Modèles et algorithmes pour les graphes dynamiques

Mathilde Vernet

Normandie Université

THESE

Pour obtenir le diplôme de doctorat

Spécialité Informatique

Préparée au sein de l'Université Le Havre Normandie

Modèles et algorithmes pour les graphes dynamiques

Présentée et soutenue par
Mathilde VERNET

Thèse soutenue publiquement le 19 octobre 2020 devant le jury composé de		
Mme Nadia BRAUNER	Professeure des universités, Université Grenoble Alpes	Examinatrice
M. Arnaud CASTEIGTS	Maitre de conférences HDR, Université de Bordeaux	Rapporteur
M. Paul DORBEC	Professeur des universités, Université Caen Normandie	Examinateur
M. Christophe PICOULEAU	Professeur des universités, CNAM Paris	Rapporteur
M. Yoann PIGNÉ	Maitre de conférences, Université Le Havre Normandie	Co-encadrant de thèse
M. Éric SANLAVILLE	Professeur des universités, Université Le Havre Normandie	Directeur de thèse

Thèse dirigée par Éric SANLAVILLE, laboratoire LITIS

UNIVERSITÉ LE HAVRE NORMANDIE

MISIΣ

litis

Le mot de la fin

Quelqu'un m'a dit un jour « la thèse est ce que le doctorant en fait ». Si j'ai pu en faire ce qu'elle est aujourd'hui, c'est grâce à une série d'évènements et de personnes qui ont marqué mon chemin. C'est grâce à vous tous et vous toutes que j'en suis là aujourd'hui.

Il y a 8 ans, je n'aurais jamais cru vouloir faire de l'informatique, il y a 5 ans, je n'aurais jamais cru vouloir faire de la recherche, il y a 3 ans je n'aurais jamais cru vouloir déménager en Normandie. Et pourtant...

Pourtant quelques cours d'algo, de graphes, d'optim et de RO plus tard, quelques mois de stages de recherche plus tard, un coup de cœur pour un sujet de thèse plus tard, me voilà débarquant au Havre un jour de juillet avec ma vie dans une camionnette. Puis me revoilà, 3 ans plus tard, écrivant ces quelques lignes comme on écrirait un générique de fin.

Tout d'abord à ceux qui ont été les plus importants et sans qui cette thèse n'existerait pas : Éric et Yoann. Merci de m'avoir guidée, de m'avoir fait confiance, de m'avoir apporté un peu de la guimauve qui me manque, d'avoir su trouver les mots pour me motiver et me pousser quand j'en avais besoin, pour me rassurer quand je doutais. Merci de m'avoir laissé prendre mes décisions et faire mes choix. Grâce à vous, j'ai pu faire ce qui m'intéressait sans jamais m'ennuyer et jamais sans votre soutien. Votre immense complémentarité m'a apporté, au quotidien, un équilibre inébranlable. Tout simplement merci de ne jamais m'avoir fait regretter ma décision de traverser le pays, pas même une seule seconde.

Pendant ces trois années, j'ai eu la chance de pouvoir bénéficier de fructueux échanges scientifiques. Merci à Maciej Drozdowski pour ces nombreuses séances de travail à Poznan qui m'ont permis de publier un premier article. Merci à Stefan Balev et à Frédéric Guinand pour les échanges et les bonnes idées qui m'ont permis d'avancer et que l'on retrouve dans ce manuscrit.

Plus globalement, merci à toutes celles et tous ceux qui ont suivi mon travail pendant ces trois années, au LITIS, mais aussi au LMAH, en m'apportant un peu de leur expérience et de leurs conseils. Je ne peux pas citer tout le monde mais je porte une attention particulière à Nathalie Verdière qui a participé à mon comité de suivi.

Avant d'en arriver là, il y a eu des gens qui m'ont donné envie de me lancer dans cette aventure. Tout d'abord à ceux qui m'ont suivie lors des deux stages de recherche en master, dans lesquels j'ai pu découvrir ce que c'était et à quel point ça me plaisait. Merci de m'avoir mis le pied à l'étrier de la recherche. À ces enseignant·e·s qui m'ont initiée à certains pans de l'informatique. Merci d'avoir participé à la construction de mon chemin. En particulier merci à Nadia Brauner qui a beaucoup contribué à mon parcours et mon évolution depuis le M1 pour enfin participer au jury de soutenance.

Le jury était aussi composé d'Arnaud Casteigts et Christophe Picouleau qui ont accepté de rapporter ce manuscrit, et Paul Dorbec qui a accepté de présider le jury. Merci pour tous les commentaires pertinents et les questions intéressantes lors de la soutenance.

À la famille, qui m'a toujours encouragée dans chacun de mes choix, sans exceptions, malgré le stress que cela a généré. Merci pour ce soutien indéfectible et cette confiance à toute épreuve, dans une constante bonne humeur.

À toutes celles et tous ceux qui sont à mes côtés, même à distance, depuis de nombreuses années. Merci de continuer de vous réjouir pour moi alors même que ce que je fais reste parfois un peu obscur.

À tous les gens que j'ai rencontré ici, avec qui j'ai tissé des liens, parfois très forts, et qui, depuis 3 ans, sont présents pour discuter de tout et de rien, rigoler mais aussi m'écouter me plaindre autour d'un café, un repas, une bière, ou simplement au détour d'un couloir. Vous avez été des vrais piliers, merci à vous.

On n'oubliera pas la Normandie pour ce beau décor, et les havrais qui m'ont accueillie et m'ont fait me sentir ici chez moi. Une partie de moi restera toujours un peu havraise.

Enfin, pensée au SARS-CoV-2, sans qui tout aurait été beaucoup trop simple. De la perturbation de la phase de rédaction à la perturbation de la soutenance, je me serais bien passé du challenge.

Table des matières

Table des figures

Liste des tableaux

Liste des définitions

Liste des théorèmes

Liste des algorithmes

Chapitre 1

Introduction

Les graphes et les problèmes qu'ils permettent de résoudre au travers d'algorithmes sont largement étudiés, mais ces sujets sont surtout connus lorsque l'on considère le graphe comme un objet fixe. On peut alors se demander ce qu'il se passe si l'on introduit une dimension temporelle. Une communauté grandissante s'intéresse à ce cas.

Le premier objectif de ce travail est de s'interroger sur les graphes et les problèmes de graphes dans un contexte dynamique. C'est à dire qu'au lieu de voir le graphe comme un ensemble figé, nous souhaitons l'observer au cours du temps en l'autorisant à évoluer. C'est ainsi que l'on obtient ce qu'on appelle un graphe dynamique.

Puisque l'on s'intéresse à des graphes dynamiques plutôt qu'à des graphes statiques, nous devons aussi redéfinir les problèmes de graphes classiques. C'est sur ce point que repose notre travail. Nous nous interrogeons, dans un premier temps, sur cette redéfinition avant de nous consacrer à leur résolution.

Les graphes statiques sont utilisés dans de nombreux domaines d'application, et il en va de même pour les graphes dynamiques. Il existe en effet une grande variété de champs applicatifs dans lesquels l'introduction d'une dimension dynamique a du sens. C'est par exemple le cas des réseaux sociaux, où la nature des relations peut évoluer, ou bien de nouvelles relations peuvent se créer et d'anciennes peuvent se rompre, des réseaux de transport où par exemple de la marchandise circule sur des « voies » dont la disponibilité peut varier. On pourra également citer les réseaux télécoms, comme les réseaux adhoc où un grand dynamisme des liens peut être observé, ou encore les réseaux biologiques où les interactions protéine-protéine sont dynamiques.

Contrairement à de nombreuses études de la littérature, notre démarche ne consiste pas à partir d'un problème réel, mais se concentre sur les graphes dynamiques comme des objets génériques en s'abstrayant des applications.

Comme nous le montrerons, la dimension temporelle ouvre de nouvelles possibilités de modélisation. Cela apporte donc de nouvelles questions sur la résolution des problèmes.

Dans l'absolu, tous les problèmes de graphes, selon la manière dont ils sont étendus aux graphes dynamiques, peuvent être pertinents à étudier. Cependant, lorsque l'on aborde ces problèmes d'un point de vue algorithmique, tous ne sont pas toujours intéressants.

Lorsque le temps n'entre pas en compte dans la résolution, ou bien lorsqu'il est possible de se ramener simplement à un problème dans un graphe statique, alors, bien que le problème en lui-même dans un graphe dynamique ait un intérêt, la dimension temporelle du graphe dynamique n'apporte aucune difficulté supplémentaire pour résoudre le problème. Dans ce cas, une méthode de résolution spécifique aux graphes dynamiques n'est pas nécessaire et chercher à concevoir un nouvel algorithme pour les graphes dynamique apparait [1] donc superflu.

Lorsqu'un problème sur un graphe dynamique ne peut pas être redéfini comme un problème sur un graphe statique, il faut alors proposer de nouvelles méthodes adaptées aux graphes dynamiques afin de les résoudre. C'est ici que se place notre travail : ce manuscrit est consacré à l'algorithmique dans les graphes dynamiques.

Afin de diriger le travail, il devient alors pertinent de s'interroger sur les problèmes eux-mêmes afin de savoir lesquels nécessitent une méthode spécifique et lesquels peuvent être résolus par des méthodes sur des graphes statiques. Parfois, la pertinence peut aussi dépendre de la manière dont le problème est étendu aux graphes dynamiques. En effet, pour un même problème sur un graphe statique, on peut trouver plusieurs manières de le définir sur un graphe dynamique et certaines seront algorithmiquement intéressantes et d'autres pas.

Illustrons ce point par des exemples. Un graphe dynamique est vu comme une succession de graphes statiques (ce qui, comme nous le verrons par la suite, est courant). Considérons comme premier exemple la recherche d'un stable maximum dans un graphe dynamique. Cela signifie intuitivement que l'on cherche un ensemble maximum de sommets tels qu'ils ne sont jamais connectés entre eux, à aucun pas de temps. On peut donc se ramener à la recherche d'un stable maximum dans l'union des graphes statiques successifs. Une méthode spécifique aux graphes dynamiques n'est, dans ce cas, pas nécessaire.

Considérons maintenant le problème de l'arbre couvrant de poids minimum. Si l'on considère que ce problème dans un graphe dynamique consiste à trouver un arbre couvrant de poids minimum à chaque pas de temps, alors la résolution est simple. En effet, il suffit de trouver un arbre couvrant par pas de temps. Il n'est alors pas nécessaire de développer une méthode spécifique aux graphes dynamiques. Si maintenant on étend la notion de chemin aux graphes dynamiques, alors on obtient un nouveau problème d'arbre couvrant dynamique pour lequel une méthode spécifique est nécessaire (voir chapitre 4, page 89).

Enfin, les problèmes de flots dans les graphes dynamiques peuvent être résolus grâce à des graphes statiques (comme nous l'expliquerons dans le chapitre 3), mais c'est inefficace à cause de la taille de ces graphes. C'est pourquoi des méthodes spécifiques aux graphes dynamiques sont utiles.

Nous travaillons sur l'algorithmique dans les graphes dynamiques, nous choisissons donc de nous intéresser à des problèmes qui ne peuvent pas être résolus, ou alors de manière inefficace, en se ramenant à un problème sur un graphe statique. Nous nous concentrons donc sur des problèmes pour lesquels nous souhaitons concevoir une nouvelle méthode de résolution.

Nous adoptons une démarche similaire pour chaque problème abordé, illustrée par le schéma de la figure 1.1. Pour un problème donné, nous avons pour objectif de répondre aux trois questions suivantes :

1. Comment ce problème s'étend-il à un graphe dynamique ?
2. Cette extension a-t-elle un sens ?
3. Comment peut-on le résoudre ?

Afin de répondre à ces questions, nous commençons par définir clairement, d'une part, le modèle de graphe le plus approprié pour le problème étudié, et d'autre part, ce que signifie exactement ce problème sur le graphe dynamique étudié. Nous pouvons ensuite travailler à la conception d'un algorithme pour résoudre le problème. Nous avons pour objectif de proposer des algorithmes exacts, dont la complexité

1. On notera que ce manuscrit est rédigé en utilisant l'orthographe réformée de 1990.

théorique reste raisonnable au pire cas. Nous nous attachons aussi à proposer des algorithmes qui soient efficaces en pratique, c'est la raison pour laquelle nous proposons une étude expérimentale afin de valider pratiquement nos algorithmes. En suivant cette démarche, nous avons traité différents problèmes.

Ce manuscrit se divise en trois grande parties. Nous présentons d'abord les généralités sur les graphes dynamiques. Puis nous nous focalisons sur les travaux que nous avons réalisés sur les problèmes de flot dans les graphes dynamiques. Nous présentons ensuite nos travaux sur la question de la connexité dans les graphes dynamiques.

Nous commençons, dans le chapitre 2, par définir les bases nécessaires à la compréhension du travail réalisé. C'est-à-dire que nous y décrivons nos motivations un peu plus en détails. Puis nous discutons de graphes dynamiques de façon générale, en présentant les modèles existants et ceux que nous considérons. Nous définissons ensuite toutes les notions utilisées dans ce travail, discutons de la littérature traitant des problèmes dans les graphes dynamiques et considérons leur résolution.

Le chapitre 3 est consacré aux flots dans les graphes dynamiques. Il existe une littérature très fournie étudiant ces problèmes. Nous en faisons une revue. Nous présentons le travail, non abouti, que nous avons réalisé sur le problème de flot maximum dans un graphe dynamique. Nous présentons ensuite notre travail complet sur une variante du problème de flot de cout minimum : un nouvel algorithme, son analyse théorique et une étude expérimentale.

Le chapitre 4 se concentre sur la connexité dans les graphes dynamiques. La littérature sur le sujet est relativement maigre, nous en faisons une revue. Nous présentons notre vision des composantes connexes dans un graphe dynamique, un algorithme, son analyse théorique et une étude expérimentale. Nous présentons des extensions possibles de notre définition. Nous décrivons aussi dans ce chapitre le travail que nous avons réalisé sur les extensions possibles du problème de Steiner aux graphes dynamiques. Nous démontrons la NP-complétude d'une variante particulière du problème.

Enfin, le chapitre 5 revient sur tout le travail réalisé afin d'en faire le bilan en rapport avec notre objectif initial et présente quelques perspectives.

On notera que le chapitre 2 donne des définitions essentielles, il est donc important à la compréhension de la suite et doit être lu avant tout. En revanche, les chapitres 3 et 4 sont indépendants entre eux.

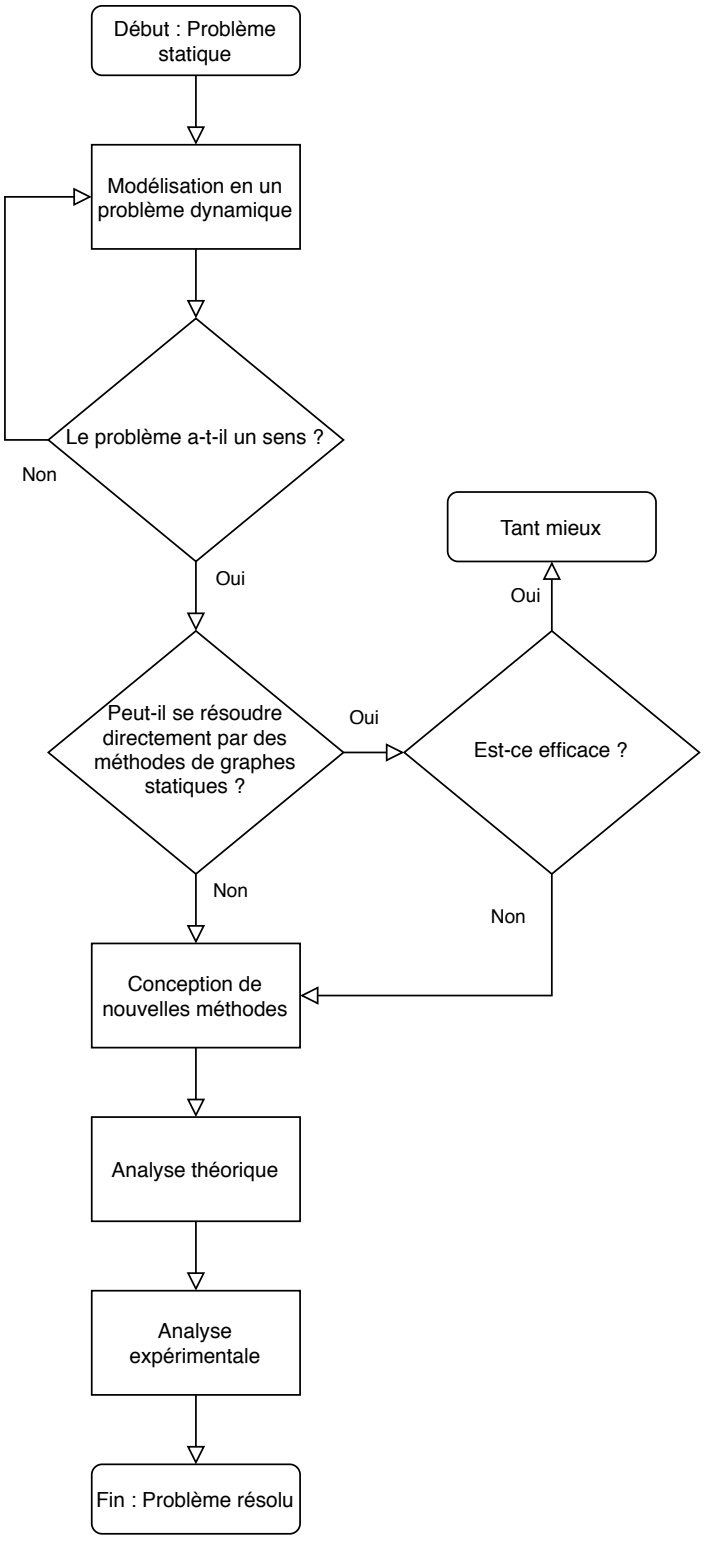


Figure 1.1 : Organigramme de notre démarche de résolution des problèmes.

Chapitre 2

Généralités sur les graphes dynamiques

L'objectif de ce chapitre est de poser les bases nécessaires pour comprendre l'étude des graphes dynamiques. Nous y présentons donc des informations générales concernant les graphes dynamiques. Tout d'abord, les motivations de ce travail ainsi que les applications possibles sont présentées en section 2.1. En section 2.2 est abordée la question des nombreux modèles de graphes dynamiques et ceux utilisés dans ce travail y sont introduits. Nous donnons les définitions importantes utilisées tout au long de ce manuscrit en section 2.3. Enfin, la section 2.4 présente différentes études de la littérature sur des problèmes dans les graphes dynamiques.

2.1 Motivations et applications

De façon générale, les graphes permettent la modélisation d'entités en interaction les unes avec les autres. C'est une vision très large, permettant d'imaginer divers types d'entités et divers types d'interactions. De fait, cela en fait un modèle très largement utilisé dans de nombreux domaines d'applications.

Les graphes peuvent ainsi modéliser des réseaux aussi variés que les réseaux sociaux ou les réseaux de transport en passant par les réseaux biologiques. En effet, Facebook où des individus ont des liens « d'amitié » avec d'autres individus, un réseau de transport en commun de ville où, par exemple, chaque arrêt est relié à un autre par une ligne de bus, ou encore des molécules dans lesquelles les atomes sont reliés les uns aux autres, peuvent tous être modélisés par un graphe.

Malheureusement, ce modèle est parfois trop restreint et n'est pas toujours suffisant pour représenter tous les aspects d'un système. En effet, beaucoup de systèmes ne sont pas « figés » dans le temps. Or, le temps peut jouer un rôle important. Les interactions peuvent avoir lieu au cours du temps ou bien être modifiées dans le temps.

Dans un tel cas, le graphe n'est pas assez complet pour modéliser des systèmes de cette complexité. Toute l'information ne peut pas être prise en compte dans le graphe. Il peut en résulter une modélisation inadéquate du système en question.

Pour pallier cette insuffisance, un graphe peut bénéficier de l'ajout d'une dimension temporelle. C'est ainsi que l'on obtient des graphes dynamiques. Grossièrement (ils seront formellement définis en section 2.3), il s'agit de graphes définis sur un ensemble de sommets, aussi appelés nœuds, et d'arêtes, comme les graphes statiques. La différence repose sur le fait que les graphes dynamiques sont observés au cours du temps et leurs différents éléments et informations peuvent varier au cours du temps. De nombreux modèles différents de graphes dynamiques peuvent ainsi être définis (ce point est présenté de façon approfondie en section 2.2).

Les graphes dynamiques offrent la possibilité d'une modélisation plus fine d'un système puisqu'ils permettent de modéliser des interactions évoluant dans le temps. De ce fait, ils seront plus adaptés que les graphes statiques dans de nombreux cas, et correspondront à une plus grande variété de systèmes.

Les graphes dynamiques présentent alors un grand intérêt. Il n'est donc pas surprenant de voir qu'ils ont été assez largement étudiés. Une grande variété de domaines d'applications peut bénéficier des graphes dynamiques. On trouve donc, dans la littérature, des travaux sur les graphes dynamiques venant de communautés très variées. L'une des principales conséquences est l'absence d'unicité du vocabulaire utilisé. En effet, quand certains parlent de *graphes*, d'autres parlent de *réseaux* et ils sont définis tantôt comme étant *dynamiques*, tantôt comme étant *temporels* ou *évolutifs*. Ce point est plus amplement développé en section 2.2.

Les travaux traitant les graphes dynamiques d'un point de vue applicatif sont nombreux. En effet, beaucoup de travaux sont issus d'un domaine d'application et cherchent à modéliser leur système grâce à un graphe dynamique. Holme (2015) a réalisé un survey complet des travaux traitant des graphes dynamiques en fonction du domaine d'application.

Les réseaux sociaux peuvent ainsi être modélisés de la sorte. On peut représenter par un graphe dynamique les interactions physiques entre des personnes. C'est par exemple l'objet du travail de Cattuto et al. (2013) où les interactions entre les sujets de l'expérience étaient captées à l'aide de puces RFID. Un graphe dynamique peut aussi représenter des communications entre des personnes, par exemple Li et al. (2014) représentent des contacts téléphoniques entre personnes. Nguyen et al. (2011), parmi d'autres, s'intéressent à la détection de communauté dans un contexte dynamique.

Les graphes dynamiques peuvent aussi permettre de modéliser des interactions biologiques. Bassett et al. (2013) représentent les évolutions des interactions entre des régions du cerveau lors de l'apprentissage d'une nouvelle tâche. Taylor et al. (2009) ou encore Luo et Kuang (2014) s'intéressent aux interactions entre des protéines.

Les graphes dynamiques sont aussi adaptés pour modéliser des réseaux de transports. Démare et al. (2017) étudient des réseaux logistiques dynamiques dans la vallée de la Seine. Rosvall et al. (2014) s'intéressent à un réseau d'aéroports connectés par des liaisons aériennes. De manière similaire, Guinand et Pigné (2015) s'intéressent à l'évolution du réseau des ports maritimes. Scholtes et al. (2014) étudient, entre autres, le réseau du métro londonien construit à partir des trajets parcourus par les passagers. Les réseaux viaires sont aussi une application possible, on citera les travaux de Tirico et al. (2018) qui s'intéressent en particulier à leur morphogenèse.

Les transactions financières sont un autre exemple de domaine pouvant bénéficier des graphes dynamiques. Zhao et al. (2015) s'intéressent à des transactions de cartes bancaires. Kondor et al. (2014) s'intéressent au Bitcoin, et à ses transactions qui sont inscrites dans le temps.

On peut donc conclure de cette rapide revue de la littérature que les domaines d'applications des graphes dynamiques sont variés, et les travaux nombreux dans chaque domaine. Les graphes dynamiques présentent donc, au delà de l'intérêt théorique pour ce type de graphes, un réel intérêt applicatif.

2.2 Modèles de graphes dynamiques

Un graphe dynamique est défini comme un graphe auquel est ajouté une dimension temporelle. Cela peut vouloir dire deux choses. D'une part le graphe peut simplement être étudié au cours du temps à travers un phénomène qui évolue dans le temps sur un graphe qui lui, reste statique, ou d'autre part le graphe lui-même peut évoluer au cours du temps. Et bien sûr, lorsque le graphe évolue au cours du temps, il est aussi étudié dans le temps. Sauf mention contraire, c'est cette seconde interprétation que nous considérons.

Le graphe dynamique est classiquement composé de nœuds et d'arêtes ou d'arcs, comme les graphes statiques dont ils sont une extension. Un intervalle de temps sur lequel le graphe est étudié est aussi défini.

Nous discutons ici les modèles de graphes dynamiques envisageables. Nous présentons ceux qui existent dans la littérature puis ceux sur lesquels nous travaillons.

2.2.1 Modèles existants

Historiquement, on trouve des travaux étudiant des graphes statiques au cours du temps dès les années 1950. Ces travaux, menés par Ford et Fulkerson (1958, 1962), étudient un phénomène au cours du temps sur des graphes statiques. Les auteurs traitent des problèmes de flot qu'ils nomment « flots dynamiques » (nous reviendrons plus particulièrement sur ces travaux dans le chapitre 3).

Dans les graphes étudiés, la dimension temporelle est ajoutée grâce à des temps de traversée sur les arcs du graphe (voir définition 6 page 13). Ainsi le flot traverse le graphe au cours du temps et est donc étudié dans le temps, quand le graphe lui, reste statique.

Des modèles similaires, toujours dans le cadre de problèmes de flots, ont été utilisés plus tard par Wilkinson (1971) puis par Hoppe et Tardos (1994, 2000).

Dans tous ces travaux, le flot est étudié dans le temps mais le graphe n'évolue pas au cours du temps. On considèrera donc qu'il s'agit d'un graphe statique. Cela n'empêche cependant pas l'étude au cours du temps d'un phénomène dynamique sur ces graphes.

Il existe de nombreux travaux dans lesquels des graphes sont étudiés au cours du temps. Il ne s'agit pas nécessairement de travaux portant spécifiquement sur les graphes, puisque leurs auteurs ne sont en général pas des spécialistes de graphes. En effet, de nombreux domaines de recherche peuvent bénéficier de modèles de graphes dynamiques. De ce fait, les travaux sont souvent issus de communautés très différentes.

L'une des conséquences majeures de cette diversité dans les travaux traitant de graphes dynamiques est l'absence de consensus sur certains points essentiels, notamment concernant le vocabulaire utilisé. En particulier, dans la littérature, on trouve des noms différents pour désigner les graphes dynamiques. Certains travaux parlent de *graphes*, d'autres parlent de *réseaux*. Et ces graphes ou réseaux sont définis comme étant *dynamiques*, *temporels*, *variables dans le temps* ou encore *évolutifs*.

On trouve quasiment toutes les combinaisons possibles pour désigner ce que nous appelons les graphes dynamiques. Par exemple, Neggaz (2016) parle de *dynamic graphs*. Michail (2016) parle de *temporal graphs*. Casteigts et al. (2012) parlent de *time-varying graphs*. Bhadra et Ferreira (2003)

parlent d'*evolving graphs*. Bui-Xuan et al. (2003) parlent de *dynamic networks*. Kempe et al. (2002) parlent de *temporal networks*. Khodayifar et al. (2017) parlent de *time-varying networks*. Le terme de *flot de liens* (*link stream*) est aussi utilisé par Latapy et al. (2017).

Malgré la multiplicité de termes, ils définissent tous une notion similaire, à savoir un ensemble de nœuds et d'arêtes ou d'arcs observés au cours du temps et dont les caractéristiques (présence ou poids) peuvent varier dans le temps. Les plus importants travaux sur les graphes dynamiques ont été réalisés principalement au cours de ces 20 dernières années.

Il existe de très nombreux modèles de graphes dynamiques. Mais avant de s'intéresser à ce qui évolue ou non dans le graphe et aux différents paramètres du graphe, on peut déjà envisager de plusieurs modèles selon la connaissance que l'on a du graphe. C'est ce que montre la figure 2.1 où apparait toute la combinatoire des modèles possibles de connaissance du graphe dynamique.

D'abord, on peut connaitre le graphe globalement ou localement. C'est à dire que l'on peut soit connaitre le graphe entièrement, soit chaque nœud connait son voisinage seulement. Ensuite, on peut connaitre l'évolution du graphe de différentes manières. On peut découvrir les différentes modifications quand elles arrivent, auquel cas on parle de modèle *en ligne* (ou *online* en anglais). Les modifications peuvent être stochastiques, auquel cas on ne connait pas précisément les modifications à l'avance mais on connait les lois de probabilités qui les régissent. Les modifications peuvent aussi être totalement déterministes, auquel cas on les connait toutes à l'avance.

Le cas où l'évolution du graphe se fait selon des lois de probabilités est étudié dans la littérature, on pourra citer, par exemple, les travaux de Lamprou et al. (2018). Nous ne considèrerons jamais ce cas dans ce manuscrit.

Toutes les combinaisons données dans la figure 2.1 sont possibles. En effet, une connaissance locale en ligne signifie que chaque nœud découvre comment évoluent ses voisins au fur et à mesure de l'évolution. Une connaissance locale stochastique signifie que chaque nœud connait la loi ou les lois de probabilité selon lesquelles ses voisins évoluent. Une connaissance locale déterministe signifie que chaque nœud sait comment ses voisins vont évoluer. Une connaissance globale en ligne signifie que l'on découvre l'évolution de tout le graphe au fur et à mesure que le temps passe. Et une connaissance globale stochastique signifie que l'on connait la loi ou les lois de probabilités selon lesquelles tout le graphe évolue. Une connaissance globale déterministe signifie que l'on sait exactement comment tout le graphe va évoluer.

On notera que les modèles où le graphe n'est connu qu'à l'échelle locale nécessitent un traitement par des algorithmes distribués.

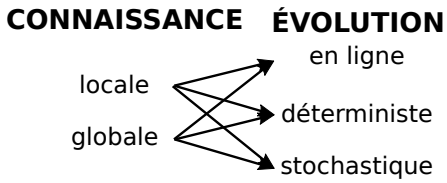


Figure 2.1 : Illustration des différents modèles possibles concernant la connaissance d'un graphe dynamique.

Outre ce que l'on connait de l'évolution du graphe, il y a un certain nombre de paramètres qui peuvent évoluer dans le graphe au cours du temps. Et cela peut donner beaucoup de modèles de graphes dynamiques différents. C'est ce que montre la figure 2.2 où apparait toute la combinatoire des différents modèles possibles selon ce qui est variable dans le graphe.

L'intervalle de temps sur lequel le graphe est étudié peut être fini ou infini. Le temps peut être décrit comme une variable discrète ou continue. Les nœuds peuvent être présents tout le temps ou bien

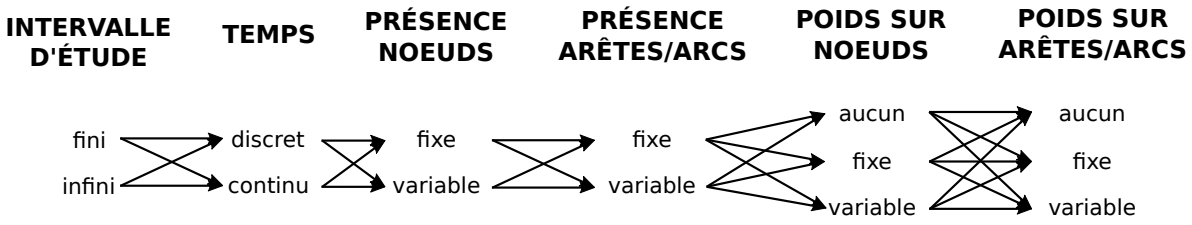


Figure 2.2 : Illustration des différents modèles possibles de graphes dynamiques en fonction des différents paramètres. Il y a d'autant plus de possibilités qu'il y a de types de poids différents.

apparaitre et disparaitre. Les arêtes (ou arcs dans le cas d'un graphe orienté) peuvent, comme les nœuds, être présentes tout le temps ou bien apparaitre et disparaitre. Les nœuds comme les arêtes (ou arcs) peuvent porter un certain nombre d'informations. En restant général, on les appelle des poids. Il est possible qu'il n'y ait aucun poids. Ces poids peuvent restent constants dans le temps ou bien peuvent varier au cours du temps.

On note qu'à chaque type de poids différent sur un nœud ou une arête correspond une colonne de plus dans le schéma de la figure 2.2. Dans le cas des problèmes de flot par exemple, un arc a deux types de poids différents : une capacité et un cout. Cela ajoute donc un certain nombre de modèles supplémentaires. On constate bien, grâce au schéma de la figure 2.2 qu'il existe de nombreux modèles possibles de graphes dynamiques.

