de v_i
6 **Pour** $(\forall v \in \bigcup_{j=p+1}^{q-1} L(w_j))$ **faire**
  //La complexité de cette boucle $\theta(n^2)$
7 x(v) ← x(v) + 1
8 **Pour** $(\forall v \in \bigcup_{j=q}^{t} L(w_j))$ **faire**
  //La complexité de cette boucle $\theta(n^2)$
9 x(v) ← x(v) + 2
10 P(v_i) ← intersection des diagonales ±1 passante par P(w_p) et P(w_q)
11 L(v_i) ← $\bigcup_{j=p+1}^{q-1}$ L(w_j) ∪ $\{v_i\}$

13.2 Rétrospective

Nous supposons que nous avons déjà l'ordre canonique, nous savons que nous pouvons faire cette partie en un temps linéaire.

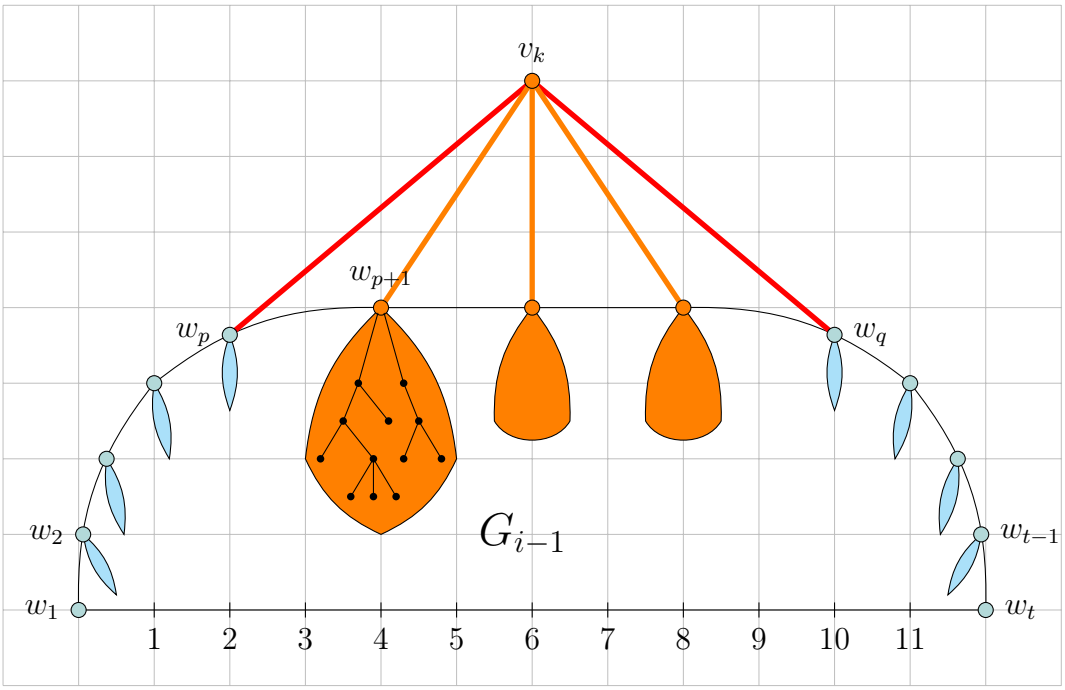


FIGURE 72 – Illustration de la position du sommet v_k sur le graphe G_{k-1}, et l'ensemble $\mathrm{L}(v_i)$ sous forme d'une structure arborescente

Dans un premier lieu on commence à dessiner les trois premiers sommets. Puis dessiner les autres, donc supposons que nous avons déjà dessiné $Gi-1$, Nous avons les points w_1 jusqu'à w_t sur la face externe, et nous avons les voisins de v_i qui vont de w_p jusqu'à w_q voir la figure 69. Et donc pour faire notre décalage, on doit trouver l'ensemble $\mathrm{L}(v_i)$ et le déplacer vers la droite, et ce qui est à sa droite, également le déplacer vers la droite, voir la figure 70.

Nous avons donc trois ensembles, le premier contient tout ce qui est à gauche, on ne va rien faire pour cette partie. Comme illustré dans la photo 70 ;
Et nous avons la partie centrale, et pour cette partie, nous regardons le $\mathrm{L}(v_i)$ de tous les sommets entre $p+1$ et $q-1$, et tous pour ces sommets nous les décalons de $+1$ vers la droite, Comme illustré dans la photo 70 ;
Puis nous avons tous les sommets de droite, qui commence par w_q jusqu'à w_t, et tous ces ensembles nous les déplaçons par $+2$ unités vers la droite, Comme illustré dans la photo 70.
Et c'est comme ça qu'on obtient notre décalage.
Maintenant on veut placer le v_k, et pour cela, on doit trouver l'intersection de la droite passante par w_p avec une pente $+1$ et de la droite passante par w_q avec une pente -1 illustré dans la figure 72. et nous avons déjà prouvé que cela nous donne un dessin planaire ;

La seule chose qui reste à faire, est de trouver l'ensemble $\mathrm{L}(v_i)$. Mais c'est encore une fois, c'est très simple, nous prenons juste l'union des ensembles de tous les sommets dans la partie orange, et nous itérons. Et donc c'est ça notre pseudo code 2.

13.3 Analyse de complexité

Dans le pseudo code 2 on remarque que tout est constant en dehors des boucles, donc nous devons seulement regarder ce qui se passe à l'intérieur des boucles.
Nous supposons que le graphe G est déjà triangulé et intégré dans le plan, et qu'un ordre canonique $\pi = \{v_1, v_2, \ldots, v_n\}$ de G est donné. Nous considérons la famille d'ensembles $L(w_i), L(w_2), \ldots, L(w_t)$ pour les sommets externes $w_1, w_2, \ldots, w_t$ du graphe G_k comme une forêt F dans G_k composée d'arbres $L(w_i), L(w_2), \ldots, L(w_t)$ enracinés aux sommets $w_1, w_2, ..., w_t$ comme illustré dans la figure suivante 71. Pour bien représenter les ensembles, la foret et l'arbre binaire, nous prenons notre exemple d'avant mais sans le sommet v_{16} car le sommet v_{16} est sommet couvrant de la forêt, ainsi on aura notre graphe G_{15}, la forêt F est représentée par des lignes épaisses et solides comme le montre la figure 73

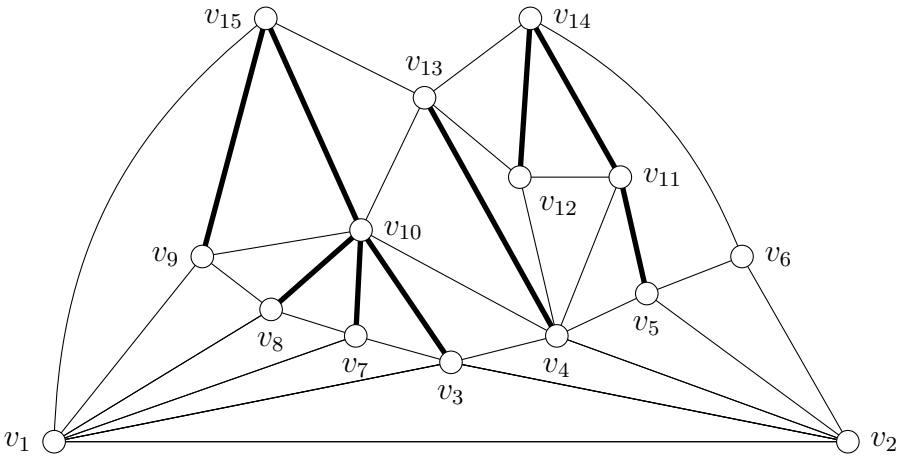


FIGURE 73 – Illustration de la forêt F dans le graphe G_{15} par des lignes épaisses et solides

Pour G_{15} la forêt F et est également représentée dans la figure suivante 74.