Si les nœuds restent toujours présents, que les arêtes restent toujours présentes, et que tous les poids sont constants, en d'autres termes si tous les paramètres du graphe sont fixes au cours du temps, alors on se trouve dans le cas d'un graphe statique. On se trouve alors dans le cas d'un modèle de graphe avec connaissance déterministe selon le schéma de la figure 2.1 car on sait que le graphe n'évoluera jamais. En revanche, le graphe peut être connu soit globalement soit localement. Comme nous l'avons dit précédemment, cela reste possible d'étudier ces graphes dans le temps au travers d'un phénomène dynamique ayant lieu sur ces graphes.

En combinant les différentes possibilités concernant la connaissance que l'on a du graphe dynamique et tous les paramètres de celui-ci, on obtient un nombre important de modèles de graphes dynamiques possibles. À cela s'ajoute bien sûr, comme pour les graphes statiques, toutes les possibilités concernant la structure même du graphe, comme par exemple le fait que le graphe soit orienté ou non ou encore qu'il possède des cycles ou non.

Il est possible de considérer des modèles très généralistes dans lesquels tout peut varier au cours du temps ou au contraire on peut considérer des modèles beaucoup plus spécifiques où par exemple un seul paramètre est variable. Le choix du modèle est souvent déterminé par l'utilisation que l'on souhaite faire du graphe dynamique. Par exemple si l'on considère une application en particulier, il n'est pas nécessaire de choisir un modèle trop général, on va plutôt prendre le modèle le plus proche de l'application donnée.

Les graphes dynamiques peuvent aussi être considérés sous différents points de vue. Casteigts et al. (2012) présentent les trois points de vue possibles. Ces points de vue concernent l'évolution du graphe. Le premier de ces points de vue est l'évolution arêtes-centrée. Dans ce cas, la présence de chaque arête du graphe est définie comme une union d'intervalles de temps. Le deuxième point de vue est l'évolution nœuds-centrée. Dans ce cas, on connait pour chaque nœud l'ensemble de ses voisins à un pas de temps donné. Enfin le dernier point de vue est l'évolution graphe-centrée. Dans ce cas, le graphe dynamique est vu comme une succession de graphes statiques. C'est ce point de vue que nous privilégions.

Dans ce même travail, Casteigts et al. proposent une large classification des graphes dynamiques. Les différentes classes décrites correspondent à différentes propriétés du graphe dynamique et sa structure.

Par exemple une classe simple est la classe des graphes dans lesquels il existe un nœud depuis lequel on peut atteindre tous les autres au cours du temps par un trajet (voir définition 7 page 14), ou encore la classe des graphes tels qu'il existe un trajet depuis n'importe quel nœud vers n'importe quel autre nœud. L'appartenance à ces classes peut être utilisée comme condition nécessaire pour certains problèmes et peut permettre de prouver certains résultats (comme l'existence d'un arbre couvrant dynamique).

Casteigts et al. proposent également des méthodes de construction de graphes ayant certaines propriétés données. Les classes de graphes définies sont reprises par Neggaz (2016).

2.2.2 Nos propositions

Nous voyons les graphes dynamiques comme une extension des graphes statiques. Le terme de *réseau* est plus souvent utilisé lorsqu'il s'agit de cas d'application (réseau social, réseau de transport,…). Puisque l'on s'abstrait des applications pour se concentrer sur l'objet lui-même, nous lui préférons le terme *graphe*. Quant au terme *dynamique*, en opposition au terme *statique*, il désigne une évolution dans le temps. Nous utiliserons donc le terme *graphe dynamique* dans ce travail.

Un graphe est composé de sommets que l'on peut aussi appeler des nœuds. Dans beaucoup de contextes sur lesquels on travaille, notamment les problèmes de flot, on parle en général de nœuds. Afin de conserver une certaine cohérence sur l'ensemble du manuscrit, on privilégiera donc le terme *nœud*.

Nous avons choisi de nous concentrer sur des modèles sans aucune incertitude. Nous considérons donc que les graphes étudiés sont connus dans leur ensemble et que leur évolution est exactement connue à l'avance.

On peut cependant noter que dans le chapitre 4, nous présentons un algorithme qui ne nécessite pas de connaitre l'évolution du graphe à l'avance. Dans ce cas, il est donc envisageable d'avoir un modèle de graphe en ligne.

En reprenant le schéma de la figure 2.1, on montre sur la figure 2.3 la connaissance que l'on a dans les modèles de graphes étudiés.

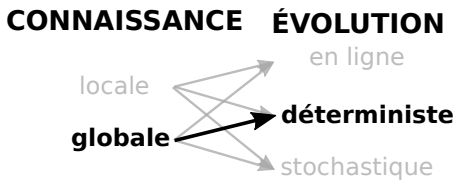


Figure 2.3 : Illustration du modèle choisi concernant la connaissance des graphes dynamiques.

Les figures 2.4 et 2.5, en reprenant le schéma de la figure 2.2, montrent les différents modèles de graphes dynamiques que l'on étudie selon les paramètres des graphes.

Nous considérons des graphes qui évoluent sur un intervalle de temps fini et discret. En effet, pour résoudre les problèmes par des algorithmes sur des graphes dynamiques en temps continu, il faut, en général, d'abord discrétiser le temps (Anderson et al., 1982 ; Hashemi et Nasrabadi, 2012).

Dans les graphes dynamiques que nous étudions, nous considérons que les nœuds sont fixes, autrement dit qu'ils ne peuvent pas disparaitre ou apparaitre. Ce n'est en fait pas une perte de généralité du modèle car, en effet, l'accessibilité d'un nœud peut être déterminée autrement que simplement par sa présence ou son absence. Si à une date θ donnée, aucune arête ne permet d'accéder à un nœud i, alors c'est équivalent au fait que ce nœud i n'est pas présent à la date θ.

En raisonnant de la même manière sur les arêtes (ou les arcs), dans le cas où les arêtes possèdent des poids, il est possible de modéliser l'absence d'une arête en jouant sur les poids. Par exemple, lorsque l'on étudie des problèmes de flot, les arêtes ont des capacités. Si la capacité d'une arête est nulle à une date θ

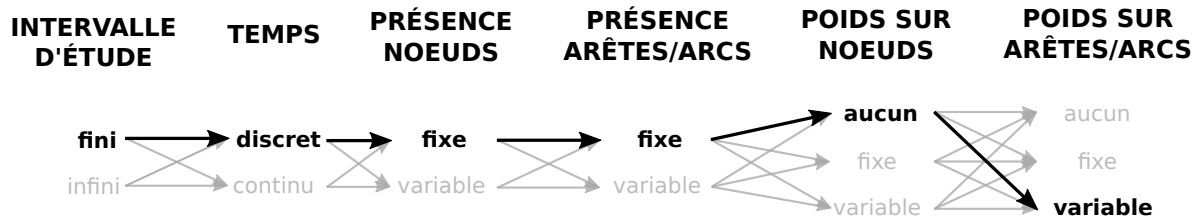


Figure 2.4 : Illustration du modèle de graphe dynamique choisi lorsque nous considérons des poids sur les arêtes.

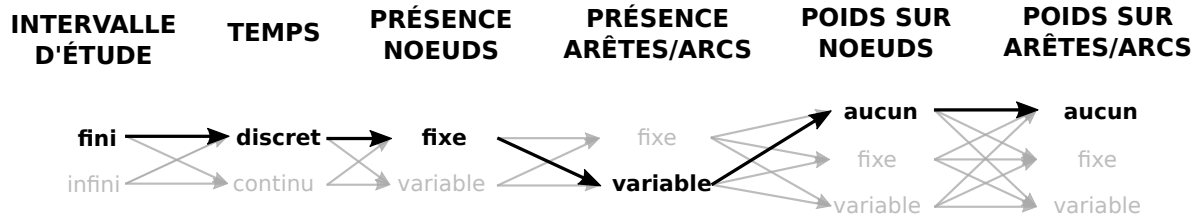


Figure 2.5 : Illustration du modèle de graphe dynamique choisi lorsque nous ne considérons pas de poids sur les arêtes.

donnée alors cette arête n'est pas utilisable à cette date, c'est donc comme si l'arête n'était pas présente à la date θ. On peut donc modéliser le fait qu'une arête peut apparaitre et disparaitre sans spécifier les dates auxquelles elle est présente, et donc en considérant qu'elle est toujours présente, et ce sans perte de généralité du modèle. Par extension, rendre une arête inaccessible grâce à un poids permet aussi de rendre un nœud inaccessible, il suffit pour cela de rendre inaccessible toutes les arêtes incidentes à ce nœud. C'est la raison pour laquelle, lorsque l'on s'intéresse à des modèles avec des poids sur les arêtes, on considère que celles-ci sont fixes : c'est le cas du modèle étudié dans le chapitre 3, où nous étudions des problèmes de flots et considérons des graphes orientés. Lorsque l'on s'intéresse à des modèles sans poids sur les arêtes, on considère que les arêtes peuvent aller et venir : c'est le cas du modèle étudié dans le chapitre 4, qui traite de la connexité et où nous considérons des graphes non orientés.

Comme expliqué précédemment, nous considérons des modèles dans lesquels nous avons une connaissance globale du graphe. De plus, nous considérons aussi le temps comme étant une variable discrète. Nous voyons donc un graphe dynamique comme une séquence ordonnée de graphes statiques définis sur le même ensemble de nœuds. Cela correspond à la vision graphe-centrée décrite par Casteigts et al. (2012).

Comme nous l'avons dit, ces modèles nous permettent de modéliser simplement l'absence de nœuds et le modèle avec poids nous permet de modéliser simplement l'absence d'arêtes. Ces modèles restent donc très généraux.

Nous supposons que les poids (lorsqu'il y en a) dépendent du temps, nous traitons donc ces paramètres variables. Mais cela signifie qu'il est aussi possible dans l'absolu de traiter des graphes où ces paramètres seraient constants avec les mêmes méthodes, bien que ces méthodes puissent ne pas être les plus efficaces pour de tels graphes.

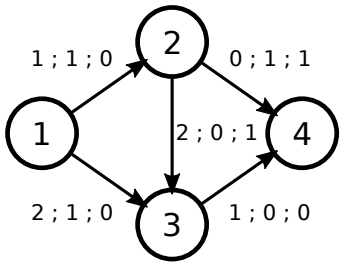
Nous considérons un intervalle de temps discret et voyons le graphe dynamique comme une succession de graphes statiques à chaque pas de temps de l'intervalle considéré. Mais il est tout à fait envisageable que chaque pas de temps de l'intervalle ne corresponde pas à des pas de temps réguliers mais plutôt à des moments de changements importants dans le graphe. Dans ce cas, cela signifie que le

temps écoulé entre une date θ et $\theta + 1$ n'est pas nécessairement le même que le temps écoulé entre une autre date θ' et $\theta' + 1$. Cela ne modifie rien au traitement que l'on fait des graphes dynamiques.

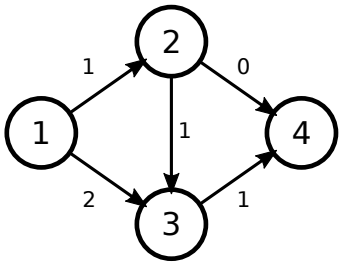
Un exemple de graphe dynamique (orienté) correspondant au modèle avec poids que nous considérons est présenté sur la figure 2.6. Et un exemple de graphe dynamique (orienté) correspondant au modèle sans poids que nous considérons est présenté sur la figure 2.7.

Sur la figure 2.6, on montre d'abord (figure 2.6a) le graphe dynamique en spécifiant, sur chaque arc les poids successifs de cet arc. Ce graphe dynamique à chaque pas de temps est représenté sur les figures 2.6b à 2.6d.

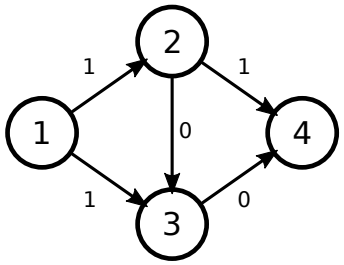
Sur la figure 2.7, on montre d'abord (figure 2.7a) le graphe dynamique en spécifiant, sur chaque arc, les pas de temps auxquels il est présent. Ce graphe dynamique à chaque pas de temps est représenté sur les figures 2.7b à 2.7d en faisant apparaitre à chaque date seulement les arcs présent à cette date.



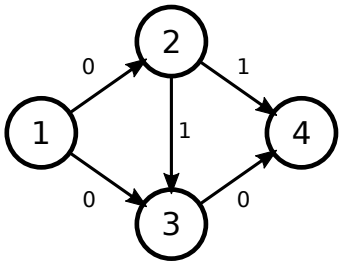
(a) Graphe dynamique avec les poids successifs indiqués sur les arcs.



(b) Premier pas de temps.



(c) Deuxième pas de temps.



(d) Troisième pas de temps.

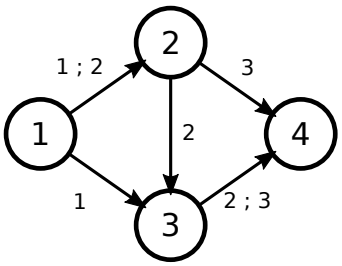
Figure 2.6 : Exemple d'un graphe dynamique sur 3 pas de temps avec poids correspondant au modèle décrit figure 2.4.

2.3 Définitions et notions importantes

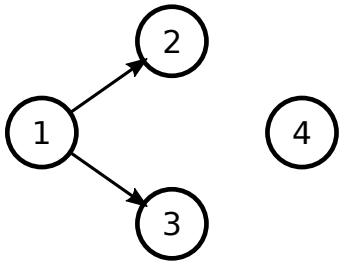
Après avoir présenté les modèles de graphes dynamiques que l'on considère, nous définissons maintenant formellement les notions utilisées tout au long de ce manuscrit.

2.3.1 Définitions générales du graphe dynamique

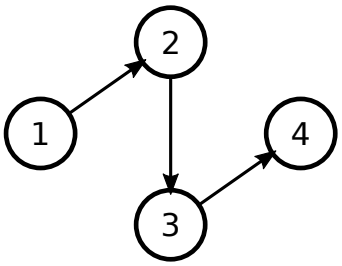
Un graphe dynamique est défini sur un intervalle de temps que l'on nomme *intervalle d'étude* (définition 1). L'horizon de temps (définition 2) d'un graphe est le dernier pas de temps auquel il est considéré. Un graphe dynamique (définition 4) est une séquence de graphes statiques définis sur le même ensemble de nœuds. Nous nommons ces graphes statiques des *t-graphes* (définition 3), que l'on peut aussi appeler en anglais *snapshot graphs*. Nous appelons *graphe sous-jacent* (définition 5) le graphe statique qui contient tous les nœuds et toutes les arêtes appartenant au graphe dynamique à un moment au moins de



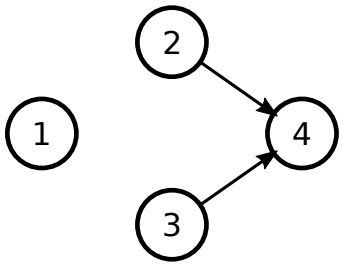
(a) Graphe dynamique avec les pas de temps de présence indiqués sur les arcs.



(b) Premier pas de temps.



(c) Deuxième pas de temps.



(d) Troisième pas de temps.

Figure 2.7 : Exemple d'un graphe dynamique sur 3 pas de temps sans poids correspondant au modèle décrit figure 2.5.

l'intervalle d'étude. Cette notion est utilisée pour la résolution de problèmes spécifiques ou encore lors de la génération aléatoires de graphes dynamiques.

Définition 1 (Intervalle d'étude)**.** *Un intervalle d'étude $\mathcal{T} = \{1 \dots T\}$ est un ensemble discret de T pas de temps sur lequel un graphe dynamique est étudié.*

Définition 2 (Horizon de temps)**.** *L'horizon de temps d'un graphe dynamique est la fin de l'intervalle d'étude.*

Définition 3 (t-graphe)**.** *Un t-graphe, noté G_θ, $\theta \in \mathcal{T}$, est un graphe statique correspondant à l'état d'un graphe dynamique G à un pas de temps θ donné.*

Définition 4 (Graphe dynamique)**.** *Un graphe dynamique $G = (G_\theta)_{\theta \in \mathcal{T}}$ est défini sur un intervalle d'étude $\mathcal{T} = \{1 \dots T\}$. C'est une succession de t-graphes $G_\theta = (V, E_\theta)$, $\theta \in \mathcal{T}$.*

Définition 5 (Graphe sous-jacent)**.** *Le graphe sous-jacent d'un graphe dynamique $G = (G_\theta)_{\theta \in \mathcal{T}}$ est le graphe statique $G_{SJ} = (V, E_{SJ})$ où $E_{SJ} = \{(i; j) \mid \exists \theta, (i; j) \in E_\theta\}$.*

2.3.2 Les trajets

Dans un graphe statique, un chemin est une succession d'arcs $e_1 \dots e_l$ tels que $e_\lambda = (i; j)$, $e_{\lambda+1} = (j; k)$. Dans le cas d'un graphe non orienté, on parle de chaine.

On peut alors s'interroger sur l'extension aux graphes dynamiques de la notion de chemin. C'est l'objet des travaux de Bui-Xuan et al. (2003). Les auteurs définissent l'équivalent, dans un graphe dynamique, au chemin d'un graphe statique, et le nomment *journey*, que l'on traduira en français par le terme *trajet*.

Définition 6 (Temps de traversée)**.** *On définit le temps de traversée d'un arc ou d'une arête ij dans un graphe dynamique comme le temps nécessaire pour traverser l'arc ou l'arête du nœud i vers le nœud j.*

En notant $\tau_\theta(i;j)$ le temps de traversée de l'arc $(i;j)$ à la date θ, le départ du nœud i à la date θ implique une arrivée au nœud j à la date $\theta + \tau_\theta(i;j)$.

Les trajets sont très liés aux temps de traversée des arcs du graphe. L'idée générale est de se déplacer sur les nœuds du graphe dans le temps.

Définition 7 (Trajet (d'après Bui-Xuan et al. (2003)))**.** *Un trajet est une séquence ordonnée d'arcs à des dates de l'intervalle d'étude tels que deux arcs successifs $(i;j)_\theta$ (l'arc $(i;j)$ à la date θ) et $(j;k)_{\theta'}$ (l'arc $j;k$ à la date θ') vérifient :*

1. *l'arc statique $(i;j) \in E_\theta$ du t-graphe G_θ*
2. *l'arc statique $(j;k) \in E_{\theta'}$ du t-graphe $G_{\theta'}$*
3. *$\theta + \tau_\theta(i;j) \leqslant \theta'$ si une pause est autorisée sur les nœuds*
   $\theta + \tau_\theta(i;j) = \theta'$ si les pauses ne sont pas autorisées sur les nœuds

On notera que dans cette définition, les indices des pas de temps correspondent à des dates. On a donc que la valeur $\theta' - \theta$ correspond au temps écoulé entre deux dates θ et θ'. On considère qu'une unité de temps s'est écoulée entre deux t-graphes successifs.

Un trajet d'un nœud i à un nœud j est une suite d'arcs dont le premier part de i à une date $\theta_{début}$ et le dernier arrive en j à une date θ_{fin}, et qui traverse d'autres nœuds du graphe.

On peut imaginer deux cas possibles, décrits dans la condition 3 de la définition 7. Dans le premier cas un trajet peut rester « arrêté » sur un nœud pendant un certain temps. Dans le second cas, un trajet arrivant sur un nœud à une date donnée doit nécessairement repartir de ce nœud à cette même date.

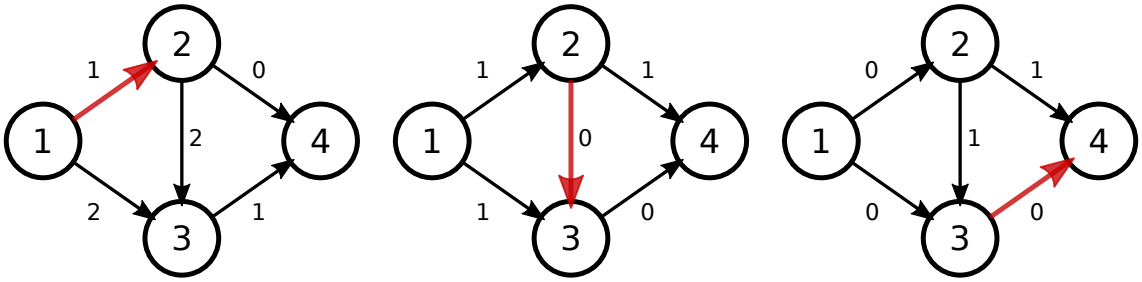
La figure 2.8 montre un graphe dynamique avec des temps de traversée. Les arcs en rouge indiquent le trajet pour aller du nœud 1 vers le nœud 4. On emprunte l'arc $(1;2)$ au premier pas de temps, qui arrive au nœud 2 à la date 2 puisque cet arc a un temps de traversée de 1. On emprunte ensuite l'arc $(2;3)$ au deuxième pas de temps, qui arrive au nœud 3 à la date 2 car le temps de traversée de cet arc est 0. On reste ensuite arrêté sur le nœud 3 pendant une unité de temps puis on emprunte enfin l'arc $(3;4)$ au troisième pas de temps qui arrive au nœud 4 à la date 3 car le temps de traversée de cet arc est 0.

La figure 2.9 montre le même graphe dynamique. Cette fois-ci, les pauses sur les nœuds au cours d'un trajet sont interdites. En rouge est indiqué un trajet possible pour aller du nœud 1 vers le nœud 4. Ce trajet emprunte d'abord l'arc $(1;3)$ au deuxième pas de temps qui arrive au nœud 3 à la date 3 puisque le temps de traversée de cet arc est 1. Le trajet emprunte ensuite l'arc $(3;4)$ au troisième pas de temps qui arrive au nœud 4 à la date 3 car le temps de traversée de cet arc est 0.

Le trajet de la figure 2.8 fait une pause sur le nœud 3 pendant un pas de temps et celui de la figure 2.9 traverse les nœuds sans s'y arrêter.

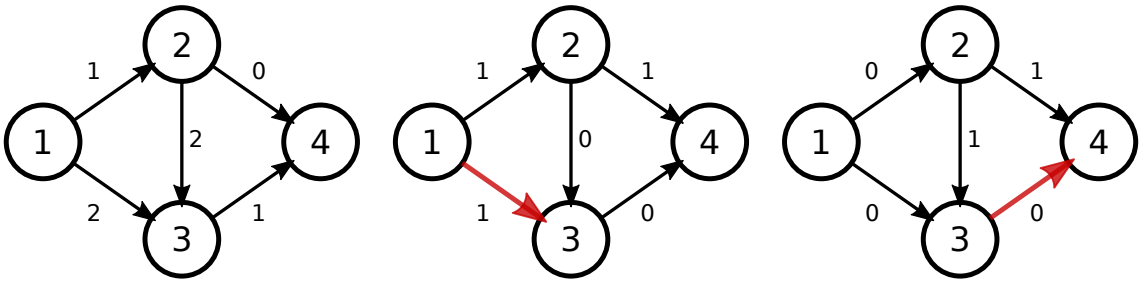
Si l'on considère des graphes dynamiques sans temps de traversée, les mêmes définitions restent entièrement valables. En effet, il s'agit simplement d'avoir $\tau_\theta(i;j) = 0\ \forall \theta \in \mathcal{T}$ et $\forall i, j \in V$.

Dans ce cas-là, un trajet d'un nœud i à un nœud j, commençant à une date $\theta_{début}$ et terminant à une date θ_{fin}, peut être vu comme une succession de chemins (au sens statique) P_θ sur chacun des t-graphes G_θ, pour $\theta_{début} \leqslant \theta \leqslant \theta_{fin}$. Le chemin $P_{\theta_{début}}$ démarre sur le nœud i dans le t-graphe $G_{\theta_{début}}$. Le chemin $P_{\theta_{fin}}$ termine sur le nœud j dans le t-graphe $G_{\theta_{fin}}$. Le chemin P_θ dans G_θ, pour $\theta_{début} \leqslant \theta < \theta_{fin}$ se termine sur un nœud l de façon à ce que le chemin $P_{\theta+1}$ dans $G_{\theta+1}$ commence sur ce même nœud l. Il peut éventuellement exister des dates θ' telles que le chemin $P_{\theta'}$ est vide. La figure 2.10 illustre un trajet dans un graphe dynamique sans temps de traversée avec des pauses autorisées sur les nœuds.



(a) Premier pas de temps. (b) Deuxième pas de temps. (c) Troisième pas de temps.

Figure 2.8 : Exemple de trajet dans un graphe dynamique avec temps de traversée sur les arêtes et pauses autorisées sur les nœuds. Les nombres à côté des arêtes correspondent au temps de traversée de l'arête. Le trajet du nœud 1 au nœud 4 est en rouge et court sur plusieurs pas de temps.



(a) Premier pas de temps. (b) Deuxième pas de temps. (c) Troisième pas de temps.

Figure 2.9 : Exemple de trajet dans un graphe dynamique avec temps de traversée sur les arêtes et pauses interdites sur les nœuds. Les nombres à côté des arêtes correspondent au temps de traversée de l'arête. Le trajet du nœud 1 au nœud 4 est en rouge et court sur plusieurs pas de temps.

Dans le cas où les pauses sur les nœuds ne sont pas autorisées, alors cela signifie que le trajet doit « repartir » de chaque nœud immédiatement. Mais sans temps de trajet, cela implique nécessairement que tout trajet part et arrive à une même date. Autrement dit, un trajet ne peut pas partir d'un nœud i à une date θ et arriver à un nœud j à une date $\theta' \neq \theta$.

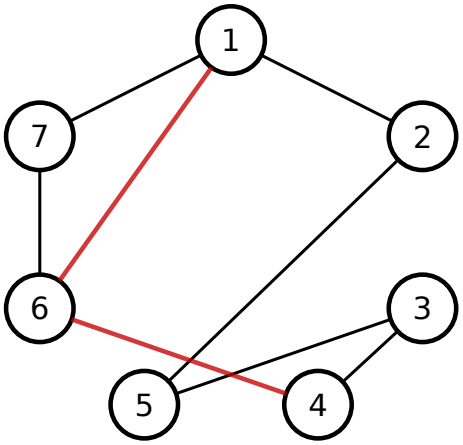
Dans un graphe statique non orienté, l'existence d'une chaine d'un nœud i vers un nœud j implique nécessairement l'existence d'une chaine de j vers i. Dans un graphe statique orienté, cette implication n'est évidemment plus valable.

Il est intéressant de noter que dans un graphe dynamique, puisque les arêtes ou arcs peuvent apparaitre et disparaitre au cours du temps, alors celui-ci impose une orientation à tous les trajets, y compris dans un graphe non orienté. Le trajet est donc une notion orientée.

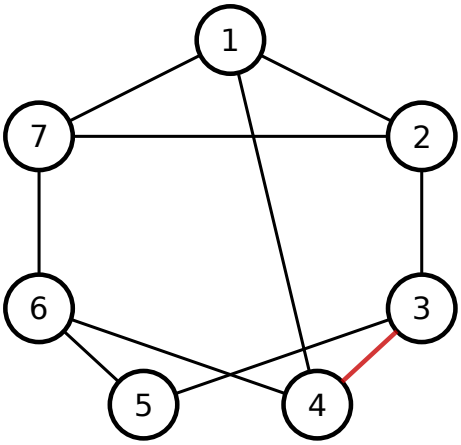
La figure 2.11 montre un graphe dynamique sans temps de traversée sur deux pas de temps. Il existe un trajet du nœud 1 vers le nœud 3 passant par l'arête $(1;2)$ au premier pas de temps et par l'arête $(2;3)$ au deuxième pas de temps. On peut observer qu'il n'existe pas de trajet du nœud 3 vers le nœud 1.

Bui-Xuan et al. (2003), qui ont défini la notion de trajet, proposent plusieurs extensions possibles du problème de plus court chemin que l'on connait dans les graphes statiques.

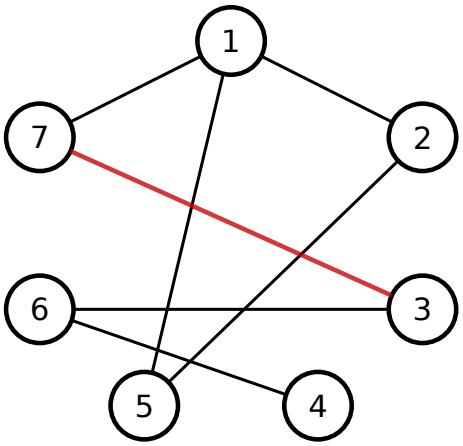
Ils définissent le trajet le plus court (en anglais *shortest journey*) d'un nœud i vers un nœud j comme étant, parmi tous les trajets existants dans le graphe de i vers j, celui qui traverse le moins d'arêtes.



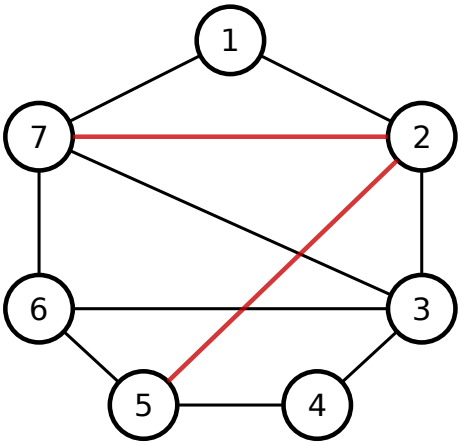
(a) G_1. Le chemin P_1 emprunte l'arête (1;6) puis l'arête (6;4).



(b) G_2. Le chemin P_2 emprunte l'arête (4;3).

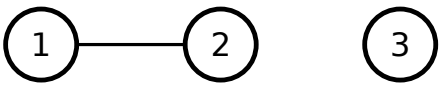


(c) G_3. Le chemin P_3 emprunte l'arête (3;7).

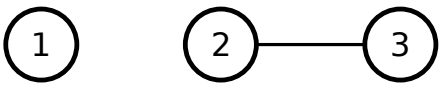


(d) G_4. Le chemin P_4 en rouge emprunte l'arête (7;2) puis l'arête (2;5).

Figure 2.10 : Exemple de trajet dans un graphe dynamique sans temps de traversée. Le trajet du nœud 1 au nœud 5 est composé des chemins P_1, P_2, P_3 et P_4, en rouge.



(a) G_1



(b) G_2

Figure 2.11 : Graphe dynamique sur deux pas de temps, sans temps de traversée.

Les auteurs définissent le trajet au plus tôt (en anglais *foremost journey*) d'un nœud i vers un nœud j comme étant, parmi tous les trajets existants dans le graphe de i vers j, celui qui arrive à j en premier.

Bui-Xuan et al. définissent le trajet le plus rapide (en anglais *fastest journey*) d'un nœud i vers un nœud j comme étant, parmi tous les trajets existants dans le graphe de i vers j, celui pour lequel le temps écoulé entre le départ de i et l'arrivée à j est le plus faible.

La notion de trajet dans un graphe dynamique est une notion très importante qui peut être utilisée dans beaucoup d'applications bien sûr, mais aussi dans la résolution d'autres problèmes sur les graphes. En particulier, c'est une notion qui peut être utilisée pour les problèmes de flots ainsi que pour les problèmes de connexité que nous aborderons dans les chapitres 3 et 4, respectivement.

2.3.3 Le graphe développé

Le graphe développé est un outil permettant de représenter un graphe dynamique par un graphe statique sans perte d'informations. Il peut aussi permettre, si le graphe n'est pas trop grand, de visualiser clairement des trajets (nous donnons des exemples en fin de section) ou des flots (nous revenons sur ce point en chapitre 3). Cet outil a été utilisé dans la littérature notamment par Minieka (1973) dès les années 1970, mais aussi plus récemment par Parpalea et Ciurea (2011) par exemple. Nous le définissons formellement dans la définition 8.

Définition 8 (Graphe développé)**.** *Le graphe développé G^{DVP} d'un graphe dynamique G est un graphe statique représentant G sans perte d'informations :*

- *À chaque nœud i de G et chaque pas de temps θ de $\mathcal{T}$ de G correspond un nœud i_θ dans G^{DVP}.*
- *À chaque arc $(i;j)$ de G au pas de temps θ et de temps de traversée $\tau_\theta(i;j)$ correspond un arc $i_\theta j_{\theta+\tau_\theta(i;j)}$ dans G^{DVP}.*

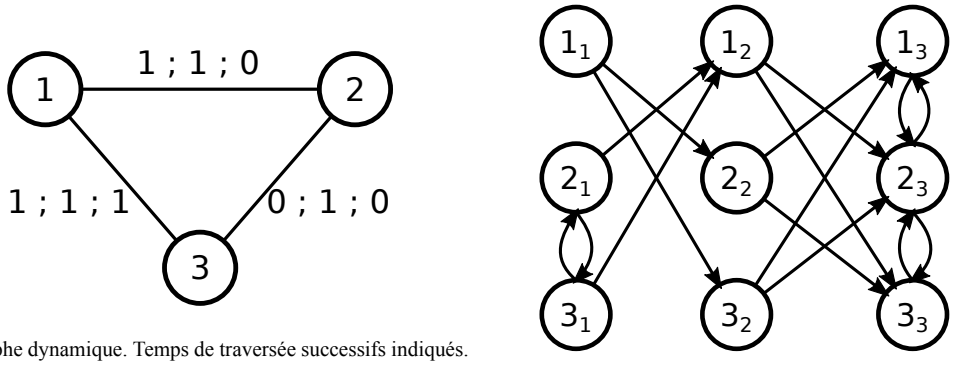
Tous les nœuds et tous les arcs du graphe dynamique sont dupliqués pour former le graphe développé. De plus, selon le modèle de graphe étudié, il est possible d'ajouter des arcs de la forme $i_\theta i_{\theta+1}$. Ces arcs permettent de représenter le fait de « rester » sur un nœud i. Par exemple, dans le cas d'un trajet, cela représente une pause possible sur un nœud. Pour du flot, cela représente le fait de stocker des unités de flot sur un nœud.

Si le graphe dynamique est non-orienté, alors chaque arête du graphe dynamique sera représentée par deux arcs dans le graphe développé. En effet, comme une arête du graphe dynamique permet de se déplacer dans l'espace (vers d'autres nœuds) et dans le temps (en arrivant à une date ultérieur), une arête $(i;j)$ à la date θ et de temps de traversée $\tau_\theta(i;j)$ dans le graphe dynamique sera représentée par un arc $(i_\theta; j_{\theta+\tau_\theta(i;j)})$ et un arc $(j_\theta; i_{\theta+\tau_\theta(ij)})$.

On note que si l'on considère des graphes sans temps de traversée, alors cela équivaut à des temps de traversée nuls. Dans ce cas, il est aussi possible de construire le graphe développé et un arc $(i;j)$ à la date θ dans le graphe dynamique est représenté par un arc $(i_\theta; j_\theta)$ dans le graphe développé.

Le graphe développé représente donc, sous la forme d'un graphe statique, un graphe dynamique. Pour un graphe dynamique de n nœuds, m arcs ou arêtes et T pas de temps, alors le graphe développé aura $T \cdot n$ nœuds et de l'ordre de $T \cdot m$ arcs.

On présente dans la figure 2.12 un graphe dynamique et son graphe développé associé. Le graphe dynamique est présenté en figure 2.12a. Il est non orienté, possède 3 nœuds et est défini sur 3 pas de temps. Sur chaque arc sont notés les temps de traversée successifs. La figure 2.12b présente le graphe développé correspondant. Chacun des nœuds est dupliqué pour chaque pas de temps. Comme le graphe dynamique n'est pas orienté, le graphe développé possède deux fois plus d'arcs que le graphe dynamique n'a d'arêtes à chaque pas de temps. Les temps de traversée sont représentés par les nœuds auxquels sont reliés les arcs. Par exemple l'arc (1;3) au premier pas de temps, de temps de traversée 1 est représenté



(a) Graphe dynamique. Temps de traversée successifs indiqués.

(b) Graphe développé correspondant.

Figure 2.12 : Illustration d'un graphe dynamique et de son graphe développé correspondant.

dans le graphe développé par les arcs $(1_1;3_2)$ et $(3_1;1_2)$. Il en va de même pour toutes les arêtes du graphe dynamique. L'arête $(1;3)$ au dernier pas de temps n'est pas représentée sur le graphe développé car le temps de traversée de 1 fait que la date d'arrivée n'est pas dans l'intervalle d'étude. Le graphe développé a 9 nœuds (3 nœuds sur 3 pas de temps) et 16 arcs (2 fois 3 arêtes sur 3 pas de temps sans l'arête $(1;3)$ du dernier pas de temps).

Le graphe développé est un outil utile. Puisqu'il permet de modéliser les graphes dynamiques par des graphes statiques il permet donc de les traiter comme tels pour certains problèmes. On peut ainsi utiliser des méthodes connues pour les graphes statiques au lieu de développer de nouvelles méthodes spécifiques aux graphes dynamiques.

L'inconvénient principal de l'utilisation du graphe développé est sa taille. Cela aura donc un impact important sur la complexité des algorithmes utilisés.

Un trajet dans le graphe dynamique correspond à un chemin (statique) dans le graphe développé. La recherche d'un trajet peut donc se faire grâce à la recherche d'un chemin. La figure 2.13 montre le graphe développé correspondant au graphe dynamique de la figure 2.8. Le même trajet est identifié par le chemin en rouge. La pause sur le nœud 3 est représentée par l'arc horizontal $(3_2;3_3)$. De la même manière, la figure 2.14 montre le graphe développé correspondant au graphe dynamique de la figure 2.9. Le même trajet est identifié par le chemin en rouge. Puisque les pauses sur les nœuds ne sont pas autorisées dans ce cas, les arcs horizontaux de la forme $(i_\theta; i_{\theta+1})$ n'existent pas.

Un flot dans un graphe dynamique correspond à un flot dans le graphe développé associé. Nous revenons plus en détails sur ce point dans le chapitre 3.

2.4 Exemples de problèmes et classification des méthodes de résolution

2.4.1 Différents problèmes

Nous avons discuté, en section 2.2, de la diversité des travaux traitant de graphes dynamiques et donc de la diversité qui pouvait exister dans le vocabulaire utilisé.

On trouve aussi dans la littérature de nombreux problèmes de graphes étudiés en contexte dynamique. Depuis l'étude des flots au cours du temps avec les travaux de Ford et Fulkerson (1958, 1962) dès les années 1950, on a vu l'émergence de travaux sur différents types de problèmes. Holme (2015) a réalisé une étude très complète des travaux existants sur les graphes dynamiques selon les domaines. Nous citons ici en exemples quelques problèmes de graphes traités dans un contexte dynamique.

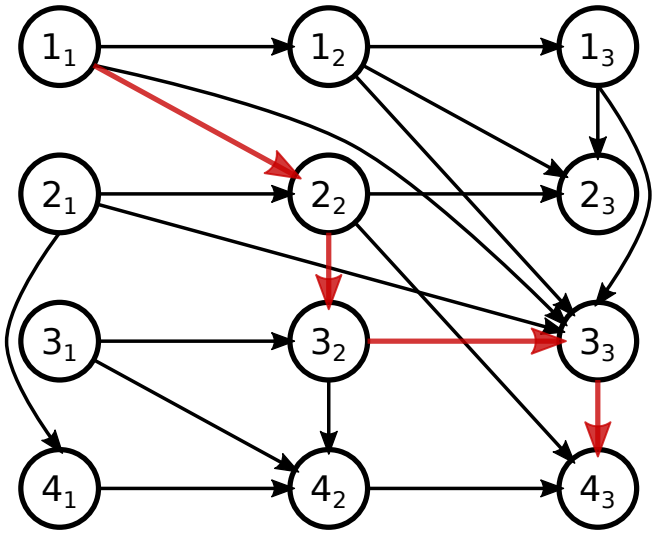


Figure 2.13 : Illustration du graphe développé correspondant au graphe dynamique de la figure 2.8. Le même chemin est aussi coloré en rouge.

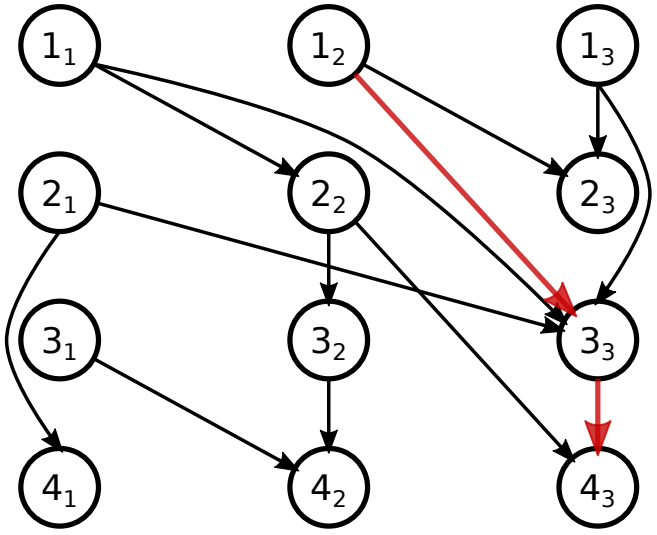


Figure 2.14 : Illustration du graphe développé correspondant au graphe dynamique de la figure 2.9. Le même chemin est aussi coloré en rouge.

Comme nous l'avons présenté en section 2.3.2, Bui-Xuan et al. (2003) s'intéressent à la notion de chemin et à son extension dans les graphes dynamiques : les trajets. Ils proposent trois manières de définir un *plus court chemin* dans un graphe dynamique.

Les problèmes de plus courts chemins ont différentes applications. Ce problèmes a donc été traité aussi du point de vue applicatif dans un contexte dynamique. On pourra citer notamment Nannicini et al. (2010) qui s'intéressent à la gestion de trafic routier.

Le problème du voyageur de commerce est un problème d'optimisation dans les graphes qui est NP-complet. Michail et Spirakis (2016) s'intéressent à ce problème dans les graphes dynamiques. Ils considèrent un modèle de graphe dynamique dans lequel les nœuds restent présents et où le graphe est complet à chaque pas de temps. Chaque arête du graphe a un poids qui varie dans le temps et vaut 1 ou 2. L'objectif est de trouver un cycle de poids minimum qui parcourt tous les nœuds du graphe exactement une fois. On considère qu'une seule arête au maximum peut être traversée à chaque pas de temps. Ils proposent des résultats d'approximation pour ce problème.

Kovanen et al. (2011) proposent une définition de motifs temporels dans un graphe dynamique ainsi qu'un algorithme pour les identifier. Cela a été repris par Oberoi (2019) qui s'intéresse au trafic routier et propose de détecter des motifs temporels dans un graphe dynamique modélisant l'évolution du trafic. La détection de motifs est un problème d'isomorphisme de sous-graphe, ce qui, dans un graphe statique, est un problème de la classe NP.

On peut aussi citer Bampis et al. (2018) qui s'intéressent au problème de couplage dans un contexte dynamique, (le problème a été originellement défini par Gupta et al. (2014) qui s'intéressent aux matroïdes). Ils travaillent sur un graphe dynamique vu comme une séquence ordonnée de graphes statiques définis sur le même ensemble de nœuds. Les arêtes possèdent un cout variable dans le temps. Ils cherchent une séquence de couplages parfaits, un pour chaque graphe statique. Un cout est associé à chaque couplage selon le cout des arêtes sélectionnées. Un cout est aussi associé à la transition entre deux couplages successifs selon le nombre d'arêtes ajoutées. L'objectif étant de trouver une séquence de couplages qui minimise le cout total. Ils prouvent notamment que le problème est difficile à approximer, même dans le cas des graphes bipartis et avec deux pas de temps seulement.

Baste et al. (2020) considèrent aussi le problème de couplage dans un graphe dynamique. Contrairement à Bampis et al., ils ne cherchent pas une séquence de couplages à chaque pas de temps du graphe. Ils cherchent un couplage dynamique de cardinalité maximum, qu'ils nomment γ-couplage. Ils ne considèrent pas de couts sur les arêtes. Ils définissent une γ-arête commençant à la date θ comme une arête présente pendant γ pas de temps consécutifs à partir de la date θ. Ils définissent un γ-couplage comme un ensemble de γ-arêtes tel qu'un nœud donné à un pas de temps donné ne peut pas appartenir à deux γ-arêtes distinctes. Ils montrent que le problème de décision associé est NP-complet pour tout $\gamma > 1$. Ils proposent un algorithme paramétré selon la taille de la solution ainsi qu'un algorithme d'approximation.

Les problèmes de flots ont été massivement étudiés et il existe beaucoup de travaux traitant ce type de problèmes dans un contexte dynamique. Nous revenons sur cette tranche de la littérature plus spécifiquement dans le chapitre 3 qui est consacré aux problèmes de flots.

Des travaux concernant la connexité dans les graphes dynamiques ont aussi été réalisés, même si la littérature à ce sujet est un peu plus maigre. Le chapitre 4 est consacré à la connexité dans les graphes dynamiques, et nous y présenterons donc la littérature existante.

2.4.2 Différentes approches

Il existe de nombreux problèmes de graphes étudiés dans un contexte dynamique. Mais on peut identifier trois types de problèmes différents selon l'approche utilisée :

1. Le problème se résout grâce à T résolutions indépendantes, une sur chaque t-graphe.
2. Le problème se ramène à un problème sur un graphe statique.
3. Le problème nécessite une méthode spécifique.

On notera que la manière dont le problème est étendu sur un graphe dynamique influe sur la catégorie, parmi celles énoncées, dans laquelle se trouve le problème. Et donc, certains problèmes peuvent se trouver dans plusieurs catégories, voire les trois, c'est par exemple le cas des flots.

Catégorie 1 Ces problèmes pourront être résolus en résolvant T problèmes de graphes statiques.

Si l'on considère un flot maximum dans un graphe dynamique sans temps de traversée et sans stockage de flot sur les nœuds (nous revenons sur cette notion dans le chapitre 3), alors le problème est équivalent à résoudre T problèmes de flot maximum, un à chaque pas de temps de l'intervalle d'étude.

Si l'on considère le problème de couplage décrit par Bampis et al. (2018) en ignorant les couts associés à la transition entre un couplage à un pas de temps et le couplage au pas de temps suivant, alors ce problème se résout comme T problèmes de couplages parfaits indépendant.

Catégorie 2 Certains problèmes peuvent se ramener à un problème dans un graphe statique. C'est-à-dire qu'il existe un graphe statique et un problème associé qui permettent de représenter le problème de graphe dynamique que l'on souhaite résoudre. Dans ce cas, la résolution du problème (dynamique) se fait en résolvant un problème sur un graphe statique.

Les graphes développés, présentés en section 2.3.3, permettent de représenter par un graphe statique de $T \cdot n$ nœuds et $T \cdot m$ arêtes un graphe dynamique sur n nœuds, m arêtes et T pas de temps. Grâce à eux, on peut trouver des chemins, comme on l'a vu en section 2.3.3, mais on peut aussi résoudre des problèmes de flot quel que soit le modèle de graphe dynamique. Nous verrons ce point en particulier dans le chapitre 3.

Si l'on s'intéresse au problème du stable maximum dans un graphe dynamique, alors on cherche un ensemble de nœuds tels qu'ils forment un stable sur tout l'intervalle d'étude, comme énoncé dans le chapitre 1. Pour cela, ils ne doivent jamais être connectés. C'est donc équivalent à chercher un stable maximum sur le graphe sous-jacent. Ce problème se résout facilement car il se ramène à un problème sur un graphe statique.

Catégorie 3 Certains problèmes ne peuvent pas être résolus simplement sur un graphe statique, que ce soit en résolvant T problèmes ou un seul. Pour ceux-là, il est nécessaire de concevoir un algorithme spécifique dédié aux graphes dynamiques.

On peut noter que les problèmes de flots, bien qu'ils puissent être résolus grâce au graphe développé, se situent aussi dans cette catégorie. En effet, comme nous le verrons, la résolution par graphe développé est très couteuse. Donc cela peut être bien plus avantageux de trouver des méthodes spécifiques pour les graphes dynamiques.

Le problème de couplages décrit par Bampis et al. (2018) et celui décrit par Baste et al. (2020) nécessitent des algorithmes (d'approximation) spécifiques. C'est aussi le cas du problème du voyageur de commerce décrit par Michail et Spirakis (2016).

Notre travail se situe dans cette troisième catégorie. Nous nous intéressons aussi aux problèmes présents dans la catégorie 2 pour lesquels une résolution, bien que faisable, est trop couteuse. Dans tous les cas, les problèmes que nous considérons peuvent grandement bénéficier d'un algorithme dédié aux graphes dynamiques.

Dans cette optique, nous avons étudié des problèmes de flots en souhaitant éviter l'utilisation du graphe développé (catégorie 2 dans la liste précédente), et nous avons étudié des problèmes de connexité dont la résolution semble impossible par un ou des graphes statiques (catégorie 3 dans la liste précédente).

Chapitre 3

Flots

Les problèmes de flots dans les graphes ont été largement étudiés dans le cas statique, et ce depuis plusieurs décennies. L'ouvrage de Ahuja et al. (1993) est consacré aux flots et présente les problèmes ainsi que différentes méthodes de résolution.

Ces problèmes sont simples à présenter. Il s'agit d'envoyer dans un graphe une certaine quantité de flot depuis un nœud source, en général noté s, vers un nœud puits, en général noté t. Ce qui définit le type de problème considéré, ce sont les contraintes à respecter et la fonction objectif à optimiser.

Toutes les possibilités de modélisation supplémentaires offertes par les graphes dynamiques amènent à s'interroger sur l'extension aux graphes dynamiques des problèmes que l'on connait, comme les problèmes de flots, dans un contexte dynamique. C'est l'objet de ce chapitre.

Comment définir un flot dans un graphe dynamique ? Quels sont les types de problèmes d'optimisation que l'on rencontre ? Comment résoudre ces problèmes ? Ce sont autant de questions auxquelles ce chapitre tente d'apporter des réponses.

En section 3.1, nous passons en revue la littérature existante sur les flots dans un contexte dynamique. Nous explorons le problème du flot maximum en section 3.2. La section 3.3 présente les résultats obtenus sur le problème du flot de cout minimum. Nous apportons enfin une conclusion à notre démarche et nos travaux sur les problèmes de flot dans les graphes dynamiques en section 3.4.

3.1 État de l'art et généralités

Il existe des algorithmes efficaces pour les problèmes de flot dans les graphes statiques. Le problème du flot maximum se résout à l'optimum par des algorithmes polynomiaux quand le problème du flot de cout minimum se résout optimalement par des algorithmes pseudo-polynomiaux qui sont en pratique très efficaces. Voir Ahuja et al. (1993) pour avoir un récapitulatif des algorithmes les plus efficaces pour les problèmes de flot.

L'ajout d'une dimension temporelle au modèle de graphe étudié peut complexifier le problème. En effet, devoir gérer le temps pour obtenir une solution peut obliger à repenser la méthode de résolution.

Cette section est consacrée à une présentations générale des problèmes de flot dans les graphes dynamiques avec une revue de la littérature existante sur le sujet.

Nous abordons en premier les modèles de graphes dynamiques appropriés aux problèmes de flots, puis traitons des différents problèmes de flot que l'on peut rencontrer dans des graphes dynamiques. Nous discutons enfin des méthodes de résolution et algorithmes proposés dans la littérature.

3.1.1 Modèles de graphes dynamiques pour les problèmes de flot

Comme nous l'avons vu dans le chapitre précédent, il est possible de créer une multitude de modèles de graphes dynamiques différents. C'est ce qui a été présenté en figure 2.2.

Quand on s'intéresse aux flots dans les graphes statiques, on distingue en général deux types de poids sur les arcs. On trouve d'une part la capacité, et d'autre part le cout unitaire. La capacité correspond à la quantité maximum de flot qui peut traverser un arc, exprimée en unité de flot. Le cout d'un arc

correspond au prix imputé à la traversée d'une unité de flot sur l'arc. On retrouve naturellement ces deux types de poids sur les arcs dans les graphes dynamiques.

Il est important de noter qu'une capacité variable sur un arc peut permettre de modéliser l'absence d'un arc sans nécessairement prendre en compte l'apparition et la disparition des arcs dans le modèle de graphe. En effet, si la capacité d'un arc vaut 0 à un pas de temps θ donné, alors celui-ci ne pourra pas être emprunté, ce qui équivaut à l'absence de cet arc.

Puisque le graphe est étudié dans le temps, on peut aussi trouver, dans les graphes dynamiques pour les problèmes de flot, un autre type de poids sur les arcs : le temps de traversée. Celui-ci correspond au nombre d'unités de temps nécessaires pour qu'une unité de flot traverse l'arc $(i; j)$. En d'autres termes, il s'agit du temps écoulé entre l'instant où l'unité part du nœud i et l'instant où l'unité arrive sur le nœud j.

On peut imaginer que, pour un modèle donné, certains de ces éléments soient absents. S'ils sont présents, ils peuvent être constants au cours du temps et ne jamais varier ou bien ils peuvent être fonction du temps et évoluer tout au long de l'intervalle d'étude du graphe.

Dans le cas du temps de traversée, son absence correspond soit à un temps de traversée nul tout au long de l'intervalle d'étude soit à un temps de traversée négligeable en comparaison de la dynamique du graphe. C'est-à-dire que soit les unités de flot traversent les arcs instantanément, soit elles les traversent très largement plus vite que le graphe n'évolue. Ce deuxième cas se retrouve par exemple sur les réseaux télécoms où la traversée d'un arc se fait dans un temps de l'ordre de la microseconde ou milliseconde quand le réseau évolue sur plusieurs secondes ou minutes, voire plus. Même les réseaux logistiques peuvent être dans ce cas-là, en effet, une traversée en quelques heures est négligeable face à des changements d'infrastructure qui prennent plusieurs mois.

Les graphes statiques étudiés dans le cas des problèmes de flot n'ont, en général, pas de poids sur les nœuds. Cependant, la dimension dynamique du graphe offre plus de possibilités dans la modélisation. En effet, les unités de flots voyagent dans l'espace (sur les nœuds du graphe), mais aussi dans le temps (au cours de l'intervalle d'étude).

Le fait que le flot voyage dans le temps offre la possibilité qu'il puisse s'arrêter à certains moments. En ce sens, les nœuds peuvent potentiellement stocker des unités de flot. Lorsqu'une unité de flot est stockée sur un nœud i, cela signifie que lorsque qu'elle arrive sur i à un temps θ, alors elle peut repartir de i à un temps $\theta' \geqslant \theta$. Le stockage sur les nœuds est exprimé via une capacité de stockage qui correspond au nombre maximum d'unités de flot qui peuvent être arrêtées sur un nœud en même temps.

Le stockage sur les nœuds peut soit être autorisé, soit être interdit. Si le stockage est autorisé, alors la capacité de stockage des nœuds peut être fixe au cours du temps ou bien être fonction du temps. La capacité de stockage peut aussi être infinie, auquel cas il n'y a pas de limite quant au nombre d'unité de flot sur un nœud en même temps. Si le stockage est interdit, les unités de flot qui arrivent sur i au temps θ doivent repartir de i au même temps θ.

Aux unités de flots stockées sur un nœud peut être imputé un cout de stockage, qui correspond à la conservation, sur un nœud donné, d'une unité de flot pendant une unité de temps. De la même manière que pour les couts de traversée des arcs, ce cout de stockage peut être nul, constant au cours du temps ou variable.

On notera par ailleurs que, dans le cas de modèles de graphes avec des temps de traversée sur les arcs, il est possible de modéliser le stockage par une boucle sur chaque nœud dont le temps de traversée vaut 1 pendant tout l'intervalle d'étude du graphe. La capacité et le cout de cet arc valent alors la capacité de stockage et le cout de stockage du nœud sur lequel boucle cet arc. Cela permet d'éviter d'introduire de nouvelles notations spécifiques au stockage. Et cela évite aussi un traitement particulier du stockage puisque celui-ci est représenté par des arcs ordinaires dans le graphe.

Pour l'intégralité de ce chapitre, nous utiliserons les mêmes notations pour les paramètres du graphe.

s	:	nœud source
t	:	nœud puits
$u_\theta(i;j)$	:	capacité de l'arc $(i;j)$ à la date θ
$\tau_\theta(i;j)$	:	temps de traversée de l'arc $(i;j)$ à la date θ
$g_\theta(i)$	:	capacité de stockage du nœud i de la date θ à la date $\theta+1$
$c_\theta(i;j)$	:	cout de traversée de l'arc $(i;j)$ à la date θ
$q_\theta(i)$	:	cout de stockage du nœud i de la date θ à la date $\theta+1$

Les variables seront notées :

x	:	valeur totale du flot
$f_\theta(i;j)$	:	flot sur l'arc $(i;j)$ partant de i à la date θ
$y_\theta(i)$	:	stockage sur le nœud i de la date θ à la date $\theta+1$

Quel que soit le problème à résoudre, le flot respecte toujours les mêmes contraintes. Dans le cas statique, le flot doit respecter les capacités des arcs, c'est à dire qu'on ne peut pas faire passer sur un arc plus d'unités de flot qu'autorisé par sa capacité. Sur chaque nœud, on doit avoir conservation du flot, c'est-à-dire que le nombre d'unités de flot qui entrent dans le nœud est strictement égal au nombre d'unités de flot qui en sortent, et le nombre d'unités partant du nœud source est strictement égal au nombre d'unités arrivant au nœud puits. Ces contraintes sont toujours présentes dans le cas dynamique, qui a une contrainte spécifique supplémentaire qui concerne le respect des capacités de stockage en chaque nœud.

Voici, ci-dessous, les contraintes qu'un problème de flot dans un graphe dynamique doit respecter. La fonction objectif du modèle d'optimisation dépend du problème de flot à résoudre.

Comme dans le cas statique, l'équation (3.1) assure le respect des capacités des arcs. La différence avec le cas statique est la nécessité d'indexer le flot et les capacités sur le temps.

L'équation (3.2) garantit que tout le flot qui part de la source arrive au puits, et ce, avant l'horizon de temps T. Dans le cas statique, cette contrainte est une implication de la conservation du flot, mais dans le cas dynamique on doit s'assurer que tout le flot est arrivé avant l'horizon de temps.

L'équation (3.3) assure la conservation du flot. Tout ce qui arrive au nœud i à la date θ doit soit partir de i, soit être stocké sur i à la date θ. À cause des temps de trajets qui sont différents d'un arc à un autre et d'un pas de temps à un autre, il est nécessaire d'avoir une somme sur les dates afin de connaitre tout le flot qui arrive à une date donnée.

Et enfin l'équation (3.4), spécifique au cas dynamique, garantit que la capacité de stockage n'est jamais excédée. Comme pour le respect des capacités des arcs, le respect des capacités de stockage est indexé sur le temps.

$$0 \leqslant f_\theta(i;j) \leqslant u_\theta(i;j) \qquad \forall \theta \in \mathcal{T}, \forall (i;j) \in E \tag{3.1}$$

$$x = \sum_{j \in V} \sum_{\theta \in \mathcal{T}} f_\theta(s;j) = \sum_{j \in V} \sum_{\substack{\theta \in \mathcal{T}, \\ \theta + \tau_\theta(j;t) \leqslant T}} f_\theta(j;t) \tag{3.2}$$

$$\sum_{k \in V} \sum_{\substack{\theta' \in \mathcal{T}, \\ \theta' + \tau_{\theta'}(k;i) = \theta}} f_{\theta'}(k;i) + y_{\theta-1}(i) = \sum_{j \in V} f_\theta(i;j) + y_\theta(i) \qquad \forall \theta \in \mathcal{T}, \forall i \in V \backslash \{s,t\} \tag{3.3}$$

$$y_\theta(i) \leqslant g_\theta(i) \qquad \forall \theta \in \mathcal{T}, \forall i \in V \tag{3.4}$$

La dimension temporelle apporte beaucoup de détails supplémentaires dans le modèle de graphe. On obtient donc un nombre important de modèles de graphes possibles pour les problèmes de flot. C'est ce qu'illustre la figure 3.1 où chaque combinaison est possible. Cette figure reprend le schéma général de la figure 2.2 donné dans le chapitre 2 page 9.

Figure 3.1 : Illustration des différents modèles de graphes dynamiques possibles pour les problèmes de flot.

On notera que si tous les paramètres sont constants au cours du temps (la présence des nœuds et arcs, les différents poids sur les nœuds et arcs), alors le graphe est statique, au sens où rien ne dépend du temps. Néanmoins, cela n'empêche pas d'étudier le flot au cours du temps, grâce notamment, à l'ajout d'un temps de traversée sur les arcs, qui donne une dimension temporelle au problème.

Historiquement, les premiers travaux dans ce sens sont ceux de Ford et Fulkerson (1958, 1962). Ils sont en effet les premiers à avoir travaillé sur les problèmes de flots au cours du temps dans des graphes qui n'évoluent pas, sous le terme de *flots dynamiques* (*dynamic flows* en anglais).

Cette idée est reprise dans les années 1970 par Wilkinson (1971), puis plus tard par Hoppe et Tardos (1994, 2000). Ces travaux étudient des modèles de graphes dynamiques où toutes les caractéristiques du graphe sont constantes et le flot est dépendant du temps grâce aux temps de traversée des arcs.

Le premier à introduire un modèle de graphe non constant pour les problèmes de flot est Minieka (1973, 1974). Dans (Minieka, 1973) le modèle de graphes présenté a des capacités d'arcs et des temps de traversée d'arcs qui dépendent du temps. Dans (Minieka, 1974) les capacités et temps de traversée sont constants mais des arcs peuvent être ajoutés ou au contraire supprimés du graphe au plus une fois.

Des modèles de graphes dynamiques pour les problèmes de flot dans lesquels les différents paramètres dépendent du temps sont des modèles très couramment utilisés, comme nous le verrons dans la suite de cette section. Mais certains travaux proposent d'autres modèles qui ont des particularités différentes.

Baumann et Köhler (2007) proposent un modèle dans lequel la capacité des arcs est constante dans le temps, mais où le temps de traversée des arcs varie. La spécificité de ce modèle repose sur le fait que, pour un arc donné, le temps de traversée ne dépend pas du temps, mais de la quantité de flot qui passe sur l'arc en question. L'idée étant que plus il y a de flot sur un arc, plus le temps de traversée sera long.

On peut aussi citer le modèle proposé par Akrida et al. (2019). Dans ce modèle, les capacités des arcs et les capacités de stockage sur les nœuds sont constants. En revanche, chaque arc dispose aussi d'une liste de pas de temps auxquels l'arc en question est présent, et il est absent à tous les autres pas de temps. En d'autres termes, un arc $(i; j)$ a une capacité qui vaut soit $u(i; j)$, soit 0.

Les travaux que l'on a évoqué ci-dessus traitaient des graphes (dynamiques ou non) au cours du temps, et le temps était une variable discrète. Mais il existe aussi dans la littérature des travaux traitant des graphes au cours du temps, en temps continu.

Les travaux de Anderson et al. (1982), ou encore ceux de Orda et Rom (1995) ou de Fleischer et Tardos (1998), ou plus récemment ceux de Hashemi et Nasrabadi (2012), étudient des graphes dynamiques en temps continu.

On notera cependant que dans ces travaux, aucune résolution de problème de flot tenant réellement compte du temps continu n'est proposée. En effet, Anderson et al. et Hashemi et Nasrabadi résolvent les problèmes en ayant au préalable discrétisé le temps. Ils se ramènent donc au cas discret. Quant à Orda et Rom et Fleischer et Tardos, ils ne proposent pas d'algorithme de résolution, mais s'intéressent plutôt à la preuve de propriétés à propos des problèmes étudiés. Orda et Rom prouvent que le théorème de flot-max/coupe-min est généralisable au modèle de graphe dynamique qu'ils proposent. Fleischer et Tardos prouvent, grâce à une transformation d'un flot discret en un flot continu, que des algorithmes de résolution conçus en temps discret peuvent être utilisés en temps continu. Il est donc suffisant de se focaliser sur les modèles et algorithmes discrets.

3.1.2 Problèmes de flot dans les graphes dynamiques

Les flots dans les graphes statiques peuvent se résumer à deux problèmes, d'une part le flot maximum, d'autre part le flot de cout minimum. Ces problèmes sont pertinents à étudier aussi dans un graphe dynamique. La dimension dynamique du graphe permet de s'intéresser à des problèmes où le temps

entre en compte dans la fonction objectif à optimiser. On trouve donc des travaux dans la littérature traitant de ces deux problèmes.

Outre la quantité de flot envoyée et le cout imputé à cet envoi, la dimension temporelle du graphe incite à s'intéresser aussi au temps que prend cet envoi. C'est l'objet du problème appelé en anglais *quickest flow*, que l'on traduira en français par *flot le plus rapide*. Ce problème est également traité dans la littérature.

3.1.2.1 Flot maximum

L'objectif du flot maximum est de maximiser la quantité de flot, en nombre d'unités de flot, à faire passer dans le graphe de la source s au puits t avant l'horizon de temps T. En respectant les contraintes données de (3.1) à (3.4), on cherche à optimiser le critère suivant :

$$x = \max\left\{ \sum_{i \in V} \sum_{\substack{\theta \in \mathcal{T}, \\ \theta + \tau_\theta(i;t) \leqslant T}} f_\theta(i;t) \right\} \tag{3.5}$$

Dans le cas statique, on veut maximiser la quantité de flot qui arrive au puits. Dans le cas dynamique, nous devons prendre en compte le paramètre de temps. C'est la raison pour laquelle on trouve dans la fonction objectif (3.5) une double somme. On somme d'une part sur tous les arcs entrant dans la source, et d'autre part sur tous les pas de temps permettant une arrivée avant l'horizon de temps.