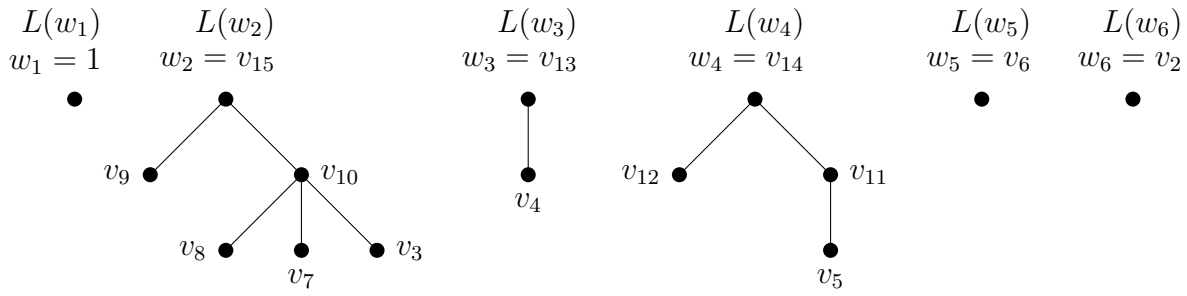


FIGURE 74 – Illustration de la forêt F du graphe G_{15}

Les enfants de la racine w_i d'un arbre $L(w_i)$ sont les sommets que w_i couvre, c'est-à-dire ses voisins qui quittent le cycle externe lorsque w_i est installé. La forêt F est représentée par un arbre binaire T comme illustré dans la figure suivante 75.

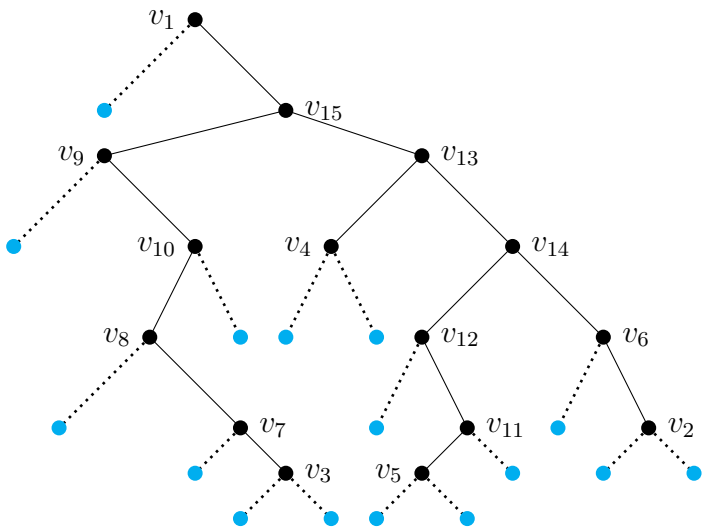


FIGURE 75 – Illustration de l'arbre T du graphe G_{15}

La racine de T est $w_1(= v_1)$. Le fils droit de w_1 est w_2, le fils droit de w_2 est w_3, et ainsi de suite. L'ensemble $L(w_1)$ est composé de w_i et de tous les nœuds du sous-arbre gauche de w_i dans T. Ainsi, le sous-arbre de T enraciné à w_i est composé des sommets dans $\bigcup_{j\geq 1} L(w_j)$. Dans le sous-arbre gauche de T enraciné à w_i, le fils gauche de w_i est le fils le plus à gauche de w_i dans l'arbre $L(w_i)$ (s'il existe), le fils droit du fils gauche dans T est son prochain frère à droite dans l'arbre $L(w_i)$ (s'il existe), le fils droit du fils droit du fils gauche dans T est son prochain frère suivant à droite dans l'arbre $L(w_i)$ (s'il existe), et ainsi de suite. Pour G_3 dans la figure 76(a), son arbre binaire T est illustré dans la figure 76(b).

FIGURE 76 – (a) Graphe G_3, et (b) l'arbre binaire du graphe G_3

Étant donné que v_k est intégré en un point $\mu(P_1, P_2)$ d'intersection des deux diagonales, la première avec une pente $+1$ passant par w_p et la deuxième avec une pente -1 passant par w_q , on a le point :

$$(1) : \mu(P_1, P_2) = [\frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p)), \frac{1}{2}(x(w_q) + x(w_p) + y(w_q) - y(w_p))]$$

Nous avons :

$$(2) : x(v_k) = \frac{1}{2}[x(w_q) + x(w_p) + y(w_q) - y(w_p)]$$

$$(3) : y(v_k) = \frac{1}{2}[x(w_q) + x(w_p) + y(w_q) - y(w_p)]$$

$$(4) : x(vk) - x(wp) = \frac{1}{2}[x(wq) - x(wp) + y(wq) - y(wp)]$$

La première observation cruciale est que, lorsque nous intégrons v_k, il n'est pas nécessaire de connaître la position exacte de w_p et w_q. Si nous connaissons uniquement leurs coordonnées y et leurs coordonnées x relatives, c'est-à-dire $x(w_q) - x(w_p)$, alors selon l'équation (3), nous pouvons calculer $y(v_k)$ et selon l'équation (4), nous pouvons calculer la coordonnée x de v_k relative à w_p, c'est-à-dire $x(v_k) - x(w_p)$.

13.4 Implémentation de l'algorithme Shift method

Pour chaque sommet $v \neq v_1$, le décalage x de v est défini comme $\Delta x(v) = x(v) - x(w)$, où w est le parent de v dans T. Plus généralement, si w est un ancêtre de v, alors le décalage x entre w et v est $\Delta x(w, v) = x(v) - x(w)$. Avec chaque sommet v, nous stockons les informations suivantes :

— $left(v)$ = le fils gauche de v dans T ;
— $right(v)$ = le fils droit de v dans T ;
— $\Delta x(v)$ = le décalage x de v par rapport à son parent dans T ;
— $y(u)$ = la coordonnée y de v.

L'algorithme se compose de deux phases : Dans la première phase, nous ajoutons de nouveaux sommets un par un, et à chaque ajout d'un sommet, nous calculons son décalage x et sa coordonnée y, et mettons à jour les décalages x de un ou deux autres sommets. Dans la deuxième phase, nous parcourons l'arbre T et calculons les coordonnées x finales en accumulant les décalages.