Ce problème a été traité par, notamment, Ford et Fulkerson (1956, 1958, 1962) ; Wilkinson (1971) ; Minieka (1973, 1974) ; Halpern (1979) ; Anderson et al. (1982) ; Orda et Rom (1995) ; Baumann et Köhler (2007) ; Akrida et al. (2019).

3.1.2.2 Flot de cout minimum

Dans le problème du flot de cout minimum dans un graphe statique, il s'agit de minimiser le cout total du flot qui traverse le graphe de la source s jusqu'au puits t, sachant que la traversée de chaque arc pour chaque unité de flot a un cout. La quantité de flot x à envoyer est une donnée de départ du problème. En respectant les contraintes données de (3.1) à (3.4), on cherche à optimiser le critère suivant :

$$\min\left\{ \sum_{(i;j) \in E} \sum_{\theta \in \mathcal{T}} f_\theta(i;j) \cdot c_\theta(i;j) + \sum_{i \in V} \sum_{\theta \in \mathcal{T}} y_\theta(i) \cdot q_\theta(i) \right\} \tag{3.6}$$

La première partie de la fonction objectif à minimiser correspond à la fonction objectif du problème de flot de cout minimum dans un graphe statique. Là encore, comme pour le problème de flot maximum, il faut prendre en compte le paramètre de temps lorsque l'on calcule le cout, d'où la double somme, sur les arcs du graphe d'une part et sur le temps de l'intervalle d'étude d'autre part.

La deuxième partie de la fonction objectif à minimiser est spécifique au contexte dynamique puisque cela correspond au stockage d'unités de flot sur les nœuds du graphe. On a vu que dans certains modèles de graphes dynamiques, le stockage sur les nœuds était autorisé, et qu'un cout pouvait lui être imputé. C'est donc ce que signifie cette double somme, on compte le cout de chaque unité de flot stockée sur chaque nœud à chaque pas de temps de l'intervalle de temps.

Ce problème a été traité par, en particulier, Orlin (1984) ; Cai et al. (2001) ; Fleischer et Skutella (2003) ; Klinz et Woeginger (2004) ; Miller-Hooks et Patterson (2004) ; Nasrabadi et Hashemi (2010) ; Parpalea et Ciurea (2011) ; Hashemi et Nasrabadi (2012) ; Rostami et Ebrahimnejad (2014) ; Grande et al. (2018).

3.1.2.3 Flot le plus rapide et flot au plus tôt

Le problème du flot le plus rapide n'existe pas dans les graphe statiques. Puisque ce problème s'intéresse au temps nécessaire à l'envoi d'un certain nombre d'unités de flot dans un graphe dynamique de sa source s à son puits t, c'est un problème spécifique au contexte dynamique. Comme pour le problème du flot de cout minimum, la quantité de flot à envoyer x est une donnée de départ du problème. En respectant les contraintes données de (3.1) à (3.4), on cherche à optimiser le critère suivant :

$$\min\left\{\max_{\substack{i\in V,\ \theta\in\mathcal{T}\\ f_\theta(i;t)\neq 0}}\{\theta+\tau_\theta(i;t)\}\right\} \tag{3.7}$$

Il s'agit ici d'envoyer une quantité de flot donnée le plus rapidement possible. On cherche donc à minimiser l'arrivée au puits la plus tardive. On regarde, pour tous les arcs sur lequel passe du flot, la date à laquelle ce flot arrive. Et on minimise la date maximale.

Comme pour le flot le plus rapide, le flot au plus tôt n'existe pas dans les graphes statiques car il s'intéresse aussi à la date d'arrivée du flot.

$$\forall\theta\in\mathcal{T}, \max\left\{\sum_{i\in V}\sum_{\substack{\theta'\in\mathcal{T},\\ \theta'+\tau_{\theta'}(i;t)=\theta}} f_{\theta'}(i;t)\right\} \tag{3.8}$$

L'objectif de ce problème est de maximiser la quantité de flot qui arrive au puits à chaque pas de temps de l'intervalle d'étude. On veut qu'à chaque date de l'intervalle d'étude, le flot arrivant en t soit maximum.

Ces problèmes ont été traité, entre autres, par Hoppe et Tardos (1994, 2000) ; Lin et Jaillet (2015).

3.1.3 Méthodes de résolution

Dans la littérature, tous les travaux sur les problèmes de flot dans les graphes dynamiques ne proposent pas une méthode de résolution pour le problème étudié. En effet, certain travaux ont une démarche plutôt théorique qui s'intéresse à démontrer des propriétés sur le problème, ou l'existence d'une solution.

Les problèmes de flot maximum étudiés en ce sens proposent une généralisation du théorème de flot-max/coupe-min en définissant au préalable la notion de coupe minimum dans un graphe dynamique. C'est par exemple le cas du travail de Orda et Rom (1995), ou plus récemment celui de Akrida et al. (2019).

Skutella (2009) donne d'importantes définitions et rappelle les principaux résultats connus de la recherche sur les flots dans les graphes dynamiques. Il prouve notamment que le problème de flot de cout minimum dans un graphe dynamique est NP-difficile.

On peut cependant nuancer ce résultat. En effet, lorsque le nombre de pas de temps T est un paramètre et que les capacités, temps de traversée et couts changent, soit à un nombre fini de pas de temps, soit ne changent jamais, alors des algorithmes polynomiaux en T ne sont pas des algorithmes polynomiaux puisque T est une donnée en entrée. Il s'agit donc d'algorithmes pseudo-polynomiaux. En revanche, si les paramètres changent à chaque pas de temps, alors la taille des données est de l'ordre de T et non pas de l'ordre de $\log(T)$. Donc un algorithme polynomial en T est bien un algorithme polynomial. Nous nous plaçons dans ce cadre.

Historiquement, les graphes développés, décrits au chapitre 2 en section 2.3.3, ont été introduits par Ford et Fulkerson (1958, 1962) qui s'intéressaient au problème de flot dynamique maximum. Ils ont été en mesure de prouver que, pour un modèle de graphe où tous les paramètres sont constants, la solution optimale consiste à répéter le même flot successivement, c'est ce qu'ils nomment en anglais les *temporally repeated flows*.

L'utilisation du graphe développé est une méthode très largement utilisée pour résoudre les problèmes de flot dans un graphe dynamique. En effet, le graphe développé est un graphe statique contenant toutes les informations, sans aucune perte, du graphe dynamique auquel il correspond. Et il est possible de ramener l'objectif à optimiser dans le graphe dynamique à un objectif à optimiser dans le graphe développé.

C'est la raison pour laquelle beaucoup de travaux dans la littérature traitant des problèmes de flot dans les graphes dynamiques ne proposent pas d'algorithme spécifique aux graphes dynamiques pour résoudre le problème auquel ils s'intéressent.

L'inconvénient de cette méthode est la taille du graphe à traiter pour résoudre le problème. En effet, le graphe développé comporte T fois plus de nœuds et d'arcs que le graphe d'origine. Il semble donc pertinent de proposer de nouveaux algorithmes, spécifiques aux graphes dynamiques, pour résoudre les problèmes de flot.

3.1.3.1 Utilisation du graphe développé

Nous avons vu au chapitre précédent la notion de graphe développé associé à un graphe dynamique. Nous nous intéressons à présent à un flot sur un graphe développé. Nous montrons qu'un flot sur le graphe développé est équivalent à un flot sur son graphe dynamique associé. Ce résultat est implicite dans les papiers de la littérature. Le montrer permet de bien comprendre la démarche de résolution.

Dans les lemmes suivants, nous comparons le graphe dynamique et le graphe développé tel que nous l'avons décrit en section 2.3.3 dans le chapitre 2. Nous reprenons les notations proposées page 26 dans la section 3.1.1 pour décrire toutes les données du graphe dynamique. Dans le graphe développé, nous utilisons les notations suivantes :

V_D	:	ensemble des nœuds, soit tous les nœuds du graphe dynamique dupliqués T fois
$\tilde{s}$	:	nœud source
$\tilde{t}$	:	nœud puits
$\tilde{x}$	:	valeur totale du flot
$\tilde{f}(i;j)$	:	flot sur l'arc $(i;j)$
$\tilde{u}(i;j)$	:	capacité de l'arc $(i;j)$
$\tilde{c}(i;j)$	:	cout de traversée de l'arc $(i;j)$

La source et le puits du graphe dynamique ont été dupliqués T fois. Pour avoir un problème de flot simple à résoudre qui contient une unique source et un unique puits, on peut considérer que le premier exemplaire de la source du graphe dynamique, le nœud s_1, est la source du graphe développé. Les unités de flot partant de la source plus tard utiliseront les arcs de la forme $(s_\theta; s_{\theta+1})$. De la même manière, on peut considérer que le dernier exemplaire du puits du graphe dynamique, le nœud t_T, est le puits du graphe développé et les unités de flot arrivant plus tôt que l'horizon de temps emprunteront les arcs de la forme $(t_\theta; t_{\theta+1})$.

Tous les nœuds étant dupliqués, le stockage est représenté par des arcs de la forme $(i_\theta; i_{\theta+1})$ donc on ne fait plus la différence entre la capacité et la capacité de stockage, ni entre le cout de traversée et le cout de stockage. La valeur $\tilde{u}(i;j)$ est la capacité de l'arc $(i;j)$, et si $(i;j)$ est de la forme $(i_\theta; i_{\theta+1})$ alors c'est la capacité de stockage du nœud i à la date θ. De la même manière, la valeur $\tilde{c}(i;j)$ est le cout de l'arc $(i;j)$, et si $(i;j)$ est de la forme $(i_\theta; i_{\theta+1})$ alors c'est la cout de stockage sur le nœud i à la date θ.

Lemme 1 (Équivalence de la valeur du flot entre un graphe développé et un graphe dynamique)**.** *Il y a équivalence entre la valeur d'un flot sur un graphe développé et la valeur du même flot sur un graphe dynamique.*

Démonstration. La valeur du flot dans le graphe développé $\tilde{x}$ est égale à la quantité de flot qui arrive au nœud puits soit :

$$\tilde{x} = \sum_{I \in V_D} \tilde{f}(I; \tilde{t})$$

Or, chaque nœud du graphe développé correspond à un nœud du graphe dynamique à un pas de temps de l'intervalle d'étude, donc on a :

$$\tilde{x} = \sum_{i \in V} \sum_{\theta \in \mathcal{T}} \tilde{f}(i_\theta; t_T)$$

On peut distinguer deux types d'arcs qui arrivent au puits t_T : ceux qui ont pour source un autre nœud de V et celui qui arrive du nœud représentant t au pas de temps précédent. On a alors :

$$\tilde{x} = \sum_{\substack{i \in V, \\ i \neq t}} \sum_{\theta \in \mathcal{T}} \tilde{f}(i_\theta; t_T) + \tilde{f}(t_{T-1}; t_T)$$

On retrouve les éléments du graphe dynamique :

$$\tilde{x} = \sum_{\substack{i \in V, \\ i \neq t}} \sum_{\substack{\theta \in \mathcal{T}, \\ \theta + \tau_\theta(i;t)}} f_\theta(i; t) + y_{T-1}(t)$$

Il s'agit de la valeur du flot dans le graphe dynamique, donc :

$$\tilde{x} = x$$

□

Lemme 2 (Équivalence du cout du flot entre un graphe développé et un graphe dynamique)**.** *Il y a équivalence entre le cout d'un flot sur un graphe développé et le cout du même flot sur un graphe dynamique.*

Démonstration. On procède de la même manière que pour le lemme 1. Notons $\tilde{C}$ le cout du flot dans le graphe développé. On a :

$$\tilde{C} = \sum_{I \in V_D} \sum_{J \in V_D} \tilde{f}(I; J) \cdot \tilde{c}(I; J)$$

Comme précédemment, chaque nœud du graphe développé correspond à un nœud du graphe dynamique à un pas de temps de l'intervalle d'étude, donc :

$$\tilde{C} = \sum_{i \in V} \sum_{\theta \in \mathcal{T}} \sum_{j \in V} \sum_{\theta' \in \mathcal{T}} \tilde{f}(i_\theta; j_{\theta'}) \cdot \tilde{c}(i_\theta; j_{\theta'})$$

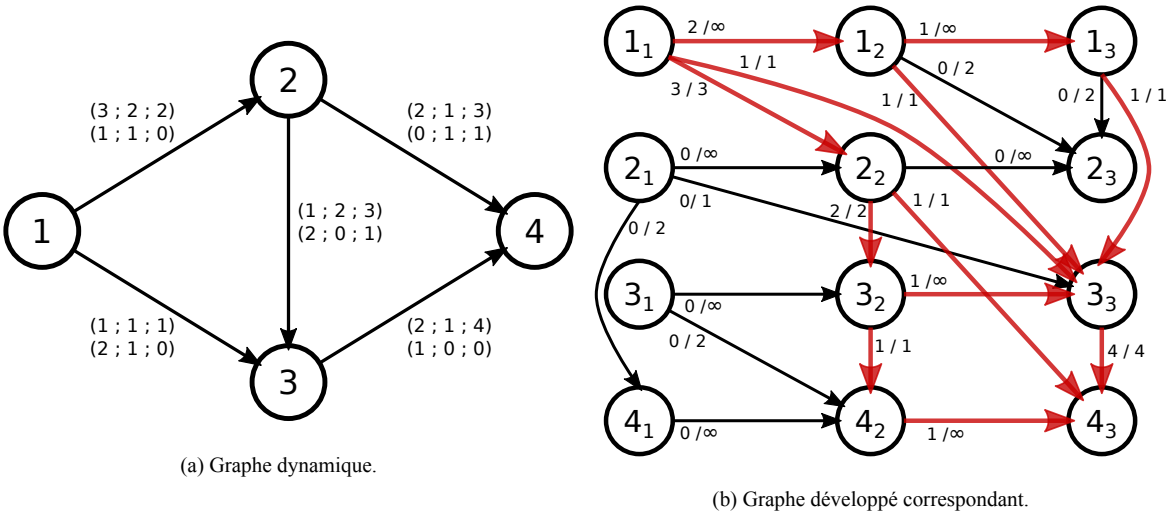
En distinguant les cas $i = j$ et $i \neq j$, on retrouve les éléments du graphe dynamique, d'où :

$$\tilde{C} = \sum_{i \in V} \sum_{\substack{j \in V, \\ i \neq j}} \sum_{\substack{\theta \in \mathcal{T}, \\ \theta + \tau_\theta(i;j) \leq T}} f_\theta(i; j) \cdot c_\theta(i; j) + \sum_{i \in V} \sum_{\substack{\theta \in \mathcal{T}, \\ \theta + \tau_\theta(i;j) < T}} y_\theta(i) \cdot q_\theta(i)$$

Il s'agit exactement du cout du flot sur le graphe dynamique. □

Théorème 3 (Équivalence entre un problème de flot sur un graphe développé et un graphe dynamique)**.** *Il y a équivalence entre un flot sur un graphe développé et le même flot sur un graphe dynamique.*

Démonstration. Puisque la valeur et le cout du flot sont égaux entre le graphe dynamique et le graphe développé (lemmes 1 et 2), il y a bien équivalence entre les deux. □



(a) Graphe dynamique.

(b) Graphe développé correspondant.

Figure 3.2 : Exemple de graphe dynamique (sous-jacent) et du graphe développé associé. Sur chaque arc du graphe dynamique, le premier vecteur $(u_1(i;j),\dots,u_\theta(i;j),\dots,u_4(i;j))$ correspond aux capacités de l'arc à chaque pas de temps, le second vecteur $(\tau_1(i;j),\dots,\tau_\theta(i;j),\dots,\tau_4(i;j))$ correspond aux temps de traversée de l'arc à chaque pas de temps. Sur le graphe développé, le couple $\tilde{f}(i;j)/\tilde{u}(i;j)$ sur chaque arc $(i;j)$ correspondant à la valeur de flot sur la capacité. Les arcs où passe du flot sont indiqués en rouge.

Le flot obtenu dans le graphe développé a donc la même valeur et le même cout que le flot dans le graphe dynamique. Un flot dans le graphe développé, qui est statique, permet d'identifier les pas de temps où envoyer du flot, ainsi que les nœuds sur lesquels il faut stocker du flot, comme le montre l'exemple de la figure 3.2.

Cette figure présente un graphe dynamique de 4 nœuds sur 3 pas de temps. On considère un stockage infini autorisé sur les nœuds à chaque pas de temps. On envoie 6 unités de flot du nœud 1 vers le nœud 4, qui doivent arriver au nœud 4 au plus tard à la date 3. Dans le graphe dynamique, la source est 1_1 et le puits est 4_3. Il est aisé d'observer sur le graphe dynamique comment sont envoyées les différentes unités. Quatre unités sont envoyées au premier pas de temps, 1 au deuxième pas de temps et 1 au troisième pas de temps. Ces deux dernières sont « stockées » sur le nœud 1 et passent sur les arcs $(1_1;1_2)$ et $(1_2;1_3)$. Une unité de flot arrive au puits au deuxième pas de temps et est stockée sur le nœud 4 en passant sur l'arc $(4_2;4_3)$, et 5 unités arrivent au puits au troisième pas de temps. Un unité de flot est aussi stockée sur le nœud 3 de la date 2 à la date 3 et c'est représenté par l'unité traversant l'arc $(3_2;3_3)$.

Utiliser le graphe développé assure donc une résolution optimale du problème. Certains travaux de la littérature se contentent de cette méthode car ils n'avaient pas pour objectif de trouver une méthode plus efficace. C'est le cas, parmi les travaux les plus anciens, de celui de Minieka (1973) qui traite le problème du flot maximum.

Plus récemment, on trouve les travaux de Parpalea et Ciurea (2011) qui traitent du problème de flot maximum de cout minimum dans un graphe dynamique, sans stockage, avec des capacités et des temps de traversée variables. L'algorithme qu'ils utilisent est en fait précédemment décrit par Ahuja et al. (1993) et fonctionne sur le graphe développé.

Dans un modèle en temps continu, Hashemi et Nasrabadi (2012) travaillent sur le flot de cout minimum et passent par le graphe développé pour apporter une approximation de la solution optimale. Ils décrivent deux types de flot en temps discret dans des graphes développés correspondant respectivement à une borne supérieure et à une borne inférieure du flot de cout minimum initial. Ils obtiennent

une solution approximative de leur problème en résolvant ces deux problèmes de flot en temps discret. Plus fine est la discrétisation du temps, meilleure est l'approximation.

D'autres travaux proposent des méthodes s'inspirant du graphe développé mais en proposent une version réduite, dont le traitement sera moins couteux que celui du graphe développé classique.

Fleischer et Skutella (2003) travaillent sur un modèle où les paramètres du graphe sont constants. Les auteurs définissent un graphe développé « condensé » dans lequel il n'est pas nécessaire de dupliquer tous les nœuds sous certaines conditions sur les temps de traversée ou bien en arrondissant les temps de traversée. Cela leur permet de proposer un algorithme d'approximation pour traiter le problème du flot de cout minimum.

Akrida et al. (2019) travaillent sur un modèle où les arcs peuvent apparaitre et disparaitre et les capacités des arcs et de stockage sont constantes. Ils définissent un graphe développé simplifié, qu'ils nomment *simplified time-expanded graphe (STEG)*, dont la taille est linéaire en la taille du graphe dynamique d'entrée.

3.1.3.2 Algorithmes spécifiques

Comme nous l'avons dit, l'inconvénient de la méthode du graphe développé est la taille du graphe à traiter pour résoudre le problème. On trouve donc, dans la littérature, des travaux dont l'objectif était de développer une méthode de résolution plus efficace. Ces travaux proposent donc de nouveaux algorithmes.

Le premier a avoir proposé une nouvelle méthode de résolution n'utilisant pas le graphe développé est Halpern (1979), qui travaille sur des graphes dont les paramètres sont non constants. Il s'agit d'une méthode inspirée du travail de Ford et Fulkerson (1956). Cette méthode a un inconvénient majeur, que l'auteur reconnait lui-même : si le graphe est très dynamique, autrement dit s'il est fréquemment modifié au cours du temps, alors la méthode proposée peut être, en pratique, moins efficace que la méthode résolvant le problème avec le graphe développé.

Certains problèmes de flot dans les graphes dynamiques peuvent s'avérer compliqués en fonction du modèle utilisé. C'est la raison pour laquelle certains travaux ne proposent pas de résoudre le problème de façon optimale.

Hoppe et Tardos (1994) s'intéressent au problème de flot au plus tôt dans un modèle de graphes avec des paramètres constants. Ils proposent un algorithme d'approximation polynomial pour résoudre ce problème.

Baumann et Köhler (2007) travaillent sur un modèle avec temps de traversée des arcs dépendant du flot. Ils traitent le problème de flot au plus tôt et proposent des résultats d'approximation.

Rostami et Ebrahimnejad (2014) travaillent sur un modèle avec des paramètres constants et proposent un algorithme d'approximation pour le problème de flot de cout minimum.

Beaucoup de travaux proposent une résolution optimale. Orlin (1984) étudie le problème de flot de cout minimum au cours du temps avec un horizon de temps infini. Il propose un modèle théorique dans lequel aucune disponibilité ou demande sur les nœuds, ni source, ni puits, ne sont définis. Les temps de traversée sont constants et les couts des arcs sont définis comme des fonctions convexes. Orlin décrit un algorithme polynomial pour résoudre le problème sur ce modèle infini.

Klinz et Woeginger (2004) s'intéressent au problème du flot de cout minimum dans un modèle de graphe avec des paramètres constants et un stockage infini autorisé sur les nœuds. Ils proposent une première variante du problème où le nombre d'unités de flot à envoyer dans le graphe est fixé à la valeur maximum de flot pouvant être envoyé dans le graphe avant l'horizon de temps T. Ils proposent

une seconde variante du problème où, cette fois-ci, l'horizon de temps est fixé au temps minimum pour l'envoi de la quantité de flot prédéfinie. Ils proposent une classe de graphes pour lesquels un algorithme glouton résout le problème de flot de cout minimum de façon optimale.

Lin et Jaillet (2015) étudient le problème du flot le plus rapide dans un graphe dont les paramètres sont constants. Ils proposent une méthode pour résoudre ce problème en se ramenant à un flot de cout minimum.

Grande et al. (2018) travaillent sur un graphe ayant plusieurs sources et puits, dont les paramètres sont constants et où le stockage n'est pas autorisé sur les nœuds. Ils proposent une méthode utilisant la génération de colonnes pour résoudre le problème de flot de cout minimum.

Dans la littérature, d'autres travaux encore proposent des algorithmes conçus spécifiquement pour le problème étudié dans des conditions de paramètres variables.

Cai et al. (2001) proposent un algorithme basé sur l'algorithme *Successive Shortest Path* (SSP) (Ahuja et al., 1993) pour résoudre le problème du flot de cout minimum. Ils décrivent un algorithme de plus courts chemins dynamiques de complexité $O(n \cdot m \cdot T^2)$. On peut tout de même noter qu'une telle complexité est aussi atteinte en utilisant l'algorithme de Bellman-Ford sur le graphe développé.

Miller-Hooks et Patterson (2004) travaillent sur un modèle de graphe avec de multiples sources et de multiples puits dans lequel un stockage infini est autorisé sur les nœuds. Les disponibilités et les demandes sur les nœuds sont définies à chaque pas de temps. L'objectif de Miller-Hooks et Patterson est de résoudre le problème du flot le plus rapide. Pour cela, elles définissent un problème de flot de cout minimum dans lequel les couts pris en compte sur les arcs sont les temps de traversée, et une solution optimale de ce problème donne une solution optimale du problème initial. Elles proposent un algorithme de complexité $O(U \cdot n^2 \cdot T^2)$ où U est le nombre d'unités de flot à envoyer dans le graphe. On peut noter qu'en utilisant directement, sur le graphe développé, l'algorithme SSP, avec l'algorithme de Dijkstra à chaque itération, on obtiendrait une complexité de $O(U \cdot T \cdot (m + n \cdot \log(n \cdot T)))$, qui est meilleur selon T et selon n.

Nasrabadi et Hashemi (2010) travaillent sur un modèle de graphes très général dans lequel tous les paramètres dépendent du temps et où il y a plusieurs sources et plusieurs puits qui ont des disponibilités et des demandes qui varient dans le temps. Ils s'intéressent au problème de flot de cout minimum. Dans ce contexte, il est tout à fait possible d'utiliser le graphe développé, et ils annoncent que la résolution du problème avec l'algorithme SSP aurait, dans ce cas, une complexité $O(B \cdot n^2 \cdot T^2)$ où B est une borne supérieure sur la disponibilité totale. Ils proposent un algorithme basé sur l'algorithme SSP, sur un graphe résiduel compact, de complexité $O(B \cdot n \cdot T \cdot (n + T))$, ce qui est bien meilleur.

3.2 Flot maximum

Nous présentons dans cette section les recherches que nous avons menées sur le problème du flot maximum dans un graphe dynamique, en collaboration avec Frédéric Guinand (LITIS, équipe RI2C). Ces travaux n'ont pas abouti à la création d'un algorithme de résolution. Néanmoins, nous jugeons la réflexion qui a entouré ce travail digne d'intérêt. C'est pourquoi nous le présentons dans cette section.

Nous présentons tout d'abord le contexte de cette étude, avec le modèle de graphe dynamique étudié, le problème traité, la méthode classique de résolution et les possibles applications. Nous présentons ensuite les pistes de recherche que nous avons explorées.

3.2.1 Contexte

3.2.1.1 Modèle

Comme présenté dans le chapitre 2, nous étudions un graphe dynamique que nous connaissons dans son ensemble. Nous connaissons à l'avance les modifications qui auront lieu de façon déterministe (voir

figure 2.3). Le graphe est étudié au cours du temps sur un intervalle d'étude fini et discret.

Afin de simplifier le modèle, mais pour autant sans perdre en généralité, nous considérons que les nœuds du graphe ainsi que les arcs sont présents tout au long de l'intervalle d'étude. Un arc ayant une capacité nulle à un pas de temps θ donné est un arc qu'aucune unité de flot ne peut traverser au temps θ, ce qui est équivalent à l'absence de l'arc. Par extension, si tous les arcs adjacents à un nœud donné ont une capacité nulle à un temps θ donné, alors ce nœud est inaccessible au temps θ, et cela simule l'absence du nœud en question.

La dynamique du graphe agit sur les capacités des arcs. Chaque arc a une capacité entière, possiblement nulle, qui varie dans le temps. Nous considérons que les arcs n'ont pas de temps de traversée. C'est à dire qu'une unité de flot mettra 0 unité de temps pour traverser un arc, et par extension aussi 0 unités de temps pour traverser tout un chemin dans le graphe. Nous autorisons sur les nœuds un stockage infini. C'est à dire que chaque nœud pourra conserver autant d'unités de flot que nécessaire à chaque pas de temps.

Puisque nous nous intéressons au problème du flot maximum, nous ne considérons pas de couts. Dans le modèle général, il s'agit d'une valeur 0, constante dans le temps, sur tous les couts de traversée et les couts de stockage.

En conservant les notations utilisées figure 3.1, le modèle étudié ici est présenté dans la figure 3.3.

3.2.1.2 Formulation du problème

On s'intéresse ici au problème du flot maximum dans un graphe dynamique dont le modèle vient d'être décrit. Le principe de ce problème est d'envoyer dans un graphe le plus d'unités de flot possible depuis un nœud source s vers un nœud puits t avant l'horizon de temps T de l'intervalle d'étude du graphe.

La fonction objectif à minimiser est la suivante, obtenues à partir des contraintes (3.1) à (3.4) :

$$\max \sum_{j \in V} \sum_{\substack{\theta \in \mathcal{T}, \\ \theta + \tau_\theta(j;t) \leqslant T}} f_\theta(j;t) \tag{3.9}$$

en respectant les contraintes suivantes :

$$0 \leqslant f_\theta(i;j) \leqslant u_\theta(i;j) \qquad \forall \theta \in \mathcal{T}, \forall (i;j) \in E \tag{3.10}$$

$$\sum_{j \in V} \sum_{\theta \in \mathcal{T}} f_\theta(s;j) = \sum_{j \in V} \sum_{\substack{\theta \in \mathcal{T}, \\ \theta + \tau_\theta(j;t) \leqslant T}} f_\theta(j;t) \tag{3.11}$$

$$\sum_{j \in V} \sum_{\substack{\theta' \in \mathcal{T}, \\ \theta' \leqslant \theta}} f_{\theta'}(j;i) \geqslant \sum_{j \in V} \sum_{\substack{\theta' \in \mathcal{T}, \\ \theta' \leqslant \theta}} f_\theta(i;j) \qquad \forall \theta \in \mathcal{T}, \forall i \in V \backslash \{s,t\} \tag{3.12}$$

L'équation (3.10) représente la contrainte de respect des capacités des arcs. L'équation (3.11) assure que toute la quantité de flot à quitter la source s pendant l'intervalle d'étude arrive au puits t avant l'horizon de temps. Enfin, la contrainte représentée par l'équation (3.12) garantit la conservation du flot tout en autorisant un stockage infini sur les nœuds. En effet, puisque le stockage infini est autorisé sur les nœuds, parmi la quantité de flot qui est arrivé sur un nœud i jusqu'à un temps θ, une partie a pu être stockée sur i et donc ne pas être encore repartie. C'est alors la contrainte (3.11) qui permet de s'assurer qu'aucune unité de flot n'est restée sur un nœud intermédiaire une fois l'horizon de temps atteint. De cette manière, il n'est pas nécessaire de modéliser par des variables spécifiques le stockage y_θ sur les nœuds.

La figure 3.4 montre un graphe dynamique de 4 nœuds sur 3 pas de temps. Les capacités sont indiquées sur chaque arc. Le nœud 1 est la source, le nœud 4 est le puits. Chaque nœud peut stocker une quantité infinie d'unités de flot.

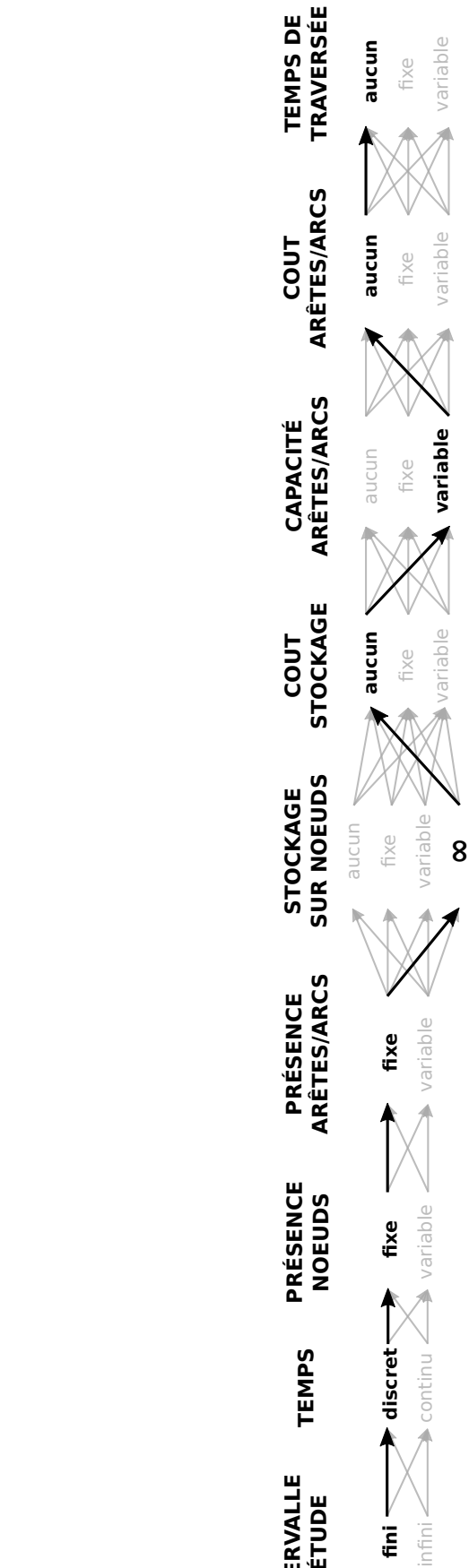
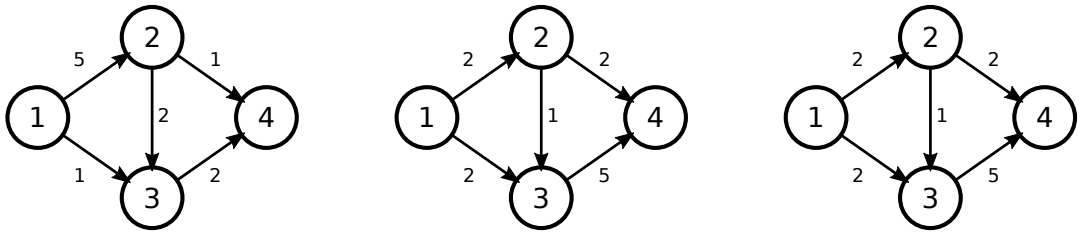


Figure 3.3 : Illustration du modèle de graphe dynamique étudié pour le problème de flot maximum.



(a) G_1 (b) G_2 (c) G_3

Figure 3.4 : Graphe dynamique sur 3 pas de temps. Chaque t-graphe est représenté. Sur chaque arc $(i; j)$, la valeur $u(i\,j)$ représente la capacité de l'arc $(i; j)$.

Si le stockage n'était pas autorisé ici, on pourrait envoyer 12 unités de flot au maximum, 3 au premier pas de temps, 4 au deuxième pas de temps et 5 au dernier pas de temps. On trouve aisément cette possibilité en résolvant un flot maximum sur chacun des t-graphes.

Le stockage permet d'envoyer plus de flot, et rend le problème algorithmiquement intéressant. En effet, puisque le stockage est autorisé, une unité de flot partant de la source s à la date θ n'arrivera pas au puits t à la date θ. Il est donc impossible de traiter les t-graphes indépendamment les uns des autres. On doit considérer tout l'intervalle d'étude pour résoudre le problème.

Notons cependant que la résolution du flot maximum sur un modèle sans stockage, donc en résolvant T flots maximum indépendants, donne une borne inférieure évidente au problème que l'on considère ici.

3.2.1.3 Résolution par graphe développé

Ce problème peut aisément se résoudre grâce au graphe développé. Le graphe développé correspondant à notre modèle de graphe dynamique possède tous les nœuds du graphe dynamique T fois, et toutes ses arêtes T fois. Dans le graphe développé, le nœud i_θ correspond au nœud i du graphe dynamique au pas de temps θ. Pour chaque arête $i\,j$ du graphe dynamique au pas de temps θ, le graphe développé possède une arête $(i_\theta; j_\theta)$ de capacité $u_\theta(i; j)$. Puisque les nœuds du graphe peuvent stocker du flot à l'infini, le graphe dynamique possède aussi des arêtes $(i_\theta; i_{\theta+1})$ de capacité infinie.

Il n'est pas nécessaire ici d'ajouter des nœuds fictifs pour symboliser la source unique et le puits unique. Le source du graphe développé est définie comme étant la représentation de la source au premier pas de temps de l'intervalle d'étude, à savoir s_1. Le puits du graphe est défini comme étant la représentation du puits à l'horizon de temps, soit t_T.

Le flot partant de la source après le premier pas de temps sera considéré comme étant stocké sur la source, et donc traversera successivement des arcs $(s_\theta; s_{\theta+1})$ dans le graphe développé. Et de façon similaire, le flot arrivant au puits avant l'horizon de temps pourra rester stocké sur le puits jusqu'à l'horizon de temps. Ce qui se traduira dans le graphe développé par du flot traversant des arcs du type $(t_\theta; t_{\theta+1})$.

La figure 3.5 montre le graphe développé correspondant au graphe dynamique donné en exemple dans la figure 3.4.

Le graphe développé permet de résoudre facilement le problème puisque nous pouvons utiliser directement un algorithme de résolution de flot maximum. Sur cet exemple, on obtient un flot maximum de valeur 14. En utilisant le stockage, on peut envoyer dans ce graphe 2 unités de plus que si on ne l'utilisait pas.

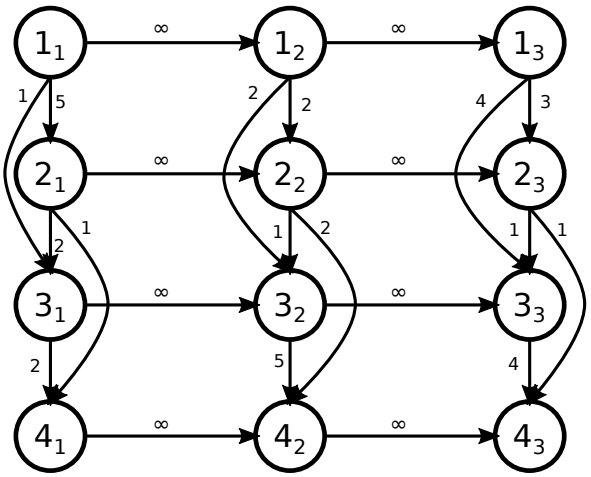


Figure 3.5 : Graphe développé correspondant au graphe dynamique de la figure 3.4. Le nœud 1_1 est la source et le nœud 4_3 est le puits.

Le graphe développé possède $n \cdot T$ nœuds, et de l'ordre de $T \cdot (m+n)$ arcs. Ce nombre de nœuds et d'arcs a un impact fort évident sur le calcul de la solution. Deux algorithmes sont très utilisés dans le cas statique pour résoudre le problème de flot maximum dans un graphe (voir (Ahuja et al., 1993)). Il s'agit de l'algorithme d'Edmonds-Karp, qui est une variante de l'algorithme de Ford-Fulkerson, et de l'algorithme Push-Relabel. Observons la complexité théorique de ces algorithmes utilisés sur le graphe développé.

L'algorithme de Ford-Fulkerson, dans sa variante d'Edmonds-Karp, résout le flot maximum dans un graphe de n nœuds et m arcs en temps $O(n \cdot m^2)$. Le même algorithme utilisé sur le graphe développé correspondant à un graphe dynamique de n nœuds, m arcs et T pas de temps, résout le flot maximum en temps $O(T^3 \cdot n \cdot (m+n)^2)$.

L'algorithme Push-Relabel, qui est un algorithme de type pré-flot, a une complexité en temps de $O(n^2 \cdot m)$. En utilisant cet algorithme dans un graphe développé possédant $n \cdot T$ nœuds et $T \cdot (m+n)$ arcs, la complexité théorique de résolution est $O(T^3 \cdot n^2 \cdot (m+n))$.

On peut constater que la taille importante du graphe développé augmente excessivement la complexité en temps de calcul pour résoudre le problème du flot maximum. La résolution par le graphe développé sera donc particulièrement inefficace dès lors que le graphe dynamique aura un nombre important de sommets ou d'arcs ou qu'il sera observé sur un long intervalle d'étude. Il devient donc naturel de vouloir utiliser d'autres méthodes pour résoudre le problème du flot maximum.

3.2.2 Méthode par capacités cumulées

L'idée de l'approche que nous avons explorée est d'exploiter la connaissance que nous avons du graphe. En effet, nous savons comment le graphe va évoluer dans le temps, nous souhaitons donc utiliser cette connaissance afin de déterminer le flot à envoyer. Nous supposons ici que le graphe est sans circuits.

3.2.2.1 Description

L'objectif est d'exploiter toute la connaissance que l'on a du graphe afin de déterminer combien d'unités de flot envoyer sur un arc. Pour chaque arc, nous souhaitons donc utiliser la connaissance du graphe en amont de l'arc en question, mais aussi la connaissance du graphe en aval, et ce grâce aux variations de capacités que nous connaissons. Afin de pouvoir utiliser ces informations, il est important

que, pour un arc donné, aucun autre arc ne puisse se trouver à la fois avant et après lui dans un chemin de la source au puits. C'est la raison pour laquelle le graphe doit être sans circuits.

La possibilité de stocker sur chaque nœud une infinité d'unités de flot nous invite à envoyer le plus de flot possible sur les arcs car les nœuds pourront évacuer ce flot plus tard si nécessaire. En utilisant tous ces éléments, et afin de connaitre la quantité maximum de flot qui pourra passer sur un arc, nous proposons de cumuler les capacités successives des arcs.

Le graphe étant sans cycles, on peut parcourir les arêtes en suivant un ordre topologique. Les nœuds sont ordonnés $i_1, \ldots, i_n$ de manière à ce que chaque arc du graphe ait pour extrémité initiale un nœud i_λ et pour extrémité finale un nœud $i_{\lambda'}$ tel que $\lambda < \lambda'$. On a ici la source du graphe $s = i_1$ et le puits du graphe $t = i_n$.

Dans la suite, nous pouvons ainsi parcourir les arcs de telle manière qu'un arc $(i; j)$ donné soit traité avant tout arc $(j; k)$.

Nous proposons, pour chaque arc $(i; j)$, un vecteur de capacités cumulées dans l'ordre chronologique $fcc(i; j) = (fcc_1(i; j), \ldots, fcc_\theta(i; j), \ldots, fcc_T(i; j))$ (fcc pour *forward cumulative capacities*). Dans ce vecteur, chaque valeur $fcc_\theta(i; j)$ correspond à la quantité maximale de flot qui peut passer sur l'arc $(i; j)$ de la date 1 à la date θ en tenant compte de la quantité maximale de flot qui peut arriver en i depuis la source du graphe.

Les arcs sont parcourus d'après l'ordre topologique défini, et les capacités cumulées sont calculées, de la date 1 à la date T, de la façon suivante :

$$fcc_\theta(ij) = \min\left\{ fcc_{\theta-1}(ij) + u_\theta(ij) \; ; \; \sum_{ki \in E} fcc_\theta(ki) \right\} \tag{3.13}$$

La quantité de flot qui passe sur un arc jusqu'à une date θ donnée ne peut excéder la quantité qui peut passer jusqu'à la date $\theta - 1$ plus la capacité de l'arc à la date θ. Elle ne peut pas non plus excéder la somme de ce qui passe sur tous les arcs prédécesseurs jusqu'à la date θ.

Lemme 4 (Le vecteur fcc est une borne supérieure du flot pouvant passer sur un arc)**.** *Le vecteur fcc donne une borne supérieure du flot pouvant passer sur un arc jusqu'à chaque date de l'intervalle d'étude.*

Démonstration. Prouvons ce résultat par double récurrence sur les pas de temps de l'intervalle d'étude et sur les nœuds du graphe. On sait que le graphe que l'on étudie est sans circuits, donc il existe un ordre topologique sur nœuds $i_1, \ldots, i_n$ tel que chaque arc du graphe est de la forme $(i_k; i_{k'})$ avec $k < k'$. Et donc, par extension, il est aussi possible d'ordonner les arcs de telle manière que tout arc de la forme $(i_k; i_\lambda)$ précède tout arc de la forme $(i_\lambda; i_{k'})$.

— Considérons le premier pas de temps de l'intervalle d'étude.
  — Considérons le premier arc de la liste, $(i_1; i_\lambda)$. On a :

$$\begin{aligned} fcc_1(i_1; i_\lambda) &= \min\left\{ fcc_0(i_1; i_\lambda) + u_1(i_1; i_\lambda) \; ; \; \sum_{(k; i_1) \in E} fcc_1(k; i_1) \right\} \\ &= \min\left\{ 0 + u_1(i_1; i_\lambda) \; ; \; 0 \right\} \\ &= u_1(i_1; i_\lambda) \end{aligned}$$

En effet, on considère que fcc_0 vaut toujours 0, et comme l'arc $(i_1; i_\lambda)$ n'a pas de prédécesseur, alors $\sum_{(k; i_1) \in E} fcc_1(k; i_1) = 0$.
Pour le premier arc de la liste, la valeur fcc_1 est bien une borne supérieure du flot pouvant passer sur cet arc jusqu'à la date 1 puisqu'il s'agit de la capacité de l'arc.

— Supposons maintenant que fcc_1 soit une borne supérieure du flot pouvant passer au premier pas de temps sur tous les arcs avant le nœud i_α dans l'ordre topologique. Considérons l'arc suivant $(i_\alpha; i_{\lambda'})$ dans la liste ordonnée. On a :

$$\begin{aligned} fcc_1(i_\alpha; i_{\lambda'}) &= \min\left\{ fcc_0(i_\alpha; i_{\lambda'}) + u_1(i_\alpha; i_{\lambda'}) \; ; \sum_{(k;i_\alpha)\in E} fcc_1(k; i_\alpha) \right\} \\ &= \min\left\{ u_1(i_\alpha; i_{\lambda'}) \; ; \sum_{(k;i_\alpha)\in E} fcc_1(k; i_\alpha) \right\} \end{aligned}$$

Le flot passant sur un arc au premier pas de temps ne peut pas excéder la capacité de l'arc, c'est l'une des contraintes du problème de flot. Et il ne peut pas non plus excéder la quantité de flot maximum qui arrive sur cet arc. On sait que $fcc_1(k; i_\alpha)$ est une borne supérieure du flot au premier pas de temps sur chaque arc $(k; i_\alpha) \in E$ par hypothèse de récurrence. Donc $fcc_1(i_\alpha; i_{\lambda'})$ est bien une borne supérieur du flot pouvant passer sur l'arc $(i_\alpha; i_{\lambda'})$ au premier pas de temps.

— La valeur fcc_1 est une borne supérieure du flot pouvant passer au premier pas de temps sur le premier arc, et on a montré récursivement que c'était aussi une borne supérieure du flot pouvant passer au premier pas de temps sur tous les arcs du graphe.

— Puisque l'on sait que fcc_1 est une borne supérieure, montrons que fcc_θ est aussi une borne supérieure.

— Supposons donc que $fcc_{\theta-1}(i_1; i_\lambda)$ est une borne supérieure du flot pouvant passer sur le premier arc de la liste $(i_1; i_\lambda)$ jusqu'à la date $\theta - 1$. Vérifions le pas de temps suivant. On a :

$$\begin{aligned} fcc_\theta(i_1; i_\lambda) &= \min\left\{ fcc_{\theta-1}(i_1; i_\lambda) + u_\theta(i_1; i_\lambda) \; ; \sum_{(k;i_1)\in E} fcc_\theta(k; i_1) \right\} \\ &= \min\left\{ fcc_{\theta-1}(i_1; i_\lambda) + u_\theta(i_1; i_\lambda) \; ; \; 0 \right\} \\ &= fcc_{\theta-1}(i_1; i_\lambda) + u_\theta(i_1; i_\lambda) \end{aligned}$$

Puisque $(i_1; i_\lambda)$ est la premier arc de la liste, alors $\sum_{(k;i_1)\in E} fcc_\theta(k; i_1) = 0$. On sait par hypothèse de récurrence que $fcc_{\theta-1}(i_1; i_\lambda)$ est une borne supérieure du flot pouvant passer sur l'arc $(i_1; i_\lambda)$ jusqu'à la date $\theta - 1$. Il est donc évident que le flot pouvant passer sur l'arc $(i_1; i_\lambda)$ jusqu'à la date θ ne peut pas excéder ce qui peut passer jusqu'à la date $\theta - 1$ plus ce qui peut passer sur l'arc $(i_1; i_\lambda)$ à la date θ, soit la capacité de l'arc. Donc la valeur $fcc_\theta(i_1; i_\lambda)$ est bien une borne supérieure du flot pouvant traverser l'arc $(i_1; i_\lambda)$ jusqu'à la date θ.

— Supposons maintenant que $fcc_{\theta-1}$ soit une borne supérieure du flot pouvant passer jusqu'à la date $\theta - 1$ sur tous les arcs avant le nœud i_α dans l'ordre topologique. Considérons l'arc suivant $(i_\alpha; i_{\lambda'})$ dans la liste ordonnée. On a :

$$fcc_\theta(i_\alpha; i_{\lambda'}) = \min\left\{ fcc_{\theta-1}(i_\alpha; i_{\lambda'}) + u_\theta(i_\alpha; i_{\lambda'}) \; ; \sum_{(k;i_\alpha)\in E} fcc_\theta(k; i_\alpha) \right\}$$

Par hypothèse de récurrence, on sait que $fcc_{\theta-1}(i_\alpha; i_{\lambda'})$ est une borne supérieure du flot pouvant traverser l'arc $(i_\alpha; i_{\lambda'})$ jusqu'à la date $\theta - 1$. Il est donc clair que le flot pouvant traverser l'arc $(i_\alpha; i_{\lambda'})$ jusqu'à la date θ ne peut excéder $fcc_{\theta-1}(i_\alpha; i_{\lambda'})$ plus ce qui peut traverser l'arc à la date θ. On sait aussi par hypothèse de récurrence que $fcc_\theta(k; i_\alpha)$ est une borne supérieure du flot pouvant passer jusqu'à la date θ sur chaque arc $(k; i_\alpha)$, donc le flot sur l'arc $(i_\alpha; i_{\lambda'})$ ne peut pas excéder la somme de tout ce qui peut arriver en i_α.

— La valeur fcc_θ est une borne supérieure du flot pouvant passer jusqu'à la date θ sur le premier arc, et on a montré récursivement que c'était aussi une borne supérieure du flot pouvant passer jusqu'à la date θ sur tous les arcs du graphe.

□

De la même manière, nous proposons, pour chaque arc $(i;j)$, un vecteur de capacités cumulées dans l'ordre antéchronologique $bcc(i;j) = (bcc_1(i;j),\dots,bcc_\theta(i;j),\dots,bcc_T(i;j))$ (bcc pour *backward cumulative capacities*). Dans ce vecteur, chaque valeur $bcc_\theta(i;j)$ correspond à la quantité maximale de flot qui peut passer sur l'arc $(i;j)$ de la date θ à la date T en tenant compte de la quantité maximale de flot qui peut être évacuée vers le puits du graphe depuis la nœud j.

Les arcs sont parcourus suivant l'ordre inverse de l'ordre topologique défini, et les capacités cumulées sont calculées, de la date T à la date 1, de la façon suivante :

$$bcc_\theta(i;j) = \min\left\{bcc_{\theta+1}(i;j) + u_\theta(i;j)\ ;\ \sum_{(j;k)\in E} bcc_\theta(j;k)\right\} \tag{3.14}$$

La quantité de flot qui peut passer sur un arc depuis la date θ jusqu'à l'horizon de temps T ne peut excéder la capacité de l'arc à la date θ plus la quantité de flot qui peut passer à partir de la date $\theta+1$ et jusqu'à l'horizon de temps. Elle ne peut pas non plus excéder la somme des quantités maximales de flot qui peuvent passer sur tous les arcs successeurs à partir de θ et jusqu'à l'horizon de temps.

Lemme 5 (Le vecteur bcc est une borne supérieure du flot pouvant passer sur un arc)**.** *Le vecteur bcc donne une borne supérieure du flot pouvant passer sur un arc à partir de chaque date de l'intervalle d'étude.*

Démonstration. La preuve est tout à fait symétrique à celle du lemme 4.

Prouvons ce résultat par double récurrence sur les pas de temps de l'intervalle d'étude et sur les nœuds du graphe. On sait que le graphe que l'on étudie est sans circuits, donc il existe un ordre topologique sur nœuds $i_1,\dots,i_n$ tel que chaque arc du graphe est de la forme $(i_k;i_{k'})$ avec $k<k'$. Et donc, par extension, il est aussi possible d'ordonner les arcs de telle manière que tout arc de la forme $(i_k;i_\lambda)$ précède tout arc de la forme $(i_\lambda;i_{k'})$.

— Considérons le dernier pas de temps de l'intervalle d'étude.
  — Considérons le dernier arc de la liste, $(i_\gamma;i_n)$. On a :

$$\begin{aligned} bcc_T(i_\gamma;i_n) &= \min\left\{bcc_{T+1}(i_\gamma;i_n) + u_T(i_\gamma;i_n)\ ;\ \sum_{(i_n;k)\in E} bcc_T(i_n;k)\right\} \\ &= \min\left\{0 + u_T(i_\gamma;i_n)\ ;\ 0\right\} \\ &= u_T(i_\gamma;i_n) \end{aligned}$$

  En effet, on considère que bcc_{T+1} vaut toujours 0, et comme l'arc $(i_\gamma;i_n)$ n'a pas de successeur, alors $\sum_{(i_n;k)\in E} bcc_T(i_n;k) = 0$.
  Pour le dernier arc de la liste, la valeur bcc_T est bien une borne supérieure du flot pouvant passer sur cet arc à partir de la date T puisqu'il s'agit de la capacité de l'arc.
— Supposons maintenant que bcc_T soit une borne supérieure du flot pouvant passer au dernier pas de temps sur tous les arcs après le nœud i_α dans l'ordre topologique. Considérons l'arc suivant $(i_{\gamma'};i_\alpha)$ dans la liste ordonnée. On a :

$$\begin{aligned} bcc_T(i_{\gamma'};i_\alpha) &= \min\left\{bcc_{T+1}(i_{\gamma'};i_\alpha) + u_T(i_{\gamma'};i_\alpha)\ ;\ \sum_{(i_\alpha;k)\in E} bcc_T(i_\alpha;k)\right\} \\ &= \min\left\{u_T(i_{\gamma'};i_\alpha)\ ;\ \sum_{(i_\alpha;k)\in E} bcc_T(i_\alpha;k)\right\} \end{aligned}$$

  Le flot passant sur un arc au dernier pas de temps ne peut pas excéder la capacité de l'arc, c'est l'une des contrainte du problème de flot. Et il ne peut pas non plus excéder la quantité de flot maximum qui peut partir de cet arc. On sait que $bcc_T(i_\alpha;k)$ est une borne supérieure

du flot au dernier pas de temps sur chaque arc $(i_\alpha; k) \in E$ par hypothèse de récurrence. Donc $bcc_T(i_{\gamma'}; i_\alpha)$ est bien une borne supérieur du flot pouvant passer sur l'arc $(i_{\gamma'}; i_\alpha)$ au dernier pas de temps.

— La valeur bcc_T est une borne supérieure du flot pouvant passer au dernier pas de temps sur le dernier arc, et on a montré récursivement que c'était aussi une borne supérieure du flot pouvant passer au dernier pas de temps sur tous les arcs du graphe.

— Puisque l'on sait que bcc_T est une borne supérieure, montrons que bcc_θ est aussi une borne supérieure.

— Supposons donc que $bcc_{\theta+1}(i_\gamma; i_n)$ est une borne supérieure du flot pouvant passer sur le dernier arc de la liste $(i_\gamma; i_n)$ à partir de la date $\theta + 1$. Vérifions le pas de temps précédent. On a :

$$\begin{aligned} bcc_\theta(i_\gamma; i_n) &= \min\left\{ bcc_{\theta+1}(i_\gamma; i_n) + u_\theta(i_\gamma; i_n) \; ; \sum_{(i_n;k)\in E} bcc_\theta(i_n; k) \right\} \\ &= \min\left\{ bcc_{\theta+1}(i_\gamma; i_n) + u_\theta(i_\gamma; i_n) \; ; \; 0 \right\} \\ &= bcc_{\theta+1}(i_\gamma; i_n) + u_\theta(i_\gamma; i_n) \end{aligned}$$

Puisque $(i_\gamma; i_n)$ est la dernier arc de la liste, alors $\sum_{(i_n;k)\in E} bcc_\theta(i_n; k) = 0$. On sait par hypothèse de récurrence que $bcc_{\theta+1}(i_\gamma; i_n)$ est une borne supérieure du flot pouvant passer sur l'arc $(i_\gamma; i_n)$ à partir de la date $\theta + 1$. Il est donc évident que le flot pouvant passer sur l'arc $(i_\gamma; i_n)$ à partir de la date θ ne peut pas excéder ce qui peut passer à partir de la date $\theta + 1$ plus ce qui peut passer sur l'arc $(i_\gamma; i_n)$ à la date θ, soit la capacité de l'arc. Donc la valeur $bcc_\theta(i_\gamma; i_n)$ est bien une borne supérieure du flot pouvant traverser l'arc $(i_\gamma; i_n)$ à partir de la date θ.

— Supposons maintenant que $bcc_{\theta+1}$ soit une borne supérieure du flot pouvant passer à partir de la date $\theta+1$ sur tous les arcs après le nœud i_α dans l'ordre topologique. Considérons l'arc suivant $(i_{\gamma'}; i_\alpha)$ dans la liste ordonnée. On a :

$$bcc_\theta(i_{\gamma'}; i_\alpha) = \min\left\{ bcc_{\theta+1}(i_{\gamma'}; i_\alpha) + u_\theta(i_{\gamma'}; i_\alpha) \; ; \sum_{(i_\alpha;k)\in E} bcc_\theta(i_\alpha; k) \right\}$$

Par hypothèse de récurrence, on sait que $bcc_{\theta+1}(i_{\gamma'}; i_\alpha)$ est une borne supérieure du flot pouvant traverser l'arc $(i_{\gamma'}; i_\alpha)$ à partir de la date $\theta + 1$. Il est donc clair que le flot pouvant traverser l'arc $(i_{\gamma'}; i_\alpha)$ à partir de la date θ ne peut excéder $bcc_{\theta+1}(i_{\gamma'}; i_\alpha)$ plus ce qui peut traverser l'arc à la date θ. On sait aussi par hypothèse de récurrence que $bcc_\theta(i_\alpha; k)$ est une borne supérieure du flot pouvant passer à partir de la date θ sur chaque arc $(i_\alpha; k)$, donc le flot sur l'arc $(i_{\gamma'}; i_\alpha)$ ne peut pas excéder la somme de tout ce qui peut partir de i_α.

— La valeur bcc_θ est une borne supérieure du flot pouvant passer à partir de la date θ sur le dernier arc, et on a montré récursivement que c'était aussi une borne supérieure du flot pouvant passer à partir de la date θ sur tous les arcs du graphe.

□

Ces deux vecteurs donnent pour chaque arc $(i; j)$ une information intéressante. Le vecteur fcc donne une borne supérieure sur la quantité de flot qui peut arriver en i depuis la source du graphe et passer sur $(i; j)$ jusqu'à la date θ. Le vecteur bcc donne une borne supérieure sur la quantité de flot qui peut passer sur $(i; j)$ et aller jusqu'au puits du graphe à partir de la date θ.

On peut déduire de ces vecteurs une quantité u_{max} pour chaque arc $(i; j)$ qui correspond à la quantité maximum de flot qui peut traverser $(i; j)$ sur tout l'intervalle d'étude du graphe en tenant compte de ce qui peut arriver depuis la source et de ce qui pourra arriver jusqu'au puits ensuite. On a donc :

$$u_{max}(i; j) = \min\left\{ bcc_1(i; j) \; ; \; fcc_T(i; j) \; ; \; fcc_\theta(i; j) + bcc_{\theta+1}(i; j), \theta \in \{1, \ldots, T-1\} \right\} \tag{3.15}$$

Théorème 6 (La valeur u_{max} est une borne supérieure du flot pouvant passer sur un arc)**.** *La valeur u_{max} est une borne supérieure du flot pouvant passer sur un arc pendant l'intervalle d'étude.*

Démonstration. On sait grâce aux lemmes 4 et 5 que les vecteurs fcc et bcc donnent des bornes supérieures de flot pouvant passer sur chaque arc du graphe, jusqu'à chaque date de l'intervalle et à partir de chaque date de l'intervalle d'étude respectivement.

Il est évident que le flot pouvant traverser un arc $(i; j)$ au total ne peut pas excéder une borne supérieure de la quantité de flot qui peut traverser l'arc à partir de la date 1, donnée par $bcc_1(i; j)$.

Il est aussi évident que le flot pouvant traverser un arc $(i; j)$ au total ne peut pas excéder une borne supérieure de la quantité de flot qui peut traverser l'arc jusqu'à la date T, donnée par $fcc_T(i; j)$.

Pour une date θ donnée, une borne supérieure de la quantité de flot pouvant traverser un arc $(i; j)$ jusqu'à θ plus une borne supérieure de la quantité de flot pouvant traverser $(i; j)$ à partir de θ est une borne supérieure de la quantité de flot pouvant traverser $(i; j)$ au total.

Donc la valeur u_{max}, qui cumule les deux vecteurs fcc et bcc donne, pour chaque arc, une borne du flot pouvant traverser cet arc pendant l'intervalle d'étude. □

Il est à présent intéressant d'utiliser cette quantité u_{max} comme capacité d'arc sur le graphe sous-jacent pour y calculer un flot maximum et tenter ensuite d'en déduire un flot sur le graphe dynamique. Le flot total sur chaque arc du graphe dynamique serait égal au flot sur l'arc correspondant dans le graphe sous-jacent.

Théorème 7 (La méthode des capacités cumulées donne une borne supérieure du flot maximum)**.** *Le flot maximum calculé sur le graphe sous-jacent statique où les capacités des arcs valent u_{max} est une borne supérieure du flot maximum dans le graphe dynamique.*

Démonstration. Puisque la capacité de chaque arc est une borne supérieure sur la quantité de flot qui peut traverser l'arc pendant l'intervalle d'étude (théorème 6), alors le flot calculé est une borne supérieure du flot maximum. □

Théorème 8 (Complexité de la méthode des capacités cumulées)**.** *Le calcul du flot maximum sur le graphe sous-jacent par la méthode des capacités cumulées se fait en temps $O(m \cdot T + m \cdot n^2)$.*

Démonstration. Pour chaque arc $(i; j)$ du graphe, à chaque pas de temps θ, on calcule la valeur $fcc_\theta(i; j)$ en temps constant. Donc le calcul de tous les vecteurs fcc du graphe se fait en temps $O(m \cdot T)$. Il en va de même pour le calcul des vecteurs bcc.

Pour chaque arc $(i; j)$ du graphe, on calcule la valeur u_{max} en temps $O(T)$. Donc le calcul de toutes les valeurs u_{max} du graphe se fait en temps $O(m \cdot T)$.

En utilisant l'algorithme de Push-Relabel, le calcul du flot sur le graphe sous-jacent, qui possède n nœuds et m arcs, se fait en temps $O(n^2 \cdot m)$.

Au total, la complexité de cette méthode est $O(m \cdot T + m \cdot n^2)$. □

Si l'on voulait résoudre le problème par le graphe développé, il faudrait appliquer un algorithme pour le flot maximum à un graphe composé de $T \cdot n$ nœuds et $T \cdot m$ arcs. En utilisant l'algorithme Push-Relabel, dont la complexité est $O(n^2 \cdot m)$, cela prendrait un temps en $O(T^3 \cdot n^2 \cdot m)$. La méthode par capacités cumulées, qui donne une borne supérieure du flot maximum, a l'immense avantage d'être linéaire en T.

3.2.2.2 Exemple

Afin de clarifier cette méthode, observons ce que l'on obtient lorsqu'on l'applique au graphe dynamique de la figure 3.4, expliqué en section 3.2.1.2.

Ce graphe est bien acyclique, on peut donc appliquer la méthode décrite précédemment. La première étape est de classer les arcs du graphe. Les étiquettes des nœuds forment un ordre topologique duquel on peut déduire l'ordre suivant sur les arcs : (1;2) ; (1;3) ; (2;3) ; (2;4) ; (3;4).

La figure 3.6 montre le graphe dynamique et sur chaque arc est indiqué en noir son vecteur de capacités, en rouge son vecteur fcc et en bleu son vecteur bcc. La figure 3.7 montre le graphe sous-jacent avec sur chaque arc la valeur de capacité u_{max} et le flot passant sur l'arc. Et enfin la figure 3.8 montre le graphe dynamique avec sur chaque arc le vecteur de flot et le vecteur de capacité et sur chaque nœud le vecteur indiquant les unités de flot stockées (pas de stockage à l'horizon de temps). Il s'agit d'un flot ayant une valeur maximum correspondant à la valeur obtenue sur le graphe sous-jacent.

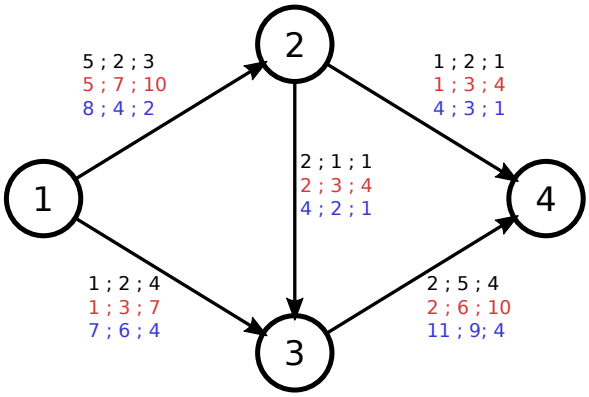


Figure 3.6 : Exemple de graphe dynamique sur 3 pas de temps (graphe de la figure 3.4). Le vecteur de capacité est indiqué sur chaque arc en noir, les vecteurs fcc et bcc sont indiqués respectivement en rouge et en bleu. Le nœud 1 est la source et le nœud 4 est le puits.

Nous calculons d'abord les vecteurs de capacités cumulées fcc. Nous parcourons pour cela les arcs dans l'ordre précédemment défini.

Aucun arc ne mène à l'arc (1;2), donc $fcc_\theta(1;2) = fcc_{\theta-1}(1;2) + u_\theta(1;2)$. Il s'agit donc simplement ici de cumuler les capacités successives de l'arc (1;2). On obtient donc $fcc(1;2) = (5,7,10)$.

On peut agir sur l'arc (1;3) de manière similaire à l'arc (1;2) car il n'a pas non plus d'arc précédent et on obtient $fcc(1;3) = (1,3,7)$.