La première phase est mise en œuvre comme suit. Tout d'abord, nous initialisons les valeurs stockées à v_1, v_2 et v_3 comme suit voir figure 76 :

— $\Delta x(v_1) = 0$; $y(v_1) = 0$; $right(v_1) = v_3$; $left(v_1) = nil$;
— $\Delta x(v_3) = 1$; $y(v_3) = 1$; $right(v_3) = v_2$; $left(v_3) = nil$;
— $\Delta x(v_2) = 1$; $y(v_2) = 0$; $right(v_2) = nil$; $left(v_2) = nil$.

Ensuite, nous intégrons les autres sommets, un par un, comme suit :

Algorithm 3: Algorithm de Shift method

1 **Pour** (k :=4 **jusqu'à** 4) **faire**
2 **Début**
3 Soit $w_1, w_2, \ldots, w_t$ le cycle externe $C_o(G_{k-1})$ de G_{k-1} ; voir figure 72
4 Soit $w_p, w_{p+1}, \ldots, w_q$ les voisins de v_k sur $C_o(G_{k-1})$;
5 Augmenter le décalage de $w_{p+1}, \ldots, w_{q-1}$ de $+1$ vers la droite ;
6 Augmenter le décalage de $w_q, \ldots, w_t$ de $+2$ vers la droite ;
   //Calcules:
7 $\Delta x(w_p, w_q) = \Delta x(w_{p+1}) + \Delta x(w_{p+2}) + ... + \Delta x(w_q)$;
8 $\Delta x(v_k) = \frac{1}{2}[\Delta x(w_p, w_q) + y(w_q) - y(w_p)]$; *cf.* (4)
9 $y(v_k) = \frac{1}{2}[\Delta x(w_p, w_q) + y(w_q) + y(w_p)]$; *cf.* (3)
10 $\Delta x(w_q) = \Delta x(w_p, w_q) - \Delta x(v_k)$;
11 **Si** $(p+1) \neq q$ **Alors**
12 $\Delta x(w_{p+1}) = \Delta x(w_{p+1}) - \Delta x(v_k)$;
13 $right(w_p) = v_k$ et $right(v_k) = w_q$;
14 **Si** $(p+1) \neq q$ **Alors**
15 $left(v_k) = w_{p+1}$ et $right(w_{q-1}) = nil$;
16 **Sinon**
17 $left(v_k) = nil$;

On observe dans les formules de calcules, que les formules de la ligne (5), la ligne (6), la ligne (8), la ligne (9), et la ligne (10) sont clairement constants, mais la ligne (7) prend plus de temps, donc il suffit simplement de calculer combien de temps avons-nous réellement besoin pour connaître la durée d'exécution, pour cela, nous devons prendre la distance x de w_q et de tous les sommets à l'intérieur. Mais les sommets à l'intérieur ce sont des sommets déjà traités. donc nous ne devons les regarder qu'une seule fois, car ils disparaissent de la face extérieure, donc nous ne regardons qu'un seul sommet plus la somme des sommets qui sont enlevés de la face extérieure, et ceux à chaque étape, au total nous examinons n sommets ici à la ligne ligne (7) :

$$\Delta_x(w_{p+1}) + \cdots + \Delta_x(w_q)$$

La figure 72 illustre la construction de T pour G_k à partir de T pour G_{k-1} par l'algorithme ci-dessus.

Dans la deuxième phase : nous calculons la coordonnée $x(v_i)$ pour chaque sommet v_i dans G. Soit Q le chemin de la racine v_1 à v_i dans l'arbre T. Ensuite, $x(v_i)$ est égal à la somme des déplacements horizontaux ($\Delta(x)$) pour chaque sommet x situé sur le chemin Q. On peut calculer $x(v_i)$ pour tous les sommets v_i en invoquant la procédure Accumulate-Offset$(v_1, 0)$; la procédure Accumulate-Offset est la suivante :

Algorithm 4: Procédure Accumulate-Offset

Entrée: (v : sommet ; δ : entier)

Début
  Si $(p+1) \neq q$ **Alors**
    $\Delta(u) = \Delta(u) + \delta$;
    Accumulate-Offset$(left(u), Ax(u))$;
    Accumulate-Offset$(droit(v), Ax(v))$;

13.5 Complexité

Il est clair que $x(v_i) = \Delta x(v_i)$ pour chaque sommet v_i dans G.
La première phase prend un temps linéaire, car l'ajout d'un sommet v_k prend au plus un temps $\theta(d(v_k))$, $d(v_k)$ représente le degré du sommet v_k dans le graphe G. Le degré d'un sommet fait référence au nombre de voisins qu'il possède, c'est-à-dire le nombre d'arêtes qui y sont incidentes.
La deuxième phase, c'est-à-dire Accumulate-Offset, prend un temps proportionnel au nombre de nœuds dans T. Ainsi, l'algorithme prend un temps linéaire au total <N. Takao, R. Saidur, 2017>.

Et nous obtenons ainsi notre temps d'exécution et notre résultat principal, à savoir que tout graphe planaire à n sommets possède un dessin planaire de ligne droite de taille $(2n-4) \times (n-2)$, peut-être calculer en temps linéaire, cet algorithme est à la base de nombreux algorithmes de dessin de graphes, et même pour les graphes planaires, il y a eu quelques améliorations.
Tout d'abord, en 1996, **Kant** (**Chrobak**) a prouvé que si le graphe d'entrée est trié connecté, alors nous pouvons également dessiner toutes les faces convexes, en conservant la même surface, et ce dans le même temps d'exécution, et un an plus tard, avec **Chrobak**, ils ont même amélioré la limite de surface à $(n-2) \times (n-2)$, Ce dessin peut être calculé en un temps $\theta(n)$, et cette limite de surface a encore été améliorée, par **Brandenburg** en 2008, à $(\frac{4}{3}n \times \frac{2}{3}n)$, Ce dessin peut être calculé en un temps $\theta(n)$, il est donc un peu plus large, mais il n'est pas si haut, mais le $(n-2) \times (n-2)$. est n^2, alors que celui-ci n'est que $\frac{8}{9}n^2$, c'est donc un peu mieux et la meilleure borne inférieure connue à ce jour est $\frac{2}{3}n^2$, cela réduit l'écart entre eux, mais il reste encore à déterminer si la réponse correcte est $\frac{2}{3}$, $\frac{8}{9}$ ou quelque chose entre les deux <Kindermann, 2010>.