L'arc (2;3) est précédé uniquement par l'arc (1;2). On a donc que $fcc_\theta(2;3) = \min\{fcc_{\theta-1}(2;3) + u_\theta(2;3)\ ;\ fcc_\theta(1;2)\}$. Dans ce cas présent, le minimum est obtenu en cumulant les capacités de l'arc, on a donc $fcc(2;3) = (2,3,4)$.

L'arc (2;4) est aussi précédé uniquement par l'arc (1;2). Comme précédemment, on a donc $fcc_\theta(2;4) = \min\{fcc_{\theta-1}(2;4) + u_\theta(2;4)\ ;\ fcc_\theta(1;2)\}$. Ici encore, le minimum est obtenu en cumulant les capacités de l'arc, et donc $fcc(2;4) = (1,3,4)$.

Enfin, l'arc (3;4) est précédé par deux arcs, (1;3) et (2;3). Donc $fcc_\theta(3;4) = \min\{fcc_{\theta-1}(3;4) + u_\theta(3;4)\ ;\ fcc_\theta(1;3) + fcc_\theta(2;3)\}$. Au premier pas de temps $fcc_1(3;4) = \min\{u_1(3;4)\ ;\ fcc_1(1;3) + fcc_1(2;3)\} = \min\{2\ ;\ 1+2\} = 2$. Au deuxième pas de temps, $fcc_2(3;4) = \min\{fcc_1(3;4)+u_2(3;4)\ ;\ fcc_2(1;3)+ fcc_2(2;3)\} = \min\{2+5\ ;\ 3+3\} = 6$. Au dernier pas de temps, $fcc_3(3;4) = \min\{fcc_2(3;4)+u_3(3;4)\ ;\ fcc_3(1;3)+ fcc_3(2;3)\} = \min\{6+4\ ;\ 7+4\} = 10$. Le vecteur $fcc(3;4) = (2,6,10)$. Ici, pour obtenir la valeur minimum, on a pris en compte les valeurs sur les vecteurs qui précèdent l'arc.

Ces vecteurs sont indiqués en rouge sur la figure 3.6.

Calculer les vecteurs bcc est très similaire, on parcourt pour cela les arcs dans le sens inverse et on cumule les capacités de la date T à la date 1 et le calcul effectué est le même.

Aucun arc ne suit l'arc $(3;4)$ donc $bcc_\theta(3;4) = bcc_{\theta+1}(3;4) + u_\theta(3;4)$. On cumule donc simplement les capacités successives de l'arc dans l'ordre antéchronologique. On obtient $bcc(3;4) = (11,9,4)$.

Tout comme l'arc $(3;4)$, l'arc $(2;4)$ n'est suivi par aucun arc. Nous allons donc cumuler les capacités de la même manière et on obtient $bcc(2;4) = (4,3,1)$.

L'arc $(2;3)$ est suivi uniquement par l'arc $(3;4)$. On utilise donc les valeurs de $bcc(3;4)$ pour calculer $bcc(2;3)$. On a $bcc_\theta(2;3) = \min\{bcc_{\theta+1}(2;3) + u_\theta(2;3)\ ;\ bcc_\theta(3;4)\}$. On obtient le minimum dans ce cas en cumulant simplement les capacités de l'arc, donc $bcc(2;3) = (4,2,1)$.

L'arc $(1;3)$ n'est suivi que de l'arc $(3;4)$ donc $bcc_\theta(1;3) = \min\{bcc_{\theta+1}(1;3) + u_\theta(1;3)\ ;\ bcc_\theta(3;4)\}$. Encore une fois, le minimum est obtenu en cumulant simplement les capacités de l'arc $(1;3)$. On obtient donc $bcc(1;3) = (7,6,4)$.

L'arc $(1;2)$ est suivi par les arcs $(2;3)$ et $(2;4)$. Il faut donc prendre en compte ces deux arcs pour calculer $bcc(1;2)$. On a alors $bcc_\theta(1;2) = \min\{bcc_{\theta+1}(1;2) + u_\theta(1;2)\ ;\ bcc_\theta(2;3) + bcc_\theta(2;4)\}$. Au dernier pas de temps de l'intervalle d'étude, $bcc_3(1;2) = \min\{u_3(1;2)\ ;\ bcc_3(2;3)+bcc_3(2;4)\} = \min\{3\ ;\ 1+1\} = 2$. Au deuxième pas de temps, $bcc_2(1;2) = \min\{bcc_3(1;2) + u_2(1;2)\ ;\ bcc_2(2;3) + bcc_2(2;4)\} = \min\{2+2\ ;\ 2+3\} = 4$. Au premier pas de temps $bcc_1(1;2) = \min\{bcc_2(1;2) + u_1(1;2)\ ;\ bcc_1(2;3) + bcc_1(2;4)\} = \min\{4+5\ ;\ 4+4\} = 8$. Le vecteur $bcc(1;2) = (8,4,2)$.

Ces vecteurs sont indiqués en bleu sur la figure 3.6.

À partir des vecteur fcc et bcc de chaque arc, on peut calculer la valeur u_{max} pour chaque arc. Il n'y a pas d'ordre particulier à respecter dans le parcours des arcs pour calculer cette valeur.

Pour l'arc $(1;2)$, on a $u_{max}(1;2) = \min\{bcc_1(1;2)\ ;\ fcc_3(1;2)\ ;\ fcc_1(1;2) + bcc_2(1;2)\ ;\ fcc_2(1;2) + bcc_3(1;2)\} = \min\{8\ ;\ 10\ ;\ 5+4\ ;\ 7+2\} = 8$.

Pour l'arc $(1;3)$, on a $u_{max}(1;3) = \min\{bcc_1(1;3)\ ;\ fcc_3(1;3)\ ;\ fcc_1(1;3) + bcc_2(1;3)\ ;\ fcc_2(1;3) + bcc_3(1;3)\} = \min\{7\ ;\ 7\ ;\ 1+6\ ;\ 3+4\} = 7$.

Pour l'arc $(2;3)$, on a $u_{max}(2;3) = \min\{bcc_1(2;3)\ ;\ fcc_3(2;3)\ ;\ fcc_1(2;3) + bcc_2(2;3)\ ;\ fcc_2(2;3) + bcc_3(2;3)\} = \min\{4\ ;\ 4\ ;\ 2+2\ ;\ 3+1\} = 4$.

Pour l'arc $(2;4)$, on a $u_{max}(2;4) = \min\{bcc_1(2;4)\ ;\ fcc_3(2;4)\ ;\ fcc_1(2;4) + bcc_2(2;4)\ ;\ fcc_2(2;4) + bcc_3(2;4)\} = \min\{4\ ;\ 4\ ;\ 1+3\ ;\ 3+1\} = 4$.

Pour l'arc $(3;4)$, on a $u_{max}(3;4) = \min\{bcc_1(3;4)\ ;\ fcc_3(3;4)\ ;\ fcc_1(3;4) + bcc_2(3;4)\ ;\ fcc_2(3;4) + bcc_3(3;4)\} = \min\{11\ ;\ 10\ ;\ 2+9\ ;\ 6+4\} = 10$.

Avec une valeur unique sur chaque arc, on s'intéresse à présent au graphe sous-jacent. Il s'agit d'un graphe statique dont la capacité de l'arc $(i;j)$ est la valeur $u_{max}(i;j)$. On peut donc aisément calculer un flot maximum sur ce graphe donc la valeur est 14 dans le cas présent. La figure 3.7 montre le graphe sous-jacent avec les valeurs u_{max} ainsi que la quantité de flot sur chaque arc.

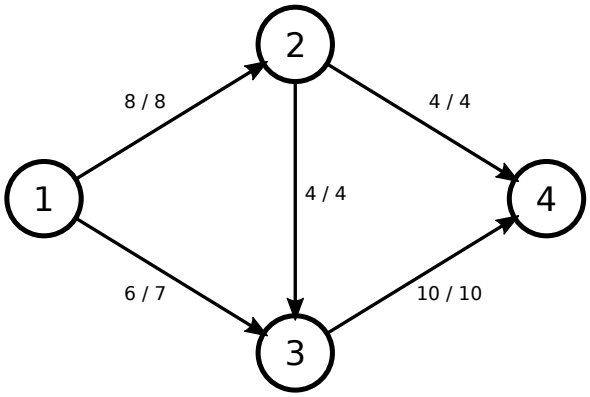


Figure 3.7 : Graphe sous-jacent correspondant au graphe dynamique de la figure 3.6. Sur chaque vecteur est indiqué le flot sur la capacité u_{max}.

Les valeurs u_{max} sur le graphe sous-jacent ont permis d'obtenir une valeur de flot circulant sur chaque arc. Il faut maintenant parvenir à trouver un flot dynamique dont la valeur sur chaque arc corresponde au u_{max} trouvé.

Puisque le stockage peut être infini sur les nœuds, on peut choisir d'envoyer les unités de flot au plus tôt sur chaque arc.

Dans cet exemple, nous avons été en mesure de trouver un flot dynamique de valeur 14. Il est pour cela nécessaire de stocker des unités de flot. Une solution est de stocker deux unités de flot sur le nœud 2 entre le premier et le deuxième pas de temps et également entre le deuxième et le troisième pas de temps, et de stocker aussi une unité de flot sur le nœud 3 entre le premier et le deuxième pas de temps.

Les valeurs du flot dynamique ainsi que les unités stockées sont indiquées sur la figure 3.8.

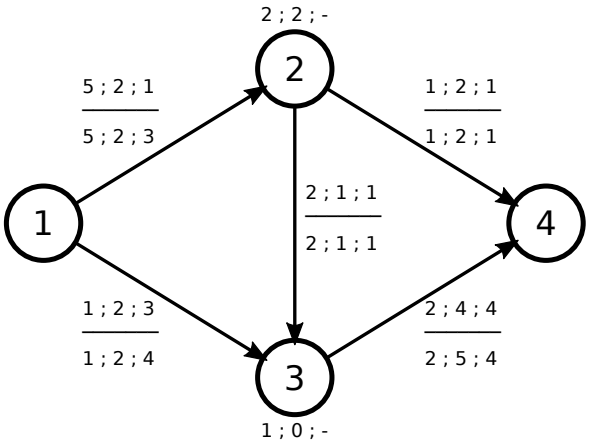


Figure 3.8 : Flot maximum du graphe de la figure 3.6.

3.2.2.3 Limites

Il peut être relativement compliqué de calculer un flot dynamique à partir du flot sur le graphe sous-jacent. En effet, il faut choisir, localement sur chaque arc, comment répartir les unités de flot entre les différents pas de temps, mais aussi choisir sur chaque nœud comment répartir les unités des arcs entrants sur les arcs sortants alors même que l'on ne dispose d'aucun critère permettant de faire ces choix.

En plus de cette difficulté, le flot obtenu sur le graphe sous-jacent avec les capacités u_{max} ne donne pas un flot optimal dans le graphe dynamique. Il s'agit d'une borne supérieure du flot optimal qui ne peut pas toujours être atteinte.

Faute d'être en mesure de prouver que cette méthode était exacte ou de trouver aisément un contre-exemple montrant qu'elle ne l'était pas, nous avons décidé de créer des instances aléatoires afin de laisser à la machine le soin de générer un contre-exemple.

Nous avons pour cela implémenté cette méthode en utilisant la librairie de gestion de graphes GraphStream.

Nous avons généré le graphe sous-jacent en utilisant le générateur Random Euclidean de GraphStream qui construit des graphes aléatoires géométriques (ce type de graphe est défini dans la section 4.4.1 page 108). Nous avons généré des graphes de 4 nœuds et 3 pas de temps pour lesquels il existe un arc entre chaque pair de nœuds. Pour chaque arc, la capacité au premier pas de temps est choisie entre 10 et 100 puis est incrémentée de 20% avec une probabilité $0,25$ et décrémentée de 20% avec une probabilité $0,25$.

Une fois le graphe généré, on crée le graphe développé correspondant. On applique la méthode des capacités cumulées sur le graphe dynamique et on calcule le flot maximum sur le graphe développé. On compare les deux valeurs, et si elles sont égales, on génère une autre instance.

Cette méthode de génération donne des capacités tendant vers 0 pour de larges horizons de temps. Cela nous a néanmoins permis de générer des contre-exemples montrant que la méthode des capacités cumulées ne donne pas le flot maximum.

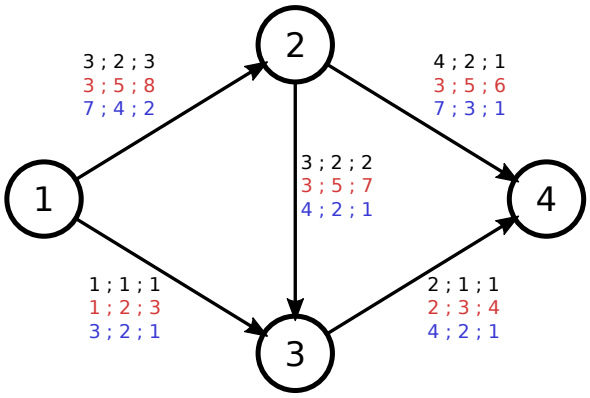


Figure 3.9 : Exemple de graphe dynamique sur 3 pas de temps. Le vecteur de capacité est indiqué sur chaque arc en noir, les vecteurs fcc et bcc sont indiqués respectivement en rouge et en bleu. Le nœud 1 est la source et le nœud 4 est le puits.

La figure 3.9 montre un exemple simple pour lequel le flot obtenu sur le graphe sous-jacent avec les capacités u_{max} est supérieur au flot maximum du graphe dynamique.

Il s'agit d'un graphe dynamique sur 3 pas de temps. Le vecteur de capacité est indiqué en noir sur chaque arc. Les vecteurs fcc et bcc ont été calculés comme décrit précédemment et sont aussi indiqués sur les arcs.

Grâce aux deux vecteurs de capacités cumulées nous pouvons déduire la capacité maximum u_{max}. La figure 3.10 montre le graphe sous-jacent sur lequel sont indiqués les capacités u_{max} et le flot maximum. Ce flot a une valeur 10.

La figure 3.11 montre le flot maximum du graphe dynamique. La valeur de ce flot est 9. Si on voulait envoyer 10 unités de flot dans ce graphe, il faudrait pouvoir envoyer 3 unités sur l'arc $(2;4)$ et une unité sur l'arc $(2;3)$ au premier pas de temps. Hors, seules 3 unités peuvent arriver sur le nœuds 2 au premier pas de temps (par l'arc $(1;2)$). Il est donc impossible de satisfaire les valeurs de flot sur les deux arcs, et cette situation ne peut pas être détectée avec les capacités cumulées.

Dans cet exemple, il n'est pas possible de construire un flot dont la valeur sur chaque arc est égale à la valeur du flot sur le graphe sous-jacent.

Afin de déterminer à quel point notre borne est éloignée de la valeur optimale, nous comparons les deux valeurs sur des graphes générés aléatoirement. Pour cela, comme précédemment, nous avons d'abord généré le graphe sous-jacent en utilisant le générateur Random Euclidean de GraphStream qui construit des graphes aléatoires géométriques. Nous avons généré des graphes de 100 nœuds et 15 pas de temps pour lesquels il existe un arc entre chaque pair de nœuds. On ajoute de la dynamique au graphe de la façon suivante : pour chaque arc, la capacité au premier pas de temps est choisie entre 10 et 30 puis est incrémentée de 50% avec une probabilité $0,4$ et décrémentée de 50% avec une probabilité $0,4$.

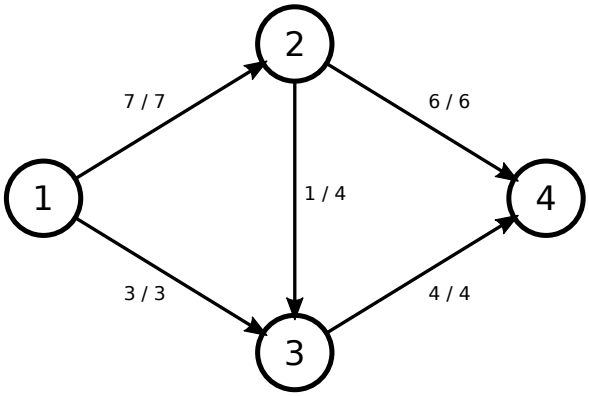


Figure 3.10 : Graphe sous-jacent correspondant au graphe dynamique de la figure 3.9. Sur chaque vecteur est indiqué le flot sur la capacité u_{max}.

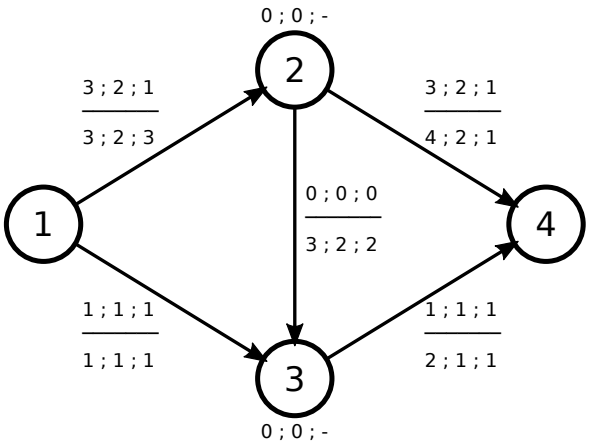


Figure 3.11 : Flot maximum du graphe de la figure 3.9.

On applique la méthode des capacités cumulées sur le graphe dynamique et on calcule le flot maximum sur le graphe développé. On compare les deux valeurs afin de déterminer par quel pourcentage la borne dépasse la valeur exacte. Cette expérience a tourné sur 1342 instances de graphes. On donne dans le tableau 3.1 les répartitions de ces valeurs.

On observe que la borne supérieure est au plus 21,09% au delà de la valeur exacte, et que dans les trois quart des cas, la borne supérieure dépasse la valeur exacte de moins de 5%.

	Borne supérieure
Minimum	0,00%
Q1	0,79%
Moyenne	3,43%
Médiane	2,41%
Q3	4,93%
Maximum	21,09%

Table 3.1 : Répartition des pourcentage au delà de la valeur exacte du flot maximum.

3.3 Flot de cout minimum

Nous présentons ici les résultats obtenus sur le flot de cout minimum dans le cadre d'une collaboration franco-polonaise avec Maciej Drozdowski (Université technologique de Poznań). Nous présentons d'abord le contexte de ce travail, en décrivant le modèle et le problème étudiés, les possibles applications et la méthode classique de résolution. Nous présentons ensuite la nouvelle approche que nous proposons en détaillant l'algorithme que nous avons conçu, dont nous montrons l'utilisation au travers d'un exemple et dont nous prouvons la correction et la complexité. Enfin, nous présentons l'étude expérimentale que nous avons réalisée afin de comparer la méthode classique avec notre nouvelle approche. Ce travail a fait l'objet d'un article publié (Vernet et al., 2020).

3.3.1 Contexte

3.3.1.1 Modèle

Tel que présenté dans le chapitre 2, nous utilisons un modèle de graphe dynamique pour lequel nous connaissons à l'avance les modifications et celles-ci se font sans incertitude (voir figure 2.3).

La dynamique du graphe, quant à elle, agit sur les capacités et les couts qui varient avec le temps. Les arêtes n'ont pas de temps de traversée. Cela signifie que pour traverser une arête $(i; j)$, si une unité de flot part du nœud i à la date θ, elle arrive au nœud j aussi à la date θ. Le stockage sur les nœuds est interdit dans notre modèle. Cela signifie que lorsqu'une unité de flot arrive sur un nœud i à un pas de temps θ, elle doit immédiatement en partir à ce même temps θ.

Par simplification du modèle mais sans perte de généralité, on considère que les arêtes du graphe sont présentes pendant toute l'intervalle d'étude du graphe, tout comme les nœuds du graphe. Il est possible de modéliser l'absence d'une arête en donnant 0 comme valeur de capacité ou ∞ comme valeur de cout sur l'arête en question. Cette arête ne sera alors pas choisie dans une solution optimale. Par extension, il est aussi possible de simuler l'absence d'un nœud en simulant l'absence de toutes ses arêtes adjacentes, le nœud en question devient alors inaccessible.

En conservant les notations utilisées figure 3.1, le modèle utilisé est synthétisé dans la figure 3.12.

On note u la capacité des arêtes en nombre d'unité de flot et dépendant du temps. Précisément, $u_\theta(i; j)$ unités de flot au maximum peuvent traverser l'arête $(i; j)$ au pas de temps θ. On note c le cout unitaire des arêtes. Une unité de flot traversant l'arête $(i; j)$ au pas de temps θ coute $c_\theta(i; j)$.

Les capacités et les couts sont positifs.

3.3.1.2 Problème

Nous nous intéressons ici au problème de flot de cout minimum dans un graphe dynamique dont le modèle à été décrit précédemment. Le principe général de ce problème est d'acheminer une certaine quantité de flot dans un graphe depuis une ou plusieurs sources vers un ou plusieurs puits. L'objectif est de réaliser cet acheminement de la façon la plus économique possible. On cherche donc à minimiser le cout total du flot qui correspond au cout de chaque unité de flot pour chacune des arêtes qu'elle a traversées.

Une source et un puits sont définis parmi l'ensemble des nœuds du graphe. Dans le problème que nous étudions, la source ainsi que le puits sont uniques. Ceci nous permet de faciliter la modélisation. Le nœud source est noté s. Le nœud puits est noté t.

Une quantité Q, $Q \in \mathbb{N}$, en nombre d'unité, doit être acheminée de la source s vers le puits t. On dit que s a une disponibilité qui vaut Q et que t à une demande qui vaut Q.

L'intégralité du flot Q à transmettre doit arriver au puits avant l'horizon de temps. Un flot pour lequel la demande de t ne serait pas satisfaite au plus tard au pas de temps T ne représenterait pas une solution réalisable du problème.

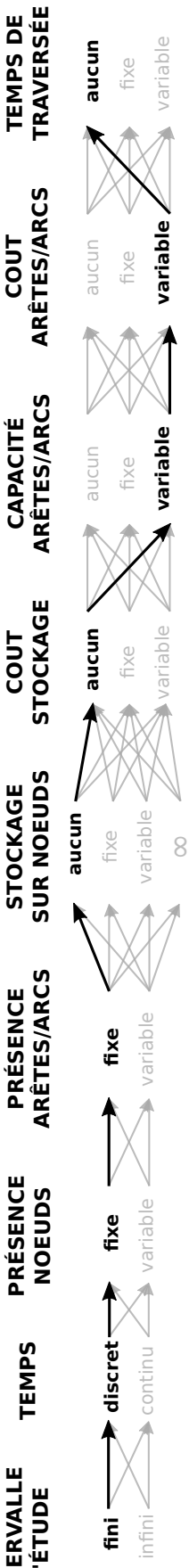


Figure 3.12 : Illustration du modèle de graphe dynamique étudié pour le problème de flot de cout minimum.

La fonction objectif à optimiser est la suivante :

$$\min \sum_{\theta \in \mathcal{T}} \sum_{(i;j) \in E} f_\theta(i;j) \cdot c_\theta(i;j) \tag{3.16}$$

en respectant les contraintes suivantes :

$$0 \leqslant f_\theta(i;j) \leqslant u_\theta(i;j) \qquad \forall \theta \in \mathcal{T}, \forall (i;j) \in E \tag{3.17}$$

$$\sum_{j \in V} \sum_{\theta \in \mathcal{T}} f_\theta(s;j) = Q \tag{3.18}$$

$$\sum_{j \in V} \sum_{\theta \in \mathcal{T}} f_\theta(j;t) = Q \tag{3.19}$$

$$\sum_{k \in V} f_\theta(k;i) = \sum_{j \in V} f_\theta(i;j) \qquad \forall \theta \in \mathcal{T}, \forall i \in V \backslash \{s,t\} \tag{3.20}$$

L'objectif, exprimé en (3.16), est de minimiser le cout total du flot. Les contraintes ont la même signification que celles exprimées en section 3.1 mais sont adaptées au cas spécifique présent.

L'équation (3.17) assure que le flot respecte les capacités des arcs. Les équations (3.18) et (3.19), correspondant à l'équation (3.2), permettent de garantir que la quantité de flot qui part de la source est la même que celle qui arrive au puits et correspond bien à la disponibilité Q.

Par soucis de symétrie par rapport aux contraintes exprimées en section 3.1, nous avons conservé ici la contrainte (3.19), bien qu'elle soit redondante avec la contrainte (3.18) pour le problème traité.

Enfin l'équation (3.20) traduit la conservation du flot. Son écriture est beaucoup plus simple que la forme générique donnée par l'équation (3.3). En effet, comme le stockage sur les nœuds est interdit, la variable y n'apparait pas et l'absence de temps de traversée sur les arcs simplifie aussi l'écriture de la quantité de flot qui arrive en un nœud à un temps donné.

La disponibilité Q à envoyer dans le graphe est définie globalement pour tout l'intervalle d'étude. C'est justement sa définition globale, et non pour chaque pas de temps de l'intervalle, qui rend le problème algorithmiquement pertinent sur notre modèle.

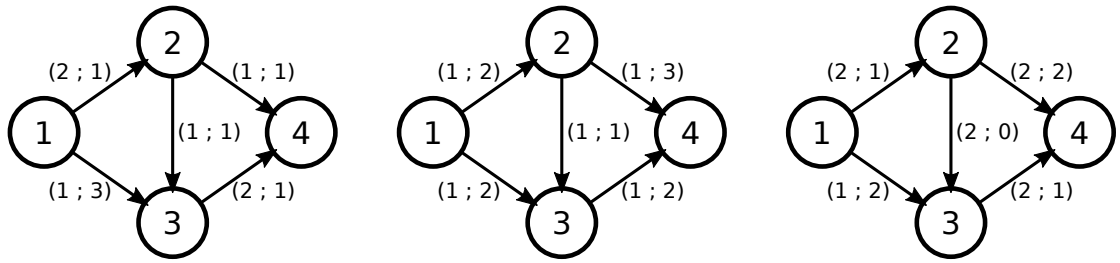
En effet, une demande qui serait faite à chaque pas de temps impliquerait de résoudre T problèmes de flot de cout minimum indépendants. Comme ici la demande est globale, il n'est pas possible de faire une résolution indépendamment sur chaque t-graphe.

Résoudre T problèmes de flot de cout minimum sur les T t-graphes ne donne pas une solution globale au problème. Puisque les couts des arêtes varient au cours du temps, il peut être nécessaire ne de pas envoyer de flot sur certains pas de temps afin d'atteindre la solution optimale. Et nous ne pouvons pas connaitre à l'avance les pas de temps les moins chers. Donc il est impossible de savoir à l'avance quelle quantité de flot envoyer dans chaque t-graphe.

La figure 3.13 montre tous les t-graphes d'un graphe dynamique sur 3 pas de temps ($\mathcal{T} = \{1;2;3\}$). La source est le nœud 1, le puits est le nœud 4 et $Q = 4$. Sur chaque arc est noté sa capacité et son cout unitaire.

Afin d'atteindre l'optimalité, il est nécessaire dans cet exemple de ne rien envoyer au pas de temps 2. En effet, il serait trop cher d'envoyer du flot sur G_2. Deux unités doivent être envoyée dans G_1. Un première sur le chemin $(1;2;4)$ dont le cout unitaire est 2 et une deuxième sur le chemin $(1;2;3;4)$ dont le cout unitaire est 3. Deux autres unités doivent être envoyées dans G_3, toutes les deux sur le chemin $(1;2;3;4)$ dont le cout unitaire est 2. Au total, cette solution coute 9. Il s'agit de la solution optimale.

Dans cet exemple, on remarque qu'aucun pas de temps ne permet d'envoyer les 4 unités de flot. Au premier pas de temps, seules 3 unités peuvent être envoyées au maximum et cela couterait 9. Au deuxième pas de temps, au maximum 2 unités peuvent être envoyées et cela couterait 9. Au troisième pas de temps, 3 unités peuvent être envoyées au maximum et cela couterait 8. Le flot de valeur 4 de cout minimum sur $\mathcal{T}$ ne peut pas être déduit des flots de cout minimum à chaque pas de temps.



(a) G_1 (b) G_2 (c) G_3

Figure 3.13 : Graphe dynamique sur 3 pas de temps. Chaque t-graphe est représenté. Sur chaque arc $(i; j)$, le couple $(u(i; j); c(i; j))$ représente la capacité et le cout de l'arc $(i; j)$, respectivement. Dans cet exemple, 4 unités de flot doivent être envoyées.

D'autre part, on remarque également que le flot maximum n'est atteint dans aucun des t-graphes afin d'obtenir la solution optimale. Une unité peut encore être envoyée au premier pas de temps, deux unités peuvent être envoyées au deuxième pas de temps sur lequel rien n'a été envoyé, et une unité peut encore être envoyée au troisième pas de temps. Il est clair avec cet exemple qu'il serait aussi couteux qu'inutile de calculer les flots maximum à chaque pas de temps et le cout minimum associé.

3.3.1.3 Résolution par graphe développé

Comme nous avons pu le voir en section 3.1, et en particulier en 3.1.3.1, la méthode la plus largement utilisée pour résoudre les problèmes de ce type consiste en l'utilisation du graphe développé. Comme cela a été défini en section 2.4, le graphe développé est un graphe statique contenant toutes les informations du graphe dynamique correspondant.

Puisqu'il s'agit d'un graphe statique, il est possible d'utiliser directement les méthodes de résolution déjà connues pour ce problème. Grâce à l'exacte équivalence entre le graphe dynamique et le graphe développé (voir section 3.1.3.1), la solution optimale obtenue dans le graphe développé peut être directement transformée en une solution optimale équivalente dans le graphe dynamique.

Cette méthode est évidemment très pratique puisqu'on obtient la solution directement par des algorithmes connus sans avoir besoin de créer de nouveaux algorithmes qui seraient spécifiques aux graphes dynamiques.

En revanche, passer par le graphe développé est particulièrement inefficace. Lorsque l'on étudie un graphe dynamique de n nœuds et m arcs, le graphe développé correspondant est T fois plus gros puisque les n nœuds et les m arcs sont dupliqués T fois. La taille du graphe sur lequel l'algorithme de résolution du problème est appliqué est bien plus conséquente que la taille du graphe dynamique d'origine. Cela aura donc un impact évident sur la complexité de l'algorithme lorsque celui-ci est utilisé sur un graphe développé.

La figure 3.14 donne une représentation schématique du graphe développé correspondant au modèle de graphe dynamique utilisé ici. Dans notre modèle, le graphe dynamique a n nœuds, m arcs et T pas de temps. Donc les n nœuds vont être dupliqués T fois et les m arcs seront aussi dupliqués T fois. Comme il n'y a pas de temps de traversée sur les arcs, tous les arcs $(i; j)$ du graphe dynamique sont représentés par des arcs de la forme $(i_\theta; j_\theta)$ dans le graphe dynamique et celui-ci ne contient pas d'arcs de la forme $(i_\theta; j_{\theta'})$ où $\theta \neq \theta'$. De la même manière, le stockage d'unités de flot sur les nœuds étant interdit, on n'ajoute pas d'arêtes pour représenter le stockage dans le graphe développé. Celui-ci ne contient donc pas d'arcs de la forme $(i_\theta; i_{\theta+1})$, $\theta \in \mathcal{T}$. Afin de rester dans la configuration de source unique et de

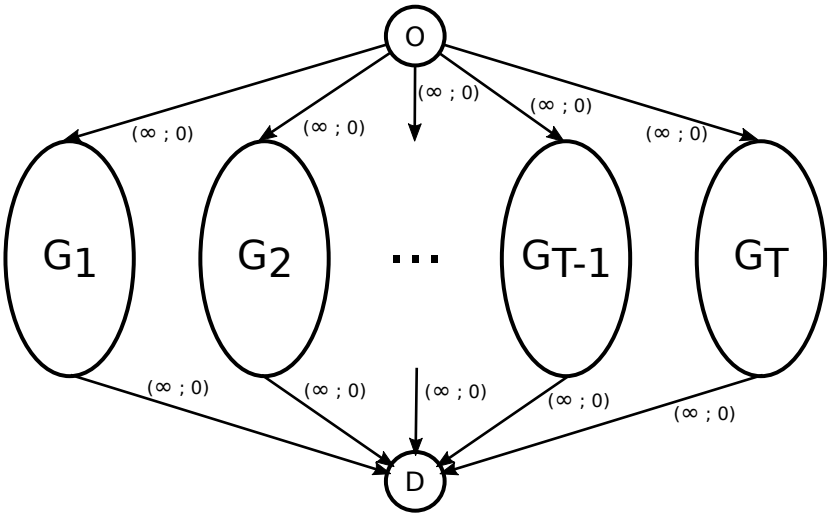


Figure 3.14 : Représentation schématique du graphe développé correspondant à notre modèle de graphe dynamique pour le flot de cout minimum. La super-source est notée O, le super-puits est noté D.

puits unique, on ajoute au graphe développé une *super-source* et un *super-puits*. On ajoute un arc de la super-source vers le nœud source à chaque pas de temps ayant la capacité ∞ et le cout 0. Cela représente T arcs supplémentaires. On ajoute de la même manière un arc de chaque nœud puits à chaque pas de temps vers le super-puits avec une capacité ∞ et un cout 0. Cela fait aussi T arcs supplémentaires. De façon générale, l'arc $(i_\theta; j_\theta)$ a une capacité qui vaut $u_\theta(i; j)$ et un cout unitaire qui vaut $c_\theta(i; j)$.