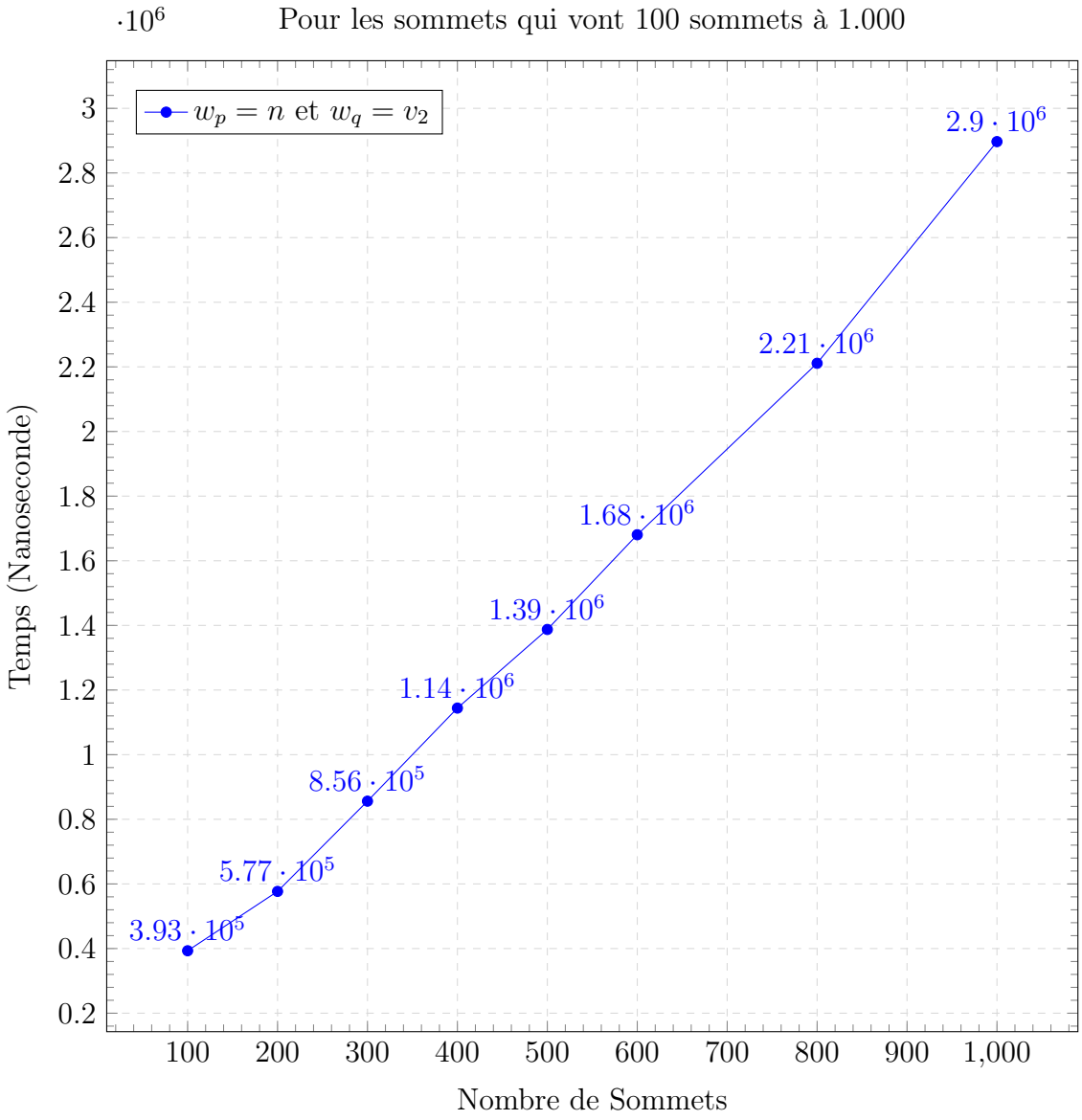
14 Étude expérimentale

Dans le cadre de cette étude expérimentale, nous avons réalisé des expériences visant à évaluer les performances de l'algorithme shiftmethode. Nous avons utilisé trois ensembles de données, chacun composé d'une série de valeurs de taille de graphe (nombre de sommets) et de temps d'exécution. Ces ensembles de données nous permettent d'observer comment les performances de l'algorithme évoluent en fonction de la taille des graphes (nombre de sommets). Dans cette section, nous présenterons ces ensembles de données, ainsi que la méthodologie mise en place pour mesurer les temps d'exécution de l'algorithme.

14.1 Ensembles de données

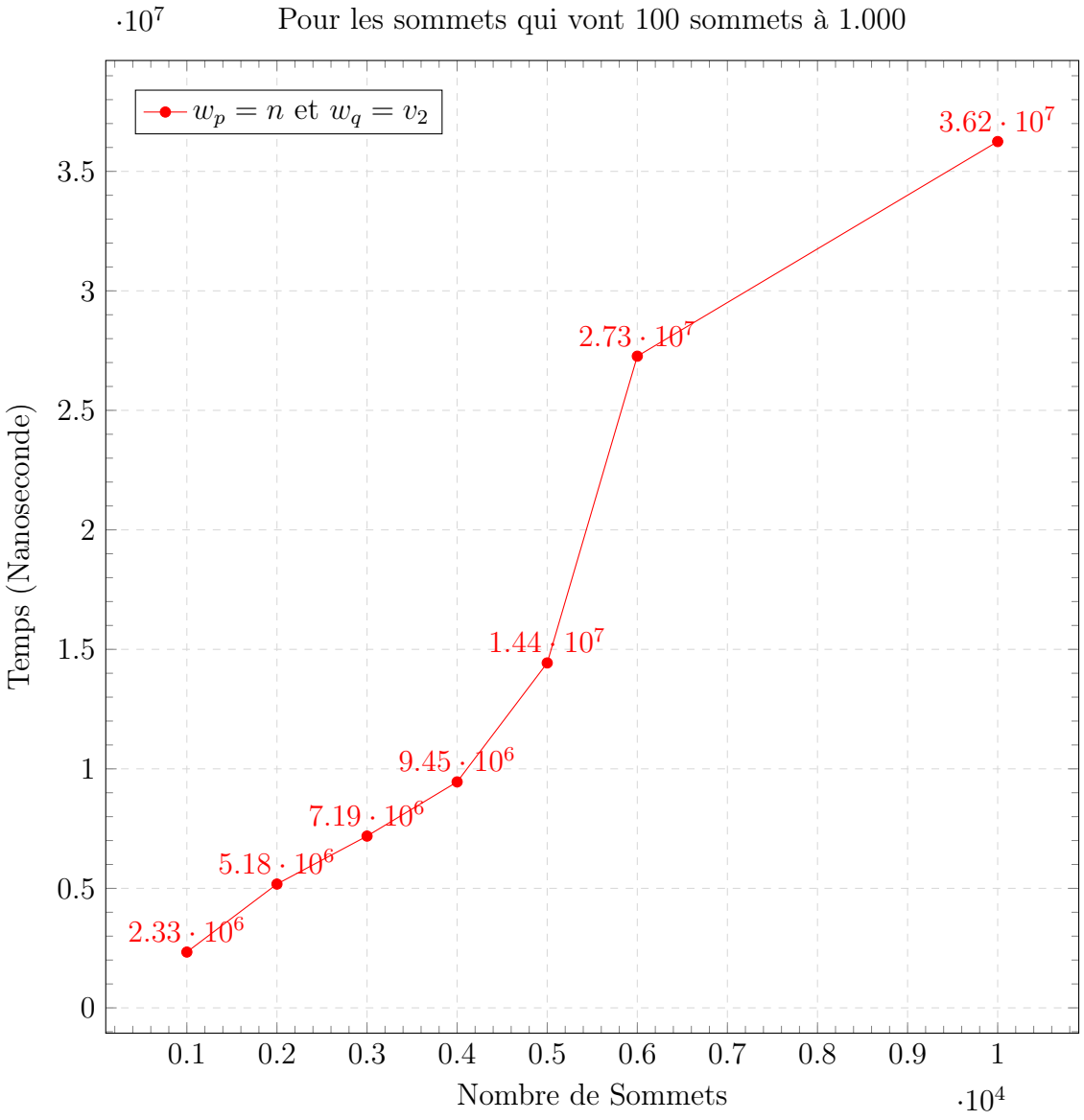
Pour chaque ensemble de données, nous avons mesuré le temps d'exécution de l'algorithme pour chaque taille de graphe. Les résultats sont visualisés à l'aide de graphes, mettant en évidence la relation entre la taille du graphe et le temps d'exécution.

Première partie Pour les petites tailles des graphe avec un nombre de sommets qui vont 100 à 1.000 sommets répété $\times 100$


$\cdot 10^6$
Pour les sommets qui vont 100 sommets à 1.000
$w_p = n$ et $w_q = v_2$
$3.93 \cdot 10^5$
$5.77 \cdot 10^5$
$8.56 \cdot 10^5$
$1.14 \cdot 10^6$
$1.39 \cdot 10^6$
$1.68 \cdot 10^6$
$2.21 \cdot 10^6$
$2.9 \cdot 10^6$
Temps (Nanoseconde)
Nombre de Sommets
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
2.2
2.4
2.6
2.8
3
100
200
300
400
500
600
700
800
900
1,000

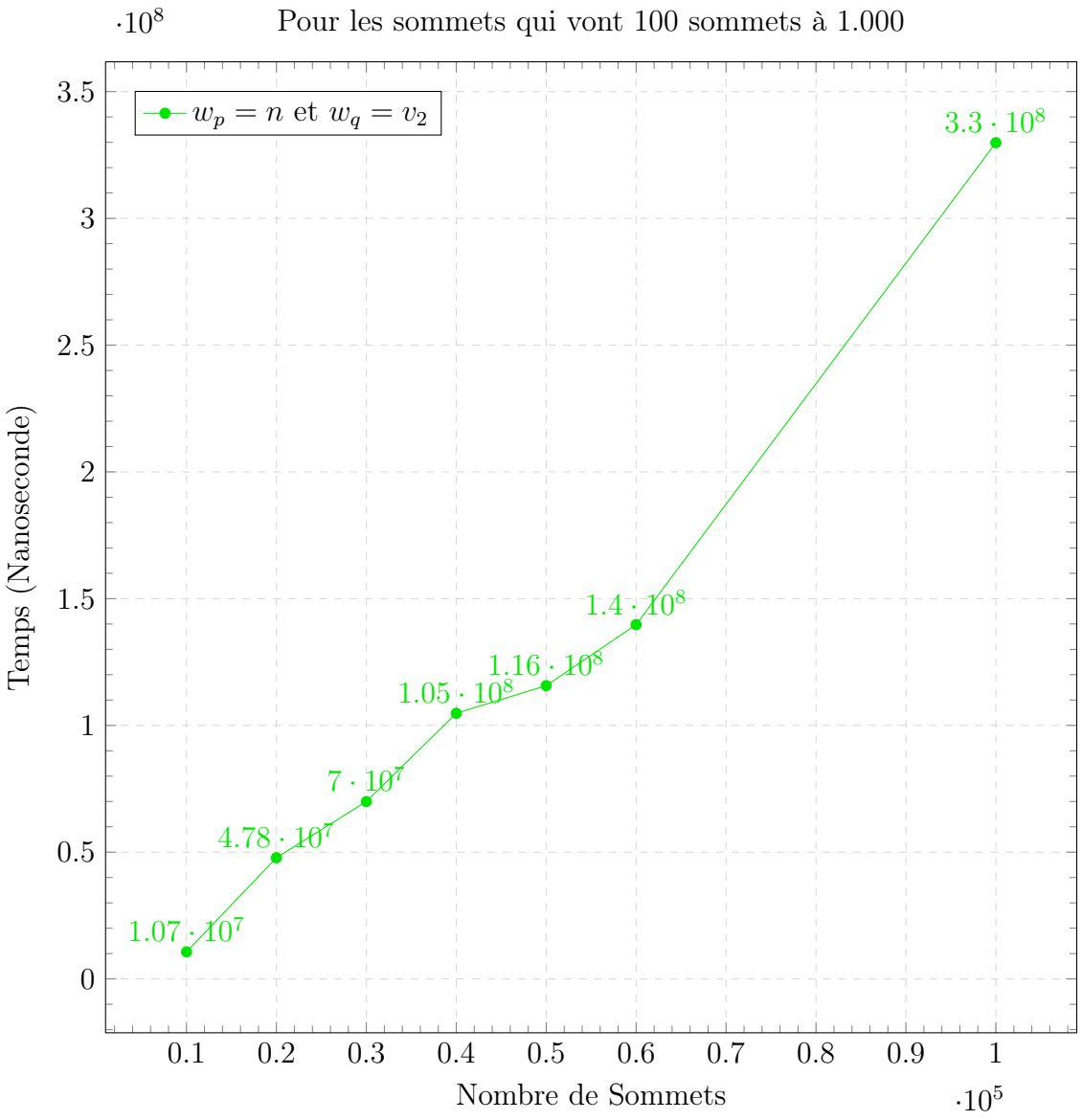
Deuxième partie

Pour les petites tailles des graphe avec un nombre de sommets qui vont 1.000 à 10.000 sommets répété $\times 50$



Troisième partie

Pour les petites tailles des graphe avec un nombre de sommets qui vont 10.00 à 100.000 sommets répété $\times 10$



Dans cette étude expérimentale, nous avons évalué les performances de l'algorithme shiftmethode pour le dessin de graphes planaires en ligne droite. Trois ensembles de tests ont été réalisés en variant la taille des graphes : les petites tailles (100 à 10.000 sommets), les tailles moyennes (1.000 à 10.000 sommets) et les grandes tailles (10.000 à 100.000 sommets).

Les résultats obtenus ont confirmé la complexité linéaire de l'algorithme shiftmethode, en démontrant une relation directe entre la taille de l'entrée (le nombre de sommets) et le temps d'exécution. Lorsque la taille de l'entrée était multipliée approximativement par 2, le temps d'exécution était également multiplié par environ 2, confirmant ainsi la caractéristique linéaire de l'algorithme.

Ces résultats expérimentaux renforcent donc l'idée que l'algorithme shiftmethode est efficace pour le dessin de graphes planaires en ligne droite, en offrant une complexité linéaire proche de la théorie. Cette méthode présente un intérêt pratique pour le dessin de graphes de différentes tailles, fournissant des résultats satisfaisants en termes de temps d'exécution.