Au total, le graphe développé correspondant à notre modèle de graphe dynamique possède $T \cdot n + 2$ nœuds et $T \cdot m + 2 \cdot T$ arcs. Le graphe dynamique donné en figure 3.13 a 4 nœuds, 5 arcs, et 3 pas de temps. Le graphe développé correspondant, montré dans la figure 3.15, possède 14 nœuds et 21 arcs. On vérifie facilement avec cet exemple simple que le graphe obtenu possède un nombre de nœuds et d'arcs très fortement augmenté.

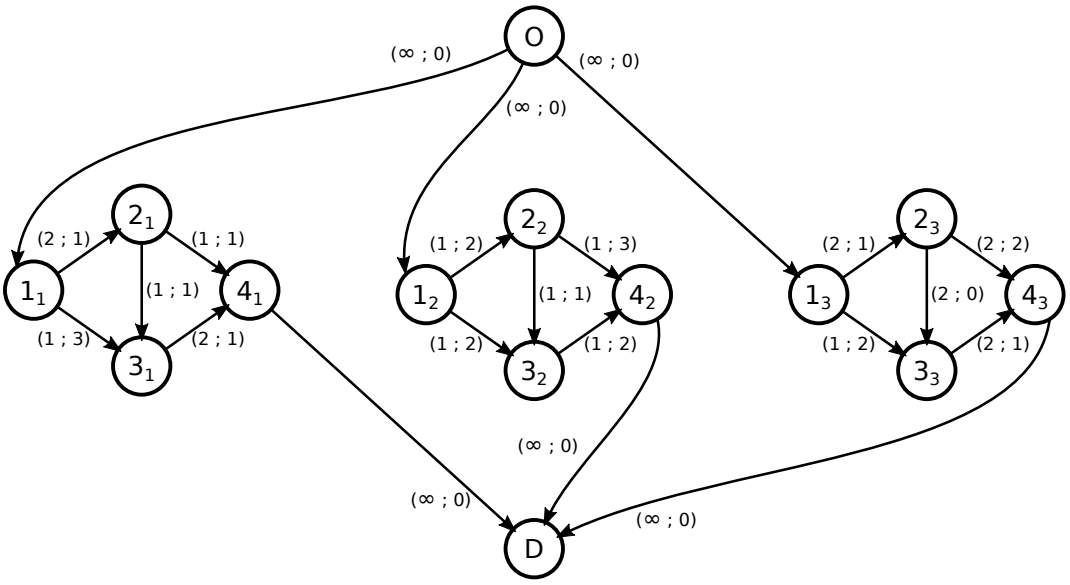


Figure 3.15 : Graphe développé correspondant au graphe dynamique donné en figure 3.13. Le nœud noté O est la super-source, le nœud noté D est le super-puits.

L'algorithme *Successive Shortest Path* (que l'on traduira par *Plus Courts Chemins Successifs*) permet de résoudre efficacement le problème du flot de cout minimum en cherchant, successivement dans le graphe, des plus courts chemins, comme son nom l'indique. On désignera dans la suite cet algorithme par ses initiales, *SSP*. Cet algorithme, créé initialement pour des graphes statiques, est décrit dans (Ahuja et al., 1993). Nous en rappellerons les principes en partie 3.3.2.

La complexité de SSP dans le cas général est $O(U \cdot (m + n \cdot \log(n)))$ où U est la disponibilité maximum. Dans notre cas, il n'y a qu'une seule source, donc $U = Q$. SSP utilise l'algorithme de Dijkstra pour trouver les plus courts chemins à chaque itération, d'où le fait que l'on retrouve la complexité de Dijkstra $(m + n \cdot \log(n))$ dans l'expression de la complexité de SSP.

L'algorithme SSP peut être utilisé directement sur le graphe développé pour résoudre le problème de flot de cout minimum. La complexité de SSP sur le graphe développé est $O(Q \cdot T \cdot (m + n \cdot \log(n \cdot T)))$ puisque celui-ci possède de l'ordre de $n \cdot T$ nœuds et de l'ordre de $m \cdot T$ arcs. Dans la suite, cette méthode sera notée SSP_{DVP}.

L'objectif que nous souhaitons atteindre en proposant une nouvelle méthode est de réduire la complexité de résolution du flot de cout minimum. Pour cela, nous souhaitons créer un algorithme qui n'utilise pas le graphe développé, car c'est sa trop grande taille qui pénalise la complexité de calcul.

On notera par ailleurs que l'utilisation du graphe développé n'est possible que lorsqu'il n'y aucune incertitude dans le modèle de graphe dynamique. En effet, dans le cas contraire, le graphe développé est impossible à construire. Cela correspond donc très bien au cas étudié ici, mais n'est pas une solution universelle pour tous les problèmes de flots dans n'importe quel graphe dynamique.

3.3.1.4 Applications

Ce travail sur le flot de cout minimum a pour objectif d'apporter des résultats de complexité dans un cadre dynamique et de proposer un algorithme efficace pour résoudre le problème. Nous devons concéder que notre modèle peut paraitre assez basique. Néanmoins, nous avons su l'exploiter pour proposer une méthode de résolution efficace comme cela est montré dans la suite.

De plus, il existe de nombreux cas d'applications pour lesquels notre modèle est pertinent, et donc pour lesquels résoudre un problème de flot de cout minimum a un sens en pratique. Puisque notre modèle n'a pas de temps de traversée sur les arcs, on retrouve cette spécificité dans chacun des cas d'application. Soit le temps mis par une unité de flot pour se rendre d'un nœud à un autre est effectivement nul, soit ce temps n'est pas nul mais est négligeable devant la dynamique du réseau dans son ensemble. C'est-à-dire que les unités de flot se déplacent nettement plus vite que le graphe ne met de temps pour évoluer. Les réseaux de communications, les réseaux logistiques ou encore les réseaux électriques ont cette particularité.

Concernant le cas des réseaux de communication, l'un des problèmes majeurs est de transporter une grande quantité de données. Les modèles de graphes et les algorithmes associés ont largement été utilisés en ce sens depuis de nombreuses années. On citera par exemple le livre de Bertsekas et al. (1992) ou bien celui de Koster et Muñoz (2009).

Observons en particulier le problème qui consiste à envoyer une grande quantité de données vers un serveur. Ces données peuvent être séparées en un ensemble de paquets de plus petite taille.

Ces paquets peuvent être envoyés séparément. Ils ne peuvent en revanche pas être stockés sur des nœuds intermédiaires. Le temps que met un paquet pour traverser le réseau est très court comparé au temps nécessaire au transfert de la totalité des données.

La capacité de chaque partie du réseau dépend de ce qui a été autorisé en amont par le fournisseur du réseau. Le cout de transfert est également défini par le fournisseur. Ce dernier peut modifier au cours du temps les capacités et les couts en fonction de ses besoins et de ses possibilités.

La façon optimale de transférer des données selon ces conditions peut être calculée grâce à notre modélisation.

On remarquera que ce problème, décrit ici de façon très générique, s'applique aussi bien au transfert de données scientifiques dans des WAN qu'au streaming mais aussi aux réseaux pair à pair ou encore aux réseaux de capteurs.

Dans les réseaux logistiques, quel que soit le mode de transport utilisé, les différentes routes sont soumises aux changements, que ce soit pour des raisons de disponibilités ou de capacité. En effet, de nouvelles routes maritimes peuvent être créées quand d'anciennes peuvent être supprimées, le planning de l'occupation des tronçons ferroviaires peut varier, une route peut être momentanément bloquée pour cause de travaux par exemple. On peut imaginer d'autres cas encore.

Les couts associés aux trajets effectués peuvent aussi varier. Des péages n'ont pas nécessairement des tarifs fixes, ils peuvent dépendre de la saison, du jour de la semaine, de l'heure de la journée par exemple. Les couts des trajets sont aussi dépendants des prix des différents types de carburant qui varient dans le temps.

On peut cependant noter que tous ces changements ne sont pas si fréquents et qu'en général, ils peuvent être anticipés. Il faut donc veiller à étudier ces réseaux avec un horizon de temps suffisamment lointain, et avec des pas de temps appropriés.

Le stockage, quant à lui, peut être extrêmement couteux. C'est la raison pour laquelle il est souvent préférable de l'éviter.

On peut citer comme exemple les travaux de Démare et al. (2017) où ils décrivent un modèle permettant de simuler du trafic de biens dans la vallée de Seine.

Dans les marchés d'électricité dérèglementés, l'électricité est vendue et achetée sur des marchés pour les 24 heures suivant la transaction (Weron, 2014), par intervalles d'une heure. De cette manière, le transfert d'énergie dans le réseau électrique est connu avec un jour d'avance.

Certains fournisseurs peuvent commander trop peu d'énergie, auquel cas, et afin de conserver la stabilité du réseau, l'énergie doit alors être fournie depuis des sources alternatives très rapidement sur ordre de l'opérateur du système.

Ce réseau est effectivement dynamique car les lignes disponibles changent toutes les heures et les couts de transmission varient aussi dans la journée. Le temps de traversée est complètement négligeable et le stockage d'énergie dans le réseau lui-même est parfaitement impossible.

Un flot de cout minimum d'après notre modèle peut permettre de construire un plan d'équilibrage du réseau à moindre cout.

3.3.2 Nouvelle approche

Comme expliqué précédemment, l'utilisation du graphe développé pour résoudre le problème de flot de cout minimum n'est pas une méthode efficace. C'est la raison pour laquelle nous proposons une méthode qui résout le problème directement sur le graphe dynamique sans passer par le graphe développé. Notre méthode est basée sur l'algorithme SSP.

3.3.2.1 Rappels sur l'algorithme Successive Shortest Path

L'algorithme SSP est basé sur la recherche successive de plus courts chemins dans le graphe. Il est décrit avec précisions dans (Ahuja et al., 1993), mais nous en donnons néanmoins les principaux éléments ici.

SSP travaille, comme beaucoup d'algorithmes de flots, sur le graphe résiduel. Pour rappel, le graphe résiduel correspond à l'état du graphe lorsque l'on y envoie un flot donné, et évolue avec chaque nouvelle

unité de flot envoyée. Il représente principalement les capacités restantes des arcs. Il a des capacités résiduelles et des couts résiduels différents des capacités originales et des couts originaux.

Le principe de l'algorithme est de trouver un plus court chemin dans le graphe résiduel, ici il s'agit de « plus court » au sens des couts, donc il s'agit en réalité d'un chemin le moins cher. Une fois trouvé, ce chemin est saturé, c'est-à-dire qu'on y envoie le plus de flot possible. Le graphe résiduel est ensuite mis à jour. Ces trois étapes représentent une itération de SSP. L'algorithme utilisé pour trouver les chemins est l'algorithme de Dijkstra. La première itération de SSP se fait sur le graphe de départ.

Sur chaque nœud i dans le graphe résiduel est défini un potentiel, noté $\pi(i)$. Il est mis à jour à chaque itération lors de la mise à jour du graphe résiduel. Sa valeur de départ est 0 et il est mis à jour en utilisant la formule suivante : $\pi(i) = \pi(i) - d(i)$, où $d(i)$ est la distance de la source à i donnée par l'algorithme de Dijkstra. Ce potentiel est nécessaire pour calculer les couts réduits.

On appelle cout réduit le cout de l'arc $(i; j)$ dans le graphe résiduel. Il est noté $\bar{c}(i; j)$. À la première étape $\bar{c}(i; j) = c(i; j)$ et il est ensuite mis à jour de la façon suivante : $\bar{c}(i; j) = c(i; j) + \pi(j) - \pi(i)$. Ces couts réduits nous assurent de travailler avec des couts toujours positifs, ce qui est essentiel pour pouvoir utiliser l'algorithme de Dijkstra.

On note $f(i; j)$ le flot courant sur l'arc $(i; j)$. La capacité résiduelle d'un arc, notée $\bar{u}(i; j)$, dépend du f, et vaut toujours $u(i; j) - f(i; j)$.

Pour chaque arc $(i; j)$ du graphe, le graphe résiduel contient aussi l'arc $(j; i)$, appelé *non conforme*, donc la capacité résiduelle $\bar{u}(j; i)$ vaut $f(i; j)$ et le cout réduit $\bar{c}(j; i)$ vaut $-\bar{c}(i; j)$.

Cet algorithme est optimal à chaque itération. Cela vient du fait que le cout unitaire du chemin trouvé à une itération donnée est supérieur ou égal au cout unitaire du chemin trouvé à l'itération précédente. Cette propriété, qui nous sera très utile par la suite, est obtenue grâce aux couts réduits qui, de par leur construction, assurent une équivalence entre la solution optimale du problème initial et la solution optimale du problème avec ces couts réduits, voir Ahuja et al. (1993).

La complexité totale de l'algorithme SSP, en utilisant l'algorithme de Dijkstra, est $O(Q \cdot (m + n \cdot \log(n)))$.

3.3.2.2 Algorithme Dynamic SSP

L'algorithme que nous proposons pour résoudre le problème de flot de cout minimum dans notre modèle de graphe dynamique est basé sur l'algorithme SSP défini pour les graphes statiques. Dans la suite, nous appellerons notre algorithme *DSSP* (pour *Dynamic SSP*). DSSP est détaillé dans l'algorithme 1.

L'idée générale de notre algorithme est de réaliser, à chaque itération de DSSP, une itération de l'algorithme SSP sur un seul t-graphe. Nous traitons donc les t-graphes indépendamment les uns des autres. Cela est rendu possible grâce au fait qu'il n'y a pas de temps de traversée sur les arcs et que le stockage de flot sur les nœuds est interdit.

L'avantage majeur est que nous travaillons sur des graphes de tailles raisonnables (n nœuds et m arcs) contrairement à la méthode qui consiste à utiliser le graphe développé.

DSSP prend en entrée un graphe dynamique G et son intervalle d'étude $\mathcal{T} = \{1, \ldots, T\}$, la source s, le puits t ainsi que la quantité disponible Q à envoyer. L'algorithme renvoie un flot f qui, pour chaque arc du graphe à chaque pas de temps de l'intervalle, a une valeur entière positive.

DSSP utilise les caractéristiques du graphe résiduel. Ici le graphe résiduel est dynamique et est aussi composé de t-graphes, comme le graphe dynamique d'origine. Il y a un t-graphe résiduel pour chaque t-graphe G_θ du graphe dynamique. Comme pour le cas statique, les couts utilisés sont les couts réduits, notés $\bar{c}$. Les capacités résiduelles sont notées $\bar{u}$. Au début de l'algorithme, le graphe résiduel est le même que le graphe dynamique, $\bar{u} = u$ et $\bar{c} = c$. Comme pour l'algorithme SSP dans un graphe statique, on utilise ici aussi un potentiel sur les nœuds que l'on note π. À chaque nœud i et pas de temps θ est associée un potentiel $\pi_\theta(i)$ qui vaut 0 au début de l'algorithme.

Algorithme 1 : DSSP

Input : graphe dynamique G ; intervalle d'étude $\mathcal{T}$; nœud source s ; nœud puits t ; disponibilité Q
Output : fonction flot $f : \mathcal{T} \times E \to \mathbb{N}$

1 **pour chaque** $\theta \in \mathcal{T}$ **faire**
2 $\quad p_\theta \leftarrow$ `Dijkstra`(G_θ, c, s, t)
3 $\quad$ `insérer`(Chemins, $p_\theta, c(p_\theta)$)
4 **fin**
5 $x \leftarrow 0$
6 $f_\theta(i;j) = 0 \ \forall \theta \in \mathcal{T}, (i;j) \in E$
7 **tant que** $x < Q$ **faire**
8 $\quad \bar{\theta} \leftarrow \theta \in \mathcal{T}$ tel que p_θ est minimum
9 $\quad x' \leftarrow$ `min`$(Q - x, \bar{u}(p_{\bar{\theta}}))$
10 $\quad f_{\bar{\theta}}(i;j) \leftarrow f_{\bar{\theta}}(i;j) + x' \ \forall (i;j) \in p_{\bar{\theta}}$
11 $\quad \pi_{\bar{\theta}}(i) \leftarrow \pi_{\bar{\theta}}(i) - d_{\bar{\theta}}(i) \ \forall i \in V$
12 $\quad \bar{c}_{\bar{\theta}}(i;j) \leftarrow c_{\bar{\theta}}(i;j) + \pi_{\bar{\theta}}(j) - \pi_{\bar{\theta}}(i) \ \forall (i;j) \in E$
13 $\quad \bar{u}_{\bar{\theta}}(i;j) \leftarrow \bar{u}_{\bar{\theta}}(i;j) - x' \ \forall (i;j) \in p_{\bar{\theta}}$
14 $\quad x \leftarrow x + x'$
15 $\quad$ **si** $x < Q$ **alors**
16 $\quad\quad p_{\bar{\theta}} \leftarrow$ `Dijkstra`$(G_{\bar{\theta}}, \bar{c}, s, t)$
17 $\quad\quad$ **si** $\bar{c}(p_{\bar{\theta}}) = \infty$ **alors**
18 $\quad\quad\quad$ `insérer`(Chemins, $p_{\bar{\theta}}, \infty$)
19 $\quad\quad$ **sinon**
20 $\quad\quad\quad$ `insérer`(Chemins, $p_{\bar{\theta}}, c(p_{\bar{\theta}})$)
21 $\quad\quad$ **fin**
22 $\quad$ **fin**
23 **fin**
24 **retourner** f

Dans le graphe résiduel, chacun de ces éléments dépend du temps, et est associé à un pas de temps θ donné.

DSSP commence par une phase d'initialisation qui consiste à calculer le chemin le moins cher p_θ de la source s au puits t dans chacun des t-graphes en utilisant Dijkstra puis à trier ces chemins dans l'ordre des couts. C'est ce qui est décrit entre les lignes 1 et 4 dans l'algorithme 1.

Dans chacun des t-graphes G_θ, on cherche le chemin le moins cher. Pour cela, on fait appel à la fonction `Dijkstra` qui renvoie un plus court chemin entre deux nœuds donnés dans un graphe en utilisant une fonction donnée pour la longueur des arcs. Ici, on utilise cette fonction pour calculer le chemin le moins cher de s à t dans G_θ en utilisant la fonction de couts c comme longueur d'arcs. Les arcs dont la capacité est nulle sont ignorés par `Dijkstra` et ne peuvent donc pas faire partie d'une solution.

Dès qu'un chemin est calculé, il est immédiatement inséré à la bonne place dans l'arbre binaire de recherche `Chemins` d'après son cout unitaire. Cette structure permet, dans la suite de l'algorithme, de récupérer efficacement le chemin le moins cher à emprunter, ce qui aura un impact positif sur la complexité de DSSP.

Après avoir initialisé à 0 la valeur du flot x et le flot f sur tous les arcs à tous les pas de temps, on peut commencer les itérations de l'algorithme (lignes 7 à 23).

Au début de chaque itération (ligne 8), on identifie le chemin le moins cher ($p_{\bar{\theta}}$) et donc le pas de temps le moins cher ($\bar{\theta}$). C'est au t-graphe correspondant à ce pas de temps que l'on va envoyer du flot. On souhaite envoyer le plus de flot possible sur ce chemin (ligne 9), ce qui équivaut soit à envoyer toute la quantité disponible qu'il reste ($Q-x$) soit à saturer le chemin $p_{\bar{\theta}}$ en envoyant du flot au maximum de sa capacité. La capacité résiduelle du chemin $p_{\bar{\theta}}$, notée ici $\bar{u}(p_{\bar{\theta}})$ est le minimum des capacités résiduelles de chaque arc qui compose le chemin. Cette valeur de flot à envoyer est notée x'.

Dans les lignes 10 à 14, sont mis à jour tous les éléments du graphe résiduel. Le flot f est mis à jour sur chaque arc du chemin emprunté (ligne 10). Le potentiel de chaque nœud est mis à jour de la même manière que dans l'algorithme SSP dans un graphe statique (ligne 11). Les couts réduits sont mis à jour pour chaque arc (ligne 12), et pas seulement pour les arcs sur lesquels a été envoyé du flot puisque les couts réduits dépendent des potentiels qui ont tous été mis à jour. Les capacités de chacun des arcs par lesquels du flot a été envoyé sont mises à jour (ligne 13). Et enfin la valeur du flot x est augmentée (ligne 14).

La fin d'une itération (lignes 16 à 21), qui consiste à calculer le nouveau plus court chemin dans le t-graphe étudié n'est réalisée que si une autre itération est nécessaire. Si toute la disponibilité a été envoyée, il n'est pas utile de réaliser cette étape. Le plus court chemin de s à t sur $G_{\bar{\theta}}$ mis à jour est calculé avec `Dijkstra` en utilisant les couts réduits comme longueur d'arc (ligne 16) puis inséré à la bonne place dans la structure `Chemins` qui contient chaque plus court chemin de chaque t-graphe en utilisant le cout réel du chemin et non pas son cout réduit. En effet, afin de comparer les couts des chemins entre différents t-graphes, il est nécessaire d'utiliser les couts réels. Si le cout réduit du plus court chemin trouvé dans $G_{\bar{\theta}}$ est l'infini, cela signifie qu'il n'existe pas de chemin de s à t dans $G_{\bar{\theta}}$, et donc le cout réel de ce chemin à insérer dans la structure `Chemins` est aussi l'infini car il n'est plus possible d'envoyer du flot à cette date et à l'avenir on ne doit plus choisir cette date.

Si la valeur du flot n'a pas encore atteint la valeur de la disponibilité Q alors il faut effectuer au moins une itération supplémentaire. Si toute la disponibilité a été envoyée, alors on peut renvoyer le flot f.

3.3.2.3 Exemple

Afin que l'algorithme DSSP décrit explicitement dans l'algorithme 1 soit parfaitement clair, observons sur un exemple toutes les étapes de son exécution. Ainsi, le cas choisi ici comme exemple d'exécution de l'algorithme est celui présenté en figure 3.13. Il s'agit d'un graphe de 4 nœuds, 5 arcs, avec un horizon de temps qui vaut 3. Chaque arc a une capacité et un cout, les deux variant au cours du temps. Dans cet exemple, 4 unités de flot sont à envoyer dans le graphe du nœud 1 vers le nœud 4.

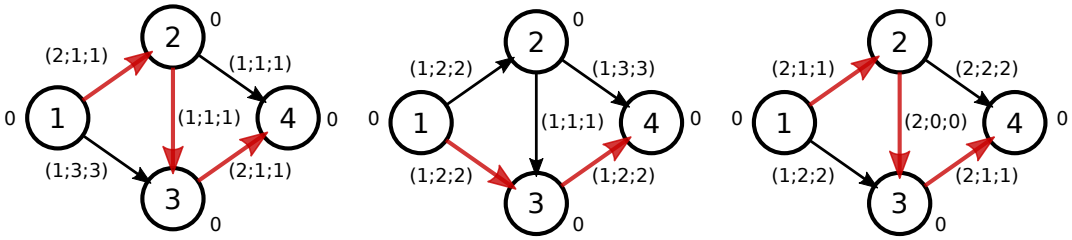
La figure 3.16 montre l'état du graphe résiduel à chaque étape de l'algorithme DSSP. Les capacités résiduelles et couts réduits des arcs sont notés à côté de ceux-ci et les potentiels des nœuds sont notés à côté des nœuds. Chaque plus court chemin est mis en évidence en rouge.

La phase d'initialisation de l'algorithme, décrite entre les lignes 1 et 4 de l'algorithme 1, est celle au cours de laquelle on calcule un premier plus court chemin dans chacun des t-graphes avec l'algorithme de Dijkstra puis classe les chemins obtenus.

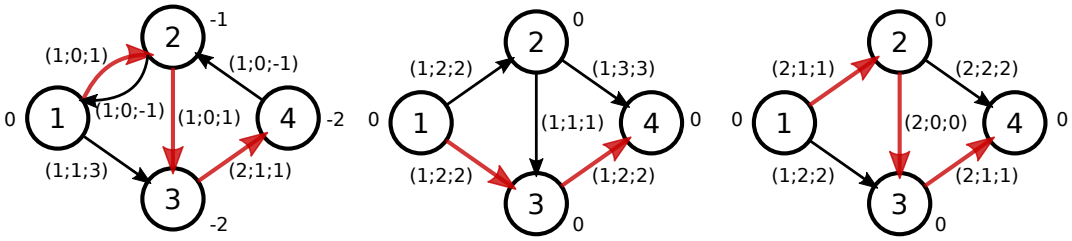
Dans notre exemple, le plus court chemin du premier t-graphe a un cout de 2, celui du deuxième t-graphe a un cout de 4 et celui du troisième t-graphe a un cout de 2 également. Ces chemins sont mis en évidence en rouge dans la figure 3.16a.

Dans chaque itération de l'algorithme, un chemin est choisi. Le chemin sélectionné est celui qui a le cout le plus faible parmi tous les chemins à disposition (un par t-graphe).

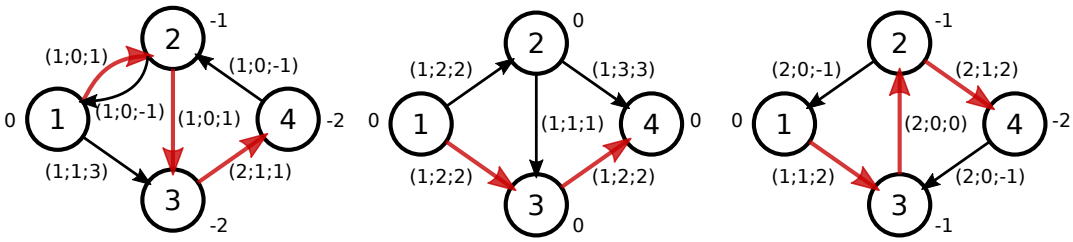
Pour la première itération de l'algorithme, le graphe résiduel sur lequel on effectue les calculs est exactement le graphe d'origine. Donc les couts réels des chemins sont égaux aux couts résiduels des chemins.



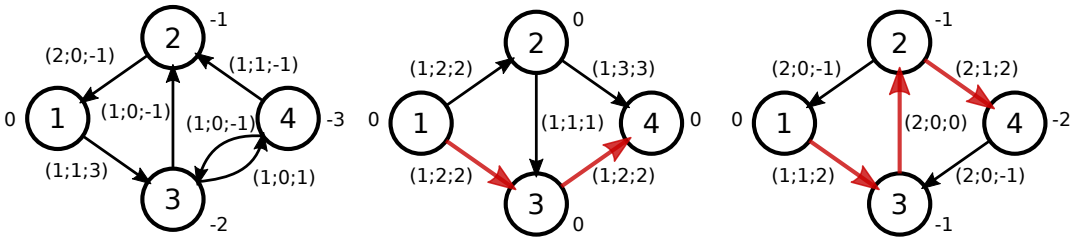
(a) Fin de la phase d'initialisation (lignes 1 à 4)



(b) Fin de la première itération



(c) Fin de la deuxième itération



(d) Fin de la troisième itération

Figure 3.16 : Évolution du graphe résiduel pendant l'exécution de l'algorithme 1 sur le graphe de la figure 3.13. Sur chaque arc $(i;j)$, le triplet $(u(i;j);\bar{c}(i;j);c(i;j))$ représente, respectivement, la capacité de l'arc $(i;j)$, le cout réduit de l'arc $(i;j)$ et le cout d'origine de l'arc $(i;j)$. Sur chaque nœud i, la valeur π_i représente le potentiel du nœud i. Sur chaque t-graphe résiduel, le plus court chemin obtenu grâce à l'algorithme de Dijkstra est représenté par des arcs en rouge.

Deux chemins ont un cout de valeur 2 (dans le premier et le troisième t-graphe), et un chemin a un cout de valeur 4 (dans le deuxième t-graphe). Nous devons donc choisir entre deux chemins qui ont un cout minimum. Dans cet exemple, le premier t-graphe est arbitrairement choisi.

Le chemin dans le graphe G_1 a une capacité de valeur 1. Donc une unité est envoyée sur ce chemin pour un cout de 2, ayant pour conséquence d'incrémenter le flot de 1, de mettre à jour les potentiels des nœuds, les couts réduits et les capacités résiduelles.

Il reste encore 3 unités de flot à envoyer dans le graphe, donc on cherche le nouveau plus court chemin dans le graphe résiduel G_1. Il a un cout réel de valeur 3. Ce chemin est mis en valeur en rouge dans la figure 3.16b qui montre aussi l'état du graphe à la fin de la première itération.

Dans la deuxième itération de l'algorithme, le chemin de G_1 a un cout de valeur 3, celui de G_2 a un cout de valeur 4 et celui de G_3 a un cout de valeur 2. Ce dernier est donc celui ayant le cout minimum. C'est sur ce chemin que des unités de flot vont être envoyées à cette itération.

La capacité du chemin $(1;2;3;4)$ dans G_3 est 2. Deux unités sont envoyées sur ce chemin pour un cout total de 4. Les capacités résiduelles, couts réduits et potentiels des nœuds sont mis à jour.

Trois unités ont été envoyées jusque là, l'algorithme n'est donc pas encore terminé. L'algorithme de Dijkstra est exécuté sur G_3 afin de trouver le nouveau plus court chemin, qui a un cout réel de valeur 4. L'état du graphe résiduel à la fin de cette itération est présenté en figure 3.16c.

Il ne reste plus qu'une unité de flot à envoyer dans le graphe donc cette troisième itération est forcément la dernière. Le cout réel du plus court chemin de G_1 est 3, celui de G_2 est 4 et celui de G_3 est 4. C'est donc dans le premier t-graphe que l'on va envoyer du flot.

La capacité du chemin $(1;2;3;4)$ dans G_1 est 1. Une unité est envoyée sur ce chemin pour un cout de 3. Le graphe résiduel, avec les potentiels des nœuds, les capacités résiduelles et les couts réduits, est mis à jour. Cette fois-ci, la quantité d'unités de flot déjà écoulée dans le graphe est égale à la quantité Q qui était à envoyer. Il n'y aura donc plus d'autre itération de l'algorithme, alors il n'est pas nécessaire de recalculer un plus court chemin dans ce t-graphe. La figure 3.16d montre l'état du graphe résiduel à la fin de cette troisième itération.

Au total, 2 unités de flot ont été envoyées au pas de temps 1, une première pour un cout de 2, une seconde pour un cout de 3, et 2 unités de flot ont été envoyées au pas de temps 3, toutes les deux pour un cout de 2. Il s'agit du flot de cout minimum de valeur 4.

Afin d'atteindre ce cout optimal de 9 pour envoyer 4 unités, l'algorithme n'a pas envoyé d'unités de flot au deuxième pas de temps qui était trop cher.

La figure 3.17 présente le graphe en mettant en évidence en rouge les arcs par lesquels passent des unités de flot. Le flot, la capacité et le cout des arcs est noté à coté de ceux-ci.

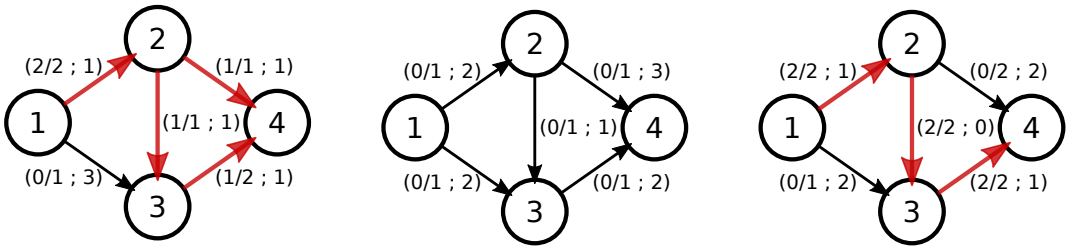


Figure 3.17 : Flot de cout minimum du graphe donné en figure 3.13. Sur chaque arc $(i;j)$, le couple $(f(i;j)/u(i;j);c(i;j))$ représente la valeur du flot sur l'arc $(i;j)$ sur la capacité totale de l'arc $(i;j)$ et le cout unitaire de l'arc $(i;j)$, respectivement. Les arcs apparaissant en rouge sont ceux sur lesquels passent des unités de flot (lorsque $f(i;j) \neq 0$).