En conclusion, l'étude expérimentale confirme l'efficacité de l'algorithme shiftmethode pour le dessin de graphes planaires en ligne droite, en respectant une complexité linéaire. Les résultats obtenus soutiennent son utilisation pratique dans le domaine, en offrant des performances cohérentes et prévisibles pour des graphes de diverses tailles.

15 Méthodologie

Dans la méthodologie de cette étude, nous détaillerons le contexte expérimental dans lequel les performances de l'algorithme shiftmethode ont été évaluées, Données d'entrée.

15.1 Description de l'environnement de travail

Nous avons utilisé un système informatique spécifique, dont les caractéristiques sont les suivantes :

1. **Configuration matérielle**
   — **Processeur** : 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
   — **Type de système** : Système d'exploitation 64 bits, processeur x64
   — **Mémoire RAM** : 16,0 Go (15,8 Go utilisables)
2. **Système d'exploitation**
   — Windows 10 Enterprise
3. **Environnement de développement**
   — **Langage de programmation** : Java
   — **Bibliothèque système JRE (Java Runtime Environment)** : JavaSE-1.8
   — **IDE (Integrated Development Environment)** : Eclipse

Par la suite, nous poursuivrons en exposant en détail le code implémenté, en mettant l'accent sur les algorithmes clés utilisés et les structures de données employées pour résoudre efficacement le problème du dessin de graphes planaires en ligne droite.

15.2 Données d'entrée

Dans le cadre de la méthodologie, les données d'entrée jouent un rôle crucial dans l'évaluation des performances de l'algorithme shiftmethode.

Premier code

Dans le premier code utilisé, nous avons adopté une approche interactive où l'utilisateur entre les points un par un, permettant ainsi de définir l'ordre canonique du graphe. Plus précisément,le programme demande à l'utilisateur de spécifier les valeurs de w_p (point inférieur) et w_q (point supérieur) par rapport au contour du graphe. Ensuite, le graphe est décalé en conséquence, et l'algorithme calcule les intersections des deux droites : l'une ayant une pente de $+1$ et passante par w_p, et l'autre ayant une pente de -1 et passante par w_q. Ces intersections forment un triangle isocèle avec un angle droit en v_k, le sommet calculé.

Le premier code a également pour fonction d'afficher les résultats obtenus. Il affiche les valeurs des points avant et après le décalage, ainsi que les arêtes qui ont été utilisées pour les

dessins dans la sous-section 9.1 (Dessin avec shiftmethode) de la section 9 (Appliquer shiftmethode). Ces informations sont essentielles pour comprendre les manipulations effectuées sur le graphe et analyser les résultats obtenus pour pouvoir les dessiner.

Deuxième code

Dans le cadre de l'optimisation du code, plusieurs améliorations ont été apportées pour améliorer les performances. Voici un résumé des principales optimisations effectuées :

1. 1. Utilisation d'une structure de données plus efficace : Au lieu d'utiliser une ArrayList pour stocker les points, une HashMap a été utilisée qui offre une complexité de recherche de $\theta(1)$, ce qui est plus rapide que la recherche linéaire dans une liste. Cela permet d'accélérer les opérations de recherche et d'accès aux points.
2. 2. Modification des méthodes de gestion des points : Les méthodes getPoints() et addPoint() ont été modifiées pour utiliser les fonctionnalités de la HashMap. La méthode getPoints() renvoie désormais une nouvelle ArrayList de points à partir des valeurs de la HashMap. La méthode addPoint() utilise la méthode put() de la HashMap pour ajouter le point avec son nom comme clé.
3. 3. Amélioration de la méthode printCoordonnee() : A été mise à jour pour obtenir directement le nom et les coordonnées de chaque point, améliorant ainsi l'efficacité de l'affichage.
4. 4. Révision de la méthode getNeighbors() : A été révisée pour tenir compte de l'utilisation de la HashMap. Les indices de matrice ont été ajustés pour correspondre aux noms des points, assurant ainsi une correspondance correcte entre les points et leurs voisins.
5. Pour stocker les points. nous avons modifié la classe GraphNode pour utiliser une HashMap au lieu d'une ArrayList pour stocker les points. Par exemple :

```
1  private Map<String, MyPoint> points;
2
3  //lors de l ajout d un point, nous pouvons l ajouter a la
       HashMap :
4  public void addPoint(MyPoint point) {
5          points.add(point);
6          points.put(point.getName(), point);
7  }
8
9  // Et on peut obtenir un point par son nom avec une
       complexite O(1) :
10 public MyPoint getPointByName(String name) {
11         return points.get(name);
12 }
```

Listing 1 – Extrait de code de la classe GraphNode.

Cela nous permettrait de récupérer un point par son nom en temps constant $\theta(1)$ au lieu du temps linéaire $\theta(n)$.

6. Pour améliorer les performances, évitez les appels redondants aux mêmes méthodes de calcul. Par exemple, dans la méthode addPointBetween(), l'appel à la méthode graph.getPointByName(firstPointName) et graph.getPointByName(secondPointName) plusieurs fois. Au lieu de cela, on peut appeler ces méthodes une fois, stocker les résultats dans des variables et les réutiliser par la suite pour éviter les calculs redondants.
7. Dans le code certaines parties, comme dans la méthode apresWq(), y a une boucle while(true) avec une condition de sortie break à l'intérieur. Cela a étè simplifié en une boucle do-while.
8. Commenter les opérations inutiles comme afficher le graphe, et les arrêtes.
9. Pour l'ordre canonique, et lors les valeurs de w_p et w_q sont générées aléatoirement à mesure que le nombre de sommets augmente. Voici le code qui génère ces valeurs dynamiquement :

```
1         int wp, wq;
2         do {
3                 wp = random.nextInt(tour - 1);
4                 wq = wp + 1 + random.nextInt(tour - wp -
                     1);
5         } while (wp == wq || wq == tour - 1);
6         String wpName = "v" + Integer.toString(wp+1);
7         String wqName = "v" + Integer.toString(wq+1);
```

Listing 2 – Extrait de code représentant la génération d'un ordre canonique aléatoire.

Cependant, il s'avère qu'avec cette approche, il existe un risque d'obtenir des cordes dans le graphe. Pour remédier à cela, nous avons décidé de prendre $w_p = n$ et $w_q = v_2$.

L'algorithme écrit en langage Java figure en annexe sur ce lien Github :
https://github.com/Kazzoul-Youness/StraightLine-ShiftMethod

15.3 Méthode de mesure

Pour mesurer les performances de l'algorithme, nous avons utilisé des compteurs de temps en nanosecondes. Ce qui permet une évaluation précise de l'efficacité de l'algorithme en fonction de la taille du graphe. Voici la méthodologie utilisée :

Mesure et calcul des temps d'exécution

Nombre total de tests :
On a effectué des tests pour la plage de 100 à 1 000 sommets, avec une répétition de chaque test 100 fois. Pour la plage de 1 000 à 10 000 sommets, on a effectué chaque test 10 fois. Dans la plage de 10 000 à 100 000 sommets, on a effectué chaque test 10 fois.

Calcul du temps d'exécution :

On a mesuré le temps d'exécution pour chaque test en utilisant la variable "executionTimesTest" pour stocker les temps d'exécution individuels de chaque test. Pour chaque test, on a calculé le temps d'exécution en sommant les temps d'exécution individuels des sommets générés à l'aide de la méthode de décalage. Calcul du total d'exécutions :

On a calculé le total d'exécutions en sommant les temps d'exécution de tous les tests en utilisant la variable "sumTest". Dans cette variable, on a stocké la somme des temps d'exécution de tous les tests effectués pour les différentes plages de nombres de sommets.

```
1  int NBtest = 10;
2  int NbSommets = 40000;       // exemple pour 40.000 sommets
3  long[] executionTimesTest = new long[NBtest];
4  long[] executionTimesTour = new long[NbSommets + 1];
5  long startTimeTest, endTimeTest, startTimeTour,
       endTimeTour;
6
7  for (int test = 0; test < NBtest; test++) {
8      //Chaque sommet qui va etre placer par Shift method, on
           genere son ordre canonique, et seulement a ce moment
           qu'on commence le calcule du temps.
9
10     // Debut du calcul du temps en nanosecondes
11     long startTime = System.nanoTime();
12
13     // Operations de calcul de l'algorithme
14     //1. Decalage Milieu +1 unite vers la droite
15     //2. Decalage Droit +2 unites vers la droite
16     //3. Calcule les coordonnees x et y du point d'intersection
17     //4. Creer le point d'intersection avec un nom qui
           s'incremente automatiquement
18     //5. Ajouter le point d'intersection au graphe
19     //6. Ajouter le point vk a la liste sur le contour
20
21     // Fin du calcul du temps en nanosecondes
22     endTimeTour = System.nanoTime();
23     executionTimesTour[tour] = endTimeTour - startTimeTour;
24 }
25     long sumTour = 0;
26     for (int i = 3; i <= NbSommets; i++) {
27         sumTour += executionTimesTour[i];
28     }
29     executionTimesTest[test] = +sumTour;
```

```
30      }
31      long avg, sumTest=0;
32      for (int i = 0; i < executionTimesTest.length; i++) {
33          sumTest += executionTimesTest[i];
34      }
35      avg = sumTest/NBtest;
36      System.out.println("Temps pour "+NbSommets+" sommets = "+
            avg + " Nano");
37      System.out.println("("+NbSommets+","+ avg + ")");
```

Listing 3 – Extrait de code représentant la Méthode utilisée pour le calcul des mesures.

L'algorithme écrit en langage Java figure en annexe sur ce lien Github :
https://github.com/Kazzoul-Youness/StraightLine-ShiftMethod

15.4 Limitations et problèmes rencontrés

Plusieurs erreurs ont était rencontré, notamment :

Une erreur **java.lang.ArrayIndexOutOfBoundsException** s'est produite, indiquant une tentative d'accès à un indice en dehors des limites du tableau. Dans le contexte de l'algorithme Shift method, cette erreur était liée à l'initialisation des trois premiers sommets, qui constitue une étape spécifique de l'algorithme. Pour remédier à cette erreur, une correction a été apportée en ajustant la boucle principale pour commencer à partir de l'indice 3, afin d'accéder uniquement aux indices valides du tableau executionTimes.

L'erreur **java.lang.OutOfMemoryError : Java heap space** s'est produite, indiquant que le programme a tenté d'allouer plus de données à la mémoire heap de la JVM (Java Virtual Machine) qu'elle n'est capable de contenir. Plusieurs approches ont été envisagées pour remédier à cette situation. Dans ce cas spécifique, une tentative a été faite en exécutant le programme sur un système Ubuntu Linux tout en augmentant la taille de la mémoire à 2 Go à l'aide de la commande java -Xmx2g MonProgramme. Cependant, il a été observé que la fermeture des fenêtres et des autres programmes consommateurs de ressources était nécessaire pour libérer suffisamment de mémoire et permettre une exécution correcte du programme. Il convient de souligner l'importance de l'optimisation des ressources et de la gestion efficace de la mémoire pour éviter les erreurs de type OutOfMemoryError.

Erreur java.lang.StackOverflowError
at ShiftMethod.moveChildren(ShiftMethod.java :247) : Cette erreur s'est produite généralement dans le cas de récursion non contrôlée, où une fonction moveChildren s'appelle elle-même indéfiniment, ce qui a conduit à un débordement de la pile d'appels, celle-ci s'appelle récursivement pour chaque enfant du parent. Et pour résoudre cette erreur, une

approche consiste à limiter le nombre maximal de sommets à 100 000, ce qui correspond à la taille maximale du graphe d'entrée. Cette limitation permet de contrôler la profondeur de la récursion et d'éviter le dépassement de la pile d'appels.

En prenant en compte ces erreurs rencontrées lors de l'exécution de l'algorithme Shift method et les solutions qui ont été apportées pour les résoudre, il est possible d'améliorer la stabilité et la fiabilité de l'algorithme. Ces erreurs ont permis de mieux comprendre les contraintes et les limitations de l'algorithme, ainsi que l'importance de mettre en place des mécanismes appropriés pour les détecter, les prévenir ou les gérer de manière efficace. En documentant ces expériences et en tirant des leçons des erreurs rencontrées, il est possible de renforcer la robustesse de l'algorithme et d'optimiser ses performances dans des contextes similaires.

L'algorithme écrit en langage Java figure en annexe sur ce lien GitHub :
https://github.com/Kazzoul-Youness/StraightLine-ShiftMethod

15.5 Implémentation en language Java

1. Dans la première implémentation de la méthode shift dossier **Graphe** dans le github, nous calculons un sommet à la fois. Pour cela, nous devons définir w_p et w_q. Il n'est pas nécessaire d'avoir introduit tout le graphe et son ordre canonique. Les valeurs de w_p et w_q sont définis par l'utilisateur et au fur et à mesure le nombre de sommets augmentent.
   Lien : https://github.com/Kazzoul-Youness/StraightLine-ShiftMethod/tree/main/Graph

2. Dans la deuxième implémentation de la méthode shift dossier **ShiftMethodTest** dans le github, vise à calculer les temps d'exécution du programme, nous ne générons pas aléatoirement les valeurs de w_p et w_q, il s'avère qu'avec cette approche, il existe un risque d'obtenir des cordes 7.14 qui vont conduire notre graphe à un graphe non 2-connexe, ou avoir des points isolés. Pour remédier à cela, nous avons décidé de prendre : $w_p = n$ et $w_q = v_2$, ou $w_p = 1$ et $w_q = i$, avec n le nombre de sommets et i l'indice de l'étape de la boucle principal, $3 \leq i \leq n$
   Lien : https://github.com/Kazzoul-Youness/StraightLine-ShiftMethod/tree/main/ShiftMethodTest

Pour toutes les informations sont sur le dossier parent sur le GitHub : https://github.com/Kazzoul-Youness/StraightLine-ShiftMethod

16 Conclusion

En conclusion, notre étude approfondie de la méthode Shift method pour la représentation en ligne droite des graphes planaires a mis en évidence son importance dans de nombreux domaines d'application. L'algorithme de Shift method, développé par Fraysseix, Pach et Pollack, offre une approche efficace pour créer des représentations visuelles claires et ordonnées de ces graphes.

Dans ce travail, nous avons examiné en détail les étapes clés de l'algorithme, en mettant l'accent sur l'ordre canonique des sommets, qui joue un rôle crucial dans la préservation de la structure et de la connectivité du graphe d'origine. Nous avons également exploré les techniques utilisées pour organiser les sommets de manière linéaire et créer des tracés optimaux, ainsi que les heuristiques de placement des arêtes pour améliorer l'esthétique des dessins en lignes droites des graphes planaires.

Cette étude a contribué à une meilleure compréhension de la représentation en ligne droite des graphes planaires et a ouvert de nouvelles perspectives intéressantes pour la visualisation et l'analyse de ces graphes. L'adoption de l'algorithme de Shift method comme outil essentiel dans ce domaine est fortement recommandée, étant donné sa capacité à générer des dessins clairs et compréhensibles, tout en préservant la structure sous-jacente du graphe.

Nous avons également analysé la complexité de l'algorithme de Shift method et identifié des opportunités d'optimisation pour améliorer ses performances. Des méthodes alternatives pour le calcul des coordonnées des sommets ont été proposées, permettant de réduire le temps de calcul nécessaire.

En conclusion, la représentation en ligne droite des graphes planaires reste un domaine de recherche riche en applications pratiques et en défis théoriques. L'algorithme de Shift method se positionne comme une approche efficace pour créer des dessins en lignes droites de ces graphes, tout en prenant en compte des considérations esthétiques. Cette étude contribue à une meilleure compréhension de la méthode et encourage l'exploration de nouvelles perspectives dans le domaine de la visualisation et de l'analyse des graphes planaires.
Nous espérons que ce travail incitera à des recherches supplémentaires dans ce domaine et favorisera l'adoption de l'algorithme de Shift method comme outil essentiel pour la représentation en ligne droite des graphes planaires, permettant ainsi d'améliorer la compréhension et l'analyse de ces structures.

Références

B Portie, J-M Menyr. Qu'est-ce qu'un graphe ? en ligne. Consulté février 2023 - https://math.univ-lyon1.fr/irem/Formation_ISN/formation_parcours_graphes/graphe/1_notion_graphe.html.

Balakrishnan, A., Ranganathan, K. Graph theory and its applications. 2018. Consulté decembre 2022.

Belbèze, Christian. Agrégats de mots sémantiquement cohérents issus d'un grand graphe de terrain. PhD thesis, Université Toulouse 1 Capitole (UT1 Capitole), 4 2012. Thèse de doctorat en Informatique.

Bennett, Ryall, Spaltzerholz et Gooch. The Aesthetics of Graph Visualization. 2007. Consulté fevrier 2023.

Bhasker, Jayaram and Sahni, Sartaj. A linear algorithm to find a rectangular dual of a planar triangulated graph. Algorithmica, 3(1-4) :247–278, 1988. doi : 10.1007/BF01762117.

De Fraysseix, Pach et Pollack. How to draw a planar graph on a grid. Combinatorica, 10(1), 41-51. doi : 10.1007/BF02125347., 1990. Consulté Janvier 2023.

Di Battista, G., Tamassia, R. Algorithms for plane representations of acyclic digraphs. 1989. Consulté decembre 2022.

Eppstein, D. and Gansner, Emden R. Leftist canonical ordering. In David Eppstein and Emden R. Gansner, editors, Graph Drawing, volume 5849 of Lecture Notes in Computer Science, pages 159–170. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-11804-3. doi : 10.1007/978-3-642-11805-0_16. URL https://dx.doi.org/10.1007/978-3-642-11805-0_16.

I. Fary. On straight line representations of planar graphs. 1948. Consulté decembre 2023.

Freeman, R. L. Electronic system design : Interference and noise control techniques. 1991. Consulté decembre 2022.

J.A. Bondy and U.S.R. Murty. Théorie des Graphes. Traduit de l'anglais par F. Havet, 2008. Consulté Octobre 2022 - url : https://www-sop.inria.fr/members/Frederic.Havet/Traduction-Bondy-Murty.pdf.

D. Jungnickel. Graphs, Networks and Algorithms. Springer, 2003. Consulté Decembre 2022.

P. Kindermann. Planar Straight-Line Drawings I : Canonical Order & Shift Method. 2010. Consulté Mars 2022 - url : https://seafile.rlp.net/f/57a4d71cdf114a3dbbf0/.

C. Laforest. À la découverte des graphes et des algorithmes de graphes. Consulté Janvier 2023 EDP-sciences, 2017.

M. Chrobak and S. Nakano. Minimum-width grid drawings of plane graphs. 1998. Consulté novembre 2023.

M. Chrobak and T. H. Payne. A linear-time algorithm for drawing a planar graph on a grid. 1995. Consulté novembre 2023.

M. Sablik. GRAPHE ET LANGAGE. Consulté Novembre 2022 - url : https://www.math.univ-toulouse.fr/~msablik/CoursIUT/Graphe/GrapheNotes.pdf.

S. Mehl. Les graphes. en ligne. Consulté le 12 octobre 2022 - http://serge.mehl.free.fr/anx/Th_graphes.html#adj.

N. Takao, R. Saidur. Planar graph drawing. World Scientific Publishing Co. Pte. Ltd., 2017. Consulté Novembre 2022.

R. Diestel. Graph Theory. Springer, 2005. Consulté Novembre 2022.

Schnyder, Walter. Planar graphs and poset dimension. Order, 5(4) :323–343, 1989. doi : 10.1007/BF00353652.

K. S. Stein. Convex maps. 1951. Consulté novembre 2023.

Todeschini, R., Consonni, V. Molecular descriptors for chemoinformatics. 2009. Consulté decembre 2022.

W. T. Tutte. How to draw a graph. Proceedings of the London Mathematical Society, 1963. Consulté fevrier 2023.

L. Vismara. Planar straight-line drawing algorithms. In R. Tamassia, editor, Handbook on Graph Drawing and Visualization, pages 193–222. Chapman and Hall/CRC, 2013. Wed, 12 Jul 2017 09 :11 :51 +0200, https ://dblp.org/rec/reference/crc/Vismara13.bib, dblp computer science bibliography, https ://dblp.org.

W. Schnyder. Embedding planar graphs on the grid. 1990. Consulté Janvier 2023.

K. Wagner. Bemerkungen zum vierfarbenproblem. 1936. Consulté decembre 2023.

H. Wang. Graph theory and interconnection networks. 2008. Consulté decembre 2022.

West, Douglas B. Introduction to Graph Theory. Prentice Hall, Urbana, 2 edition, 2001. 2001, isbn=81-7808-830-4, University of Illinois, Consulté Mars 2023.

H. Whitney. Congruent Graphs and the Connectivity of Graphs. 1933. Consulté Mars 2023.