3.3.2.4 Correction et complexité

Théorème 9 (Correction de DSSP)**.** *L'algorithme termine en un nombre fini d'itérations borné par Q. Le flot obtenu a pour valeur Q et son cout total est le cout total minimum.*

Démonstration. On rappelle au préalable que les capacités des arcs et les couts unitaires sont des entiers positifs ou nuls.

De plus, on suppose que la valeur du flot maximum du graphe est supérieure ou égal à Q. Dans le cas contraire, l'algorithme peut aisément être modifié pour détecter ce cas particulier. Nous le ne prenons donc pas en compte dans la preuve de correction.

Chaque itération de DSSP est en réalité une itération de SSP sur un t-graphe donné. Grâce aux propriétés de l'algorithme SSP, on sait que les capacités résiduelles des différents chemins trouvés successivement sont entières et positives. Le flot construit étape après étape est donc aussi entier. À chaque itération, un chemin est choisi et la valeur du flot courant est incrémentée d'au moins une unité.

L'algorithme peut s'arrêter dans deux cas. Dans le premier cas, la valeur du flot maximum a été atteinte par le flot courant dans chacun des t-graphes. Cela signifie que la quantité de flot Q que l'on souhaite envoyer ne peut pas être atteinte car elle est supérieure à la somme des flots maximums de chaque t-graphe. Nos hypothèses de départ ne considèrent pas ce cas. Le second cas d'arrêt de l'algorithme est lorsque la valeur Q est atteinte par la valeur x du flot courant.

De ce fait, l'algorithme termine en un nombre d'itérations fini et borné par Q.

Soit f^* un flot optimal tel que $f^* = (f_1^*, \ldots, f_\theta^*, \ldots; f_T^*)$ où f_θ^* est le flot pour le t-graphe G_θ. Soit le flot $f = (f_1, \ldots, f_\theta, \ldots, f_T)$ renvoyé par l'algorithme.

Ces deux flots sont de valeur Q. On peut décomposer ces flots selon leur valeur dans chaque t-graphe. De cette façon, on peut donner les vecteurs associés v^* et v, appelés vecteur de valeur de flot. On a alors $v^* = (v_1^*, \ldots, v_\theta^*, \ldots, v_T^*)$ et $v = (v_1, \ldots, v_\theta, \ldots, v_T)$ où v_θ^* et v_θ correspondent à la valeur des flots f_θ^* et f_θ respectivement.

Il est à noter que comme f^* est globalement optimal, alors chacun des f_θ^* est optimal sur G_θ pour la valeur v_θ^*. On peut donc considérer que chacun de ces flots peut être obtenu avec l'algorithme SSP.

Supposons d'abord que f^* et f aient le même vecteur de valeur de flot. C'est-à-dire $v^* = v$, et donc $\forall\theta\ v_\theta^* = v_\theta$. D'après SSP qui construit des flots de cout minimum à chaque itération, le cout associé à f est minimal sur chaque t-graphe et sa valeur est v_θ. On a alors que f et f^* sont identiques sur chaque t-graphe. Donc f est optimal.

Supposons à présent qu'il existe un pas de temps θ tel que $v_\theta^* > v_\theta$. Cela veut donc dire qu'il existe un pas de temps θ' tel que $v_{\theta'}^* < v_{\theta'}$. La figure 3.18 illustre par un graphique les valeurs prises par le flot et leurs couts telles que nous les considérons dans la suite.

Imaginons qu'on l'on utilise SSP sur G_θ. Les couts unitaires successifs des chemins calculés par l'algorithme sont $c_1 \leq \ldots \leq c_i \leq \ldots \leq c_j$, où c_i est le dernier cout unitaire calculé pour f et c_j est le dernier cout unitaire calculé pour f^*. De la même manière, les couts unitaires successifs pour $G_{\theta'}$ sont $c'_1 \leq \ldots \leq c'_k \leq \ldots \leq c'_l$, où c'_k est le dernier cout unitaire calculé pour f^* et c'_l est le dernier cout unitaire calculé pour f.

Supposons d'abord que $i = j$. Comme $v_\theta^* > v_\theta$, alors cela signifie que DSSP n'utilise pas toute la capacité du chemin correspondant. Cette situation est possible seulement si ce chemin est utilisé lors de la dernière itération de l'algorithme DSSP, c'est-à-dire lorsque la quantité Q a été envoyée. De ce fait, on a $c'_l \leq c_j$, et donc $c'_k \leq c_j$.

Supposons maintenant que $i < j$. Le chemin ayant pour cout unitaire c_{i+1} n'est jamais utilisé par DSSP. Cela veut dire que $c_{i+1} \geq c'_l$. On a donc $c_j \geq c_{i+1} \geq c'_l \geq c'_k$.

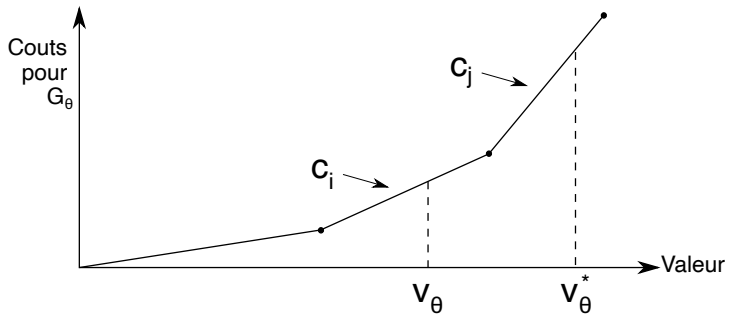


Figure 3.18 : Graphique montrant le cout du flot sur un t-graphe G_θ en fonction de la valeur de ce flot.

Donc dans les deux cas on a $c_j \geq c'_k$. Il est alors possible de retirer une unité de flot à f^*_θ et d'ajouter une unité de flot à $f^*_{\theta'}$. Le nouveau flot obtenu après cet échange a un cout au plus égal au cout du flot d'origine. Ce nouveau flot est donc optimal.

Cette opération d'échange peut être réalisée jusqu'à ce que les vecteurs v^* et v soient égaux. Donc f est de cout minimum et le résultat est prouvé. □

Théorème 10 (Complexité de DSSP)**.** *L'algorithme DSSP a une complexité au pire de* $O((Q+T)\cdot(m+n\cdot\log(n)+\log(T)))$.

Démonstration. Afin de sélectionner efficacement le plus court chemin à utiliser, nous les conservons classés selon leur cout réel en utilisant un arbre de recherche.

Lors de la phase d'initialisation (entre les lignes 1 et 4), on exécute une fois l'algorithme de Dijkstra sur chacun des T t-graphes ayant chacun n nœuds et m arcs. Les chemins obtenus sont ensuite classés et stockés dans l'arbre de recherche. La complexité de cette étape est donc $O(T\cdot(m+n\cdot\log(n))+T\cdot\log(T))$.

Observons maintenant la complexité d'une itération de l'algorithme décrite entre les lignes 7 et 23 de l'algorithme 1. Tout d'abord, on récupère à la ligne 8 le chemin le moins couteux parmi les T chemins. Cela se fait en $O(\log(T))$ grâce à l'arbre de recherche.

Ensuite, le graphe résiduel $G_{\bar{\theta}}$ est actualisé entre les lignes 10 et 14. Cette actualisation est faite en $O(n+m)$.

Une exécution de Dijkstra sur le t-graphe $G_{\bar{\theta}}$ à la ligne 16 se fait en $O(m+n\cdot\log(n))$. Ce nouveau chemin obtenu est inséré dans l'arbre de recherche en $O(\log(T))$. Au total, la complexité d'une itération de l'algorithme DSSP est $O(m+n\cdot\log(n)+\log(T))$.

Le nombre d'itérations de l'algorithme DSSP est borné par Q, comme annoncé par le théorème 9.

On obtient donc une complexité au pire de $O((Q+T)\cdot(m+n\cdot\log(n)+\log(T)))$ pour l'algorithme DSSP. □

Dans le cas où $T=1$, autrement dit si le graphe est statique, la complexité de l'algorithme DSSP vaut $O(Q\cdot(m+n\cdot\log(n)))$, ce qui est est exactement la complexité de l'algorithme SSP sur un graphe statique.

Comparons notre algorithme DSSP avec l'algorithme proposé par Nasrabadi et Hashemi (2010). On rappelle que l'algorithme qu'ils proposent a une complexité $O(B\cdot n\cdot T\cdot(n+T))$ avec B une borne supérieure de la disponibilité totale. Dans notre cas, la disponibilité totale vaut Q, on peut donc considérer, pour la comparaison, que $B=Q$. L'algorithme de Nasrabadi et Hashemi est en T^2 alors que DSSP

est en $T \cdot \log(T)$. Et leur algorithme est en n^2 alors que DSSP est en $n \cdot \log(n)$. Nous avons donc bien exploité le fait que notre modèle est restreint par rapport au leur qui est très général afin de proposer un algorithme plus efficace dans ce cas particulier.

On notera de plus que la complexité au pire de l'algorithme DSSP, que l'on vient de prouver, est théoriquement meilleure que la complexité au pire de l'algorithme SSP utilisé sur le graphe développé, décrit en section 3.3.1.3.

Les expérimentations réalisées, et présentées en section suivante, ont pour but d'observer le comportement réel des deux méthodes de résolution. Nous cherchons en particulier à voir si la différence d'ordre de grandeur entre les complexités au pire théoriques des deux méthodes se retrouve aussi dans le comportement pratique des algorithmes.

3.3.3 Extensions et limites

Nous avons présenté notre algorithme de résolution de flot de cout minimum pour un modèle particulier avec source unique et puits unique, où le stockage de flot sur les nœuds est interdit, où les temps de traversée sont nuls. Nous nous posons maintenant la question de savoir si on peut aller plus loin avec cet algorithme.

3.3.3.1 Sources et puits multiples

Le problème de flot de cout minimum sur lequel nous travaillons ici envoie des unités de flot depuis une source unique vers un puits unique. Comme nous l'avons dit lorsque le problème a été présenté, cela permet de simplifier le modèle.

Nous pouvons néanmoins imaginer une adaptation pour un modèle avec plusieurs sources et plusieurs puits. Pour cela, il serait nécessaire d'ajouter deux nœuds fictifs au graphe dynamique.

Le premier est une super-source reliée à toutes les sources du graphe par un arc dont la capacité est la disponibilité des sources en question. De la même manière, le second nœud fictif à ajouter est un super-puits, et tous les puits du graphe ont un arc sortant vers le super-puits dont la capacité est la demande des puits en question.

Adapter le modèle de graphe dynamique par l'ajout de nœuds fictifs n'est malheureusement pas suffisant pour résoudre le problème de flot de cout minimum pour le cas avec sources et puits multiples. Il faut également adapter l'algorithme de résolution DSSP. En effet, la disponibilité de chaque nœud étant à présent représentée par une capacité sur un arc, on ne peut pas exécuter l'algorithme de la même manière.

L'envoi d'une unité de flot depuis une source s^i vers un puits t^j à un pas de temps θ a pour effet de diminuer la capacité de l'arc allant de la super-source vers s^i et de l'arc allant de t^j vers le super-puits dans le t-graphe G_θ. On remarque que si l'on se contente de cela, alors l'information n'est pas propagée sur les autres t-graphes. L'algorithme serait alors incorrect puisqu'il deviendrait possible d'envoyer plus de flot depuis une source que sa disponibilité d'origine.

Pour pallier ce problème, il est nécessaire que chacun des t-graphes ait accès à l'information du flot envoyé dans les autres t-graphes. La première idée naturelle serait de mettre à jour la capacité des arcs partant de la super-source et arrivant au super-puits dans chaque t-graphe à chaque unité de flot envoyée. Cela impliquerait d'avoir une boucle en $O(T)$ dans chaque itération de l'algorithme. Cette solution serait beaucoup trop couteuse en terme de complexité de l'algorithme.

Pour ne pas impacter la complexité de l'algorithme, il faut alors conserver les disponibilités restantes des sources et les demandes restantes des puits dans une structure spécifique commune à tous les t-graphes. Il n'est alors plus nécessaire de mettre à jour les capacités de tous les t-graphes dans chaque itération, seule la structure globale est mise à jour.

3.3.3.2 Impossible extension de l'algorithme

Nous proposons un algorithme pour résoudre le problème de flot de cout minimum sur le modèle que nous avons présenté en section 3.3.1.1. D'après le théorème 10 que nous avons montré en section 3.3.2.4, l'algorithme DSSP a une complexité satisfaisante.

Il ne faut pas négliger l'impact du modèle sur la complexité de l'algorithme. En effet, notre modèle de graphe est restreint. Il n'y a ni stockage sur les nœuds, ni temps de traversée sur les arêtes. Ces éléments rendent les t-graphes indépendants les uns des autres, ce qui nous permet de les traiter séparément.

L'algorithme DSSP travaille sur les t-graphes indépendamment les uns des autres, de cette façon, les graphes sur lesquels on applique l'algorithme de Dijkstra sont plus petits. C'est grâce à cette particularité que l'on obtient un algorithme performant qui atteint une faible complexité.

Nous avons suivi une démarche de travail assez classique, qui consiste à nous intéresser à un modèle restreint afin d'en exploiter au mieux les caractéristiques. Et cela nous a effectivement permis de développer un algorithme efficace. La suite logique est d'essayer d'étendre cet algorithme à des modèles plus généraux.

Pour avoir un modèle de graphes dynamiques plus général, il faudrait autoriser le stockage sur les nœuds ou bien qu'il y ait un temps de traversée sur les arcs. On peut d'ailleurs noter qu'ajouter des temps de traversée sur les arcs permet également de modéliser un possible stockage de flot sur les nœuds. Il suffit pour cela d'avoir une boucle sur les nœuds dont le temps de traversée est 1.

Or, tout le fonctionnement et l'efficacité de notre algorithme reposent sur le fait que les t-graphes peuvent être traités séparément parce qu'ils sont indépendants. Mais si les unités de flot peuvent partir de la source à un pas de temps θ et arriver au puits à un pas de temps $\theta' \neq \theta$, alors les t-graphes ne sont pas indépendants. Et ce qu'il se passe sur l'un a un impact sur ce qu'il se passe sur les autres.

Une unité de flot qui arrive au puits à une date différente de celle à laquelle elle est partie de la source est la principale conséquence de temps de traversée non nuls ou de stockage sur les nœuds. Dans ce cas, il devient impossible de traiter les t-graphes séparément, et ce qui faisait le noyau principal de notre algorithme n'existe plus ici.

En résumé, on peut dire que l'algorithme que nous proposons est intéressant. En effet, l'étude théorique montre que sa complexité est significativement meilleure que la méthode de résolution traditionnelle par le graphe développé. De plus, nous montrons dans la suite qu'il est aussi performant en pratique. En revanche, cet algorithme ne permet pas d'aller plus loin dans la résolution les problèmes de flots de cout minimum car il ne peut pas être modifié afin d'être étendu à des modèles plus généraux.

3.3.4 Étude expérimentale

Nous évaluons l'algorithme que nous proposons en le comparant à la méthode que nous avons appelé SSP_{DVP} qui consiste à utiliser le graphe développé. Pour cela, nous avons créé un générateur de graphes dynamiques et implémenté les deux méthodes que nous allons comparer. Nous utilisons pour cela la librairie GraphStream [1], qui est une librairie Java. Pour de plus amples informations concernant GraphStream, voir le travail de Dutot et al. (2007).

Toutes les expériences ont été réalisées sur des machines possédant un processeur Intel Core 64 bits, ayant 8 cœurs, 2 GHz de fréquence, 4 Mo de mémoire cache et 96 Go de RAM. L'environnement d'exécution Java utilisé est OpenJDK 1.8.

1. http ://grahpstream-project.org

3.3.4.1 Générateur de graphes dynamiques

La première étape avant de pouvoir tester un algorithme sur des graphes dynamiques est de pouvoir être en mesure de générer aléatoirement des graphes dynamiques ayant les propriétés dont nous avons besoin. C'est la raison pour laquelle nous avons créé un générateur de graphes dynamiques, basé sur l'un des générateur de graphes de GraphStream.

Pour générer un graphe dynamique, on commence par générer le graphe sous-jacent, c'est à dire l'ensemble des nœuds du graphe, et l'ensemble des arcs qui seront présents à un moment dans le graphe. Nous ajoutons ensuite, pour chaque arc, un vecteur de capacité, qui correspond aux capacités successives de l'arc. En autorisant la capacité à être nulle, on simule l'absence d'un arc. De la même manière, sur chaque arc est ajouté un vecteur de couts, qui correspond aux couts successifs de l'arc tout au long de l'intervalle d'étude du graphe. Une fois le graphe dynamique obtenu, un nœud est désigné pour être la source, et un autre nœud est désigné pour être le puits.

Pour créer notre générateur de graphes dynamiques, nous nous sommes basé sur le générateur de graphes statiques *Random Euclidean Generator* de la librairie GraphStream. Ce générateur crée des graphes aléatoires géométriques (qui sont définis dans la section 4.4.1 page 108). Ce type de graphes est particulièrement approprié pour les applications où la position spatiale des nœuds est importante et où les liens existants entre les nœuds sont étroitement liés à la position de ces derniers.

Les nœuds sont placés de façon aléatoire sur un espace fini à deux dimensions $[0;1] \times [0;1]$. Les arcs sont présents selon une valeur de seuil (*threshold* en anglais). Ce seuil est caractérisé par un paramètre th. Deux nœuds sont connectés directement dans le graphe s'ils sont suffisamment proches, c'est-à-dire si la distance euclidienne qui les sépare est plus petite que le seuil th. Une fois la connexion faite entre deux nœuds, elle est aléatoirement orientée dans un sens ou dans l'autre pour obtenir un arc du graphe.

Avec un tel générateur, on choisit le nombre de nœuds du graphe et le seuil. On ne peut pas choisir à l'avance le nombre d'arc qu'aura le graphe généré. En revanche, le seuil th a évidemment un impact direct sur la densité du graphe, qui correspond au nombre d'arcs du graphe sur le nombre d'arcs potentiels. Plus le seuil est grand, plus les nœuds auront de voisins et donc plus le graphe sera dense. De cette manière, on peut choisir le nombre de nœuds que l'on souhaite dans notre graphe dynamique et indirectement choisir sa densité. On obtient ainsi une estimation du nombre d'arcs.

Nous avons réalisé des expériences afin de déterminer au mieux le lien entre seuil et densité. Nous avons pour cela construit de multiples graphes aléatoirement en utilisant le Random Euclidean Generator de GraphStream avec différentes valeurs de seuil. Les résultats de ces expériences nous ont permis de déterminer qu'un seuil de $0,08$ donnait approximativement 1% de densité, un seuil de $0,19$ donnait environ 5% de densité et un seuil de $0,44$ donnait environ 20% de densité. Cette correspondance entre seuil et densité est rappelée dans le tableau 3.2.

La densité du graphe est déterminée seulement par le seuil donné au générateur. En effet, le nombre de nœuds du graphe n'impacte pas sa densité.

Seuil th	Densité du graphe
$0,08$	1%
$0,19$	5%
$0,44$	20%

Table 3.2 : Correspondance entre seuil th et densité du graphe avec le générateur Random Euclidean Generator. Valeurs obtenues expérimentalement.

Nous travaillons sur un problème de flot de cout minimum. Nous considérons le cas où la source du graphe est unique, tout comme le puits. Nous devons identifier deux nœuds dans le graphe dynamique généré, un pour la source et un pour le puits.

Nous avons décidé de ne pas les choisir de manière aléatoire, mais de laisser le choix à l'utilisateur du générateur de graphe dynamique que nous créons. Notre générateur prend donc en paramètre les coordonnées de deux points, correspondant à la source et au puits souhaités pour le graphe. Le nœud source et le nœud puits du graphe dynamique sont donc ceux dont la position est la plus proche de celle souhaitée.

Une fois le graphe sous-jacent créé, ce sont les capacités et les couts, tous deux variables, qui font la dynamicité. Nous représentons cela par un vecteur de taille T. Chaque arc possède donc un vecteur cout de taille T et un vecteur capacité de taille T. La i^{me} valeur du vecteur de cout ou de capacité correspond respectivement au cout ou à la capacité de l'arc au pas de temps i.

Nos vecteurs de cout et de capacité sont générés suivant le même procédé, à savoir des valeurs qui varient dans un intervalle donné en fonction de paramètres donnés. Pour cela on donne la borne inférieure min et la borne supérieure max de l'intervalle. Aucune des valeurs du vecteur ne pourra être inférieure à min ou supérieure à max.

Ces valeurs sont positives puisqu'elles correspondent à une capacité d'arc ou à un cout unitaire. Dans le cas des capacités, si la borne inférieure vaut 0, cela signifie qu'un arc peut avoir une capacité nulle auquel cas il ne peut pas être emprunté, donc c'est comme s'il n'était pas présent au pas de temps concerné.

Afin d'avoir un graphe qui évolue dans le temps de façon réaliste, nous avons souhaité que les t-graphes ne soient pas complètement indépendants les uns des autres, et donc que les capacités et les couts ne soient pas décidés de façon aléatoire pour chaque pas de temps. La valeur de capacité et de cout pour un arc donné à un pas de temps donné doit dépendre de sa valeur au pas de temps précédent.

La première valeur du vecteur (correspondant au premier pas de temps) est sélectionnée aléatoirement dans l'intervalle $[min, max]$ défini par les bornes. À chaque pas de temps, la valeur peut être incrémentée, décrémentée ou bien peut rester inchangée au pas de temps suivant. Cela est défini par des probabilités. La valeur est incrémentée avec la probabilité p_{inc} et est décrémentée avec la probabilité p_{dec}. Et donc, trivialement, elle reste inchangée avec la probabilité $1 - p_{inc} - p_{dec}$.

Lorsque qu'une valeur vaut min, elle est incrémentée au pas de temps suivant avec la probabilité p_{inc} et reste inchangée sinon. De la même manière, lorsque d'une valeur vaut max, elle est décrémentée au pas de temps suivant avec la probabilité p_{dec} et reste inchangée sinon.

Incrémenter (ou décrémenter) une valeur par un certain pourcentage de celle-ci n'est pas une méthode de variation intéressante. Nous souhaitions avoir la possibilité d'avoir des arcs absents et ils sont représentés par une capacité nulle. Nous souhaitions aussi avoir la possibilité d'avoir des arcs sur lesquels du flot pourraient transiter gratuitement, ce qui se modélise par un cout unitaire nul. Or, si la borne inférieure choisie min est 0, alors pour un horizon de temps T suffisamment lointain, les valeurs tendent vers 0, qui est une valeur absorbante.

Nous avons donc fait le choix d'incrémenter et de décrémenter les valeurs d'une quantité fixe. Nous introduisons pour cela deux nouveaux paramètres : inc et dec. Ils correspondent respectivement à la quantité ajoutée lors d'une incrémentation et à la quantité retranchée lors d'une décrémentation.

En résumé, les valeurs sont incrémentées de inc avec la probabilité p_{inc}, et donc décrémentées de dec avec la probabilité p_{dec}.

On rappelle ci-dessous tous les paramètres utilisés pour la création des vecteurs de couts et de capacités.

min	:	borne inférieure des valeur du vecteur
max	:	borne supérieure des valeur du vecteur
inc	:	quantité ajoutée en cas d'incrémentation
p_{inc}	:	probabilité d'incrémenter la valeur
dec	:	quantité retranchée en cas de décrémentation
p_{dec}	:	probabilité de décrémenter la valeur

3.3.4.2 Expériences préliminaires

Afin d'évaluer la qualité notre algorithme DSSP, mais aussi l'impact de certains paramètres sur cet algorithme, nous avons mené une étude expérimentale. Pour définir le plan de cette étude, nous avons d'abord réalisé un certain nombre d'expériences préliminaires nous permettant de déterminer les paramètres à tester et les expériences à réaliser.

Il a rapidement été décidé de tester l'efficacité de notre algorithme par rapport à la taille des graphes dynamiques sur lesquels il est exécuté. C'est pourquoi nous faisons deux types d'expériences, une première où on observe le temps de calcul en fonction du nombre de nœuds du graphe, et une seconde où on observe le temps de calcul en fonction de l'horizon de temps.

Nous avons d'abord eu à décider de la densité des graphes et ensuite de la quantité de flot à envoyer. C'est sur ces points que portent nos expériences préliminaires.

La création des graphes dynamiques et des graphes développés ainsi que leur stockage en mémoire afin d'effectuer des algorithmes dessus demandent énormément de ressources. Malgré des machines performantes, il nous était impossible de traiter des graphes de trop grande taille.

Afin de déterminer nos limites de calcul, nous avons réalisé une expérience. Le but était de voir jusqu'où nous pouvions aller en termes de dimensions de graphes sur lesquels nos algorithmes s'effectuent. Pour cela, nous avons construit des graphes dynamiques de tailles différentes (en variant le nombre de nœuds n), de durées différentes (en variant l'horizon de temps T), et de densités différentes (en variant le seuil th pour la création des arcs), puis nous y avons appliqué les algorithmes de calcul de flot de cout minimum que nous souhaitons comparer (le nôtre DSSP, et la méthode classique SSP_{DVP}).

Dans le tableau 3.3, le symbole « ✓ » indique qu'un graphe dynamique de durée T donnée sur la ligne, de taille n donnée sur la colonne, et ayant 1% de densité peut être traité par nos machines. Par exemple, nous pouvons construire un graphe dynamique de 1000 nœuds, 2000 pas de temps, et 1% de densité, ainsi que le graphe développé correspondant, et nous pouvons calculer le flot de cout minimum en utilisant notre algorithme et la méthode classique. En revanche, nous ne pouvons pas le faire pour un graphe de 2000 nœuds, 1000 pas de temps et 1% de densité.

Les tableaux 3.4 et 3.5 montrent les instances que nous pouvons calculer pour les densités 5% et 20% respectivement. Plus les graphes sont denses, plus cela demande de ressources, ce qui est tout à fait logique puisqu'ils ont beaucoup plus d'arcs. En comparant les tableaux 3.3, 3.4 et 3.5, cela confirme donc qu'avec un seuil th plus grand, on ne peut pas traiter des graphes d'aussi grande taille qu'avec un seuil plus petit.

Nos seconde expérience préliminaire porte sur la quantité de flot à envoyer dans le graphe dynamique. L'idée était d'envoyer une quantité conséquente de flot dans le graphe dynamique afin d'avoir un nombre important d'itérations de DSSP. Mais nous ne voulions pas envoyer trop d'unités de flot car si tous les chemins ou presque entre la source et le puits sont saturés alors il y aura nécessairement moins de choix à faire au cours de l'exécution de l'algorithme. C'est pourquoi nous avons souhaité définir la quantité Q comme un pourcentage du flot maximum.

Nous avons pour cela dû faire une estimation de ce flot maximum. Il est évident que nous n'allons pas calculer le flot maximum avant d'exécuter l'algorithme de flot de cout minimum en envoyant un

	50	100	500	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
10	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
50	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
100	✓	✓	✓	✓	✓	✓	✓	✓					
500	✓	✓	✓	✓	✓								
1000	✓	✓	✓	✓									
2000	✓	✓	✓	✓									
3000	✓	✓	✓	✓									
4000	✓	✓	✓										
5000	✓	✓	✓										
6000	✓	✓	✓										
7000	✓	✓	✓										
8000	✓	✓	✓										
9000	✓	✓	✓										
10000	✓	✓	✓										

Table 3.3 : Instances pour lesquelles le graphe dynamique et le graphe développé ont pu être construit et les deux algorithmes exécutés avec $th = 0,08$, soit 1% de densité. En lignes, l'horizon de temps. En colonnes, le nombre de nœuds.

	50	100	500	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
10	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
50	✓	✓	✓	✓	✓	✓							
100	✓	✓	✓	✓	✓								
500	✓	✓	✓	✓									
1000	✓	✓	✓										
2000	✓	✓	✓										
3000	✓	✓	✓										
4000	✓	✓											
5000	✓	✓											
6000	✓	✓											
7000	✓	✓											
8000	✓	✓											
9000	✓	✓											
10000	✓	✓											

Table 3.4 : Instances pour lesquelles le graphe dynamique et le graphe développé ont pu être construit et les deux algorithmes exécutés avec $th = 0,19$, soit 5% de densité. En lignes, l'horizon de temps. En colonnes, le nombre de nœuds.

	50	100	500	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
10	✓	✓	✓	✓	✓	✓							
50	✓	✓	✓	✓									
100	✓	✓	✓	✓									
500	✓	✓	✓										
1000	✓	✓											
2000	✓	✓											
3000	✓	✓											
4000	✓	✓											
5000	✓	✓											
6000	✓	✓											
7000	✓	✓											
8000	✓	✓											
9000	✓	✓											
10000	✓	✓											

Table 3.5 : Instances pour lesquelles le graphe dynamique et le graphe développé ont pu être construit et les deux algorithmes exécutés avec $th = 0,44$, soit 20% de densité. En lignes, l'horizon de temps. En colonnes, le nombre de nœuds.

pourcentage du flot maximum exact, cela prendrait beaucoup trop de temps, d'où la nécessaire estimation.

Nous avons construit de multiples graphes avec un pas de temps ($T = 1$), et en variant le nombre de nœuds n et le seuil th, en suivant la méthode expliquée en section 3.3.4.1. Nous avons calculé le flot maximum sur ces graphes. Nous avons répété l'opération 100 fois pour chaque jeu de paramètres. Nous avons conservé la valeur médiane de flot maximum comme une estimation du flot maximum attendu pour un unique pas de temps.

Puisque les t-graphes de nos graphes dynamiques sont indépendants (comme il n'y a ni stockage, ni temps de traversée des arcs), et qu'ils ont les mêmes caractéristiques de par leur construction, on peut alors dire que le flot maximum sur T pas de temps est proche de T fois le flot maximum sur un pas de temps. Dans ce cas, en multipliant par T la valeur médiane de flot maximum pour un pas de temps, on obtient une estimation du flot maximum pour les graphes dynamiques que nous construisons.

Les figures 3.19, 3.20 et 3.21 montrent la valeur médiane du flot maximum en fonction du nombre de nœuds du graphe pour des graphes ayant un pas de temps et respectivement 1%, 5% et 20% de densité.

Sur ces graphiques, nous avons représenté une fonction de régression linéaire en n, calculée sur les valeurs de flot maximum obtenues expérimentalement. Il s'agit d'une régression linéaire qui, comme toutes celles que nous utilisons tout au long de ce manuscrit, est calculée avec la méthode des moindres carrés. Ces régressions permettent de valider l'observation selon laquelle l'augmentation du flot maximum est proportionnelle au nombre de nœuds du graphe. Afin d'évaluer la qualité de cette régression, autrement dit afin d'estimer précisément à quel point elles sont fidèles aux résultats expérimentaux obtenus, nous calculons pour chaque régression sa valeur R^2. Cette valeur R^2 se calcule de la façon suivante :

$$R^2 = 1 - \frac{\sum_{i=1}^{k}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{k}(y_i - \bar{y})^2} \tag{3.21}$$

Dans cette expression, k est la nombre de valeurs sur lesquelles la régression a été calculée, y_i est la $i^{\text{ème}}$ valeur expérimentale, $\hat{y}_i$ est la $i^{\text{ème}}$ valeur prédite avec la fonction de régression et $\bar{y}$ est la valeur moyenne. La valeur est comprise dans l'intervalle $[0;1]$. Plus la valeur R^2 est proche de 1, meilleure est la régression.

Pour les graphes de densité 1%, la valeur $R^2 = 0,966$. Pour les graphes de densité 5%, la valeur $R^2 = 0,994$. Pour les graphes de densité 20%, la valeur $R^2 = 0,998$. Ces valeurs, toutes très proches de 1, indiquent que la régression linéaire est parfaitement adaptée à ces données, et donc que le flot maximum dépend linéairement du nombre de nœuds du graphe.

En résumé, le flot maximum est par construction linéaire en T, et l'expérience a montré qu'il était aussi linéaire en n. De ce fait, la quantité de flot Q à envoyer dans le graphe, qui est un pourcentage du flot maximum estimé, est de l'ordre de $O(T)$ et de $O(n)$.

3.3.4.3 Paramètres d'expérience

Nous expliquons dans cette section l'étude expérimentale que nous avons menée. Nous réalisons deux expériences, une première dans laquelle nous faisons varier le nombre de pas de temps du graphe, et une seconde où nous faisons varier le nombre de nœuds du graphe. Nous étudions ensuite les résultats obtenus.

Il nous faut construire des graphes dynamiques aléatoirement de la manière détaillée dans la section 3.3.4.1. Afin de construire ces graphes, nous avons dû choisir comment fixer les différents paramètres.

Pour chacune des deux expériences, nous avons choisi de fixer le seuil th à $0,08$, ce qui nous donne des graphes avec 1% de densité. Cette densité de graphe nous permet d'avoir suffisamment de puissance de calcul pour exécuter notre algorithme sur une plus grande variété de nombre de pas de temps et de nombre de nœuds.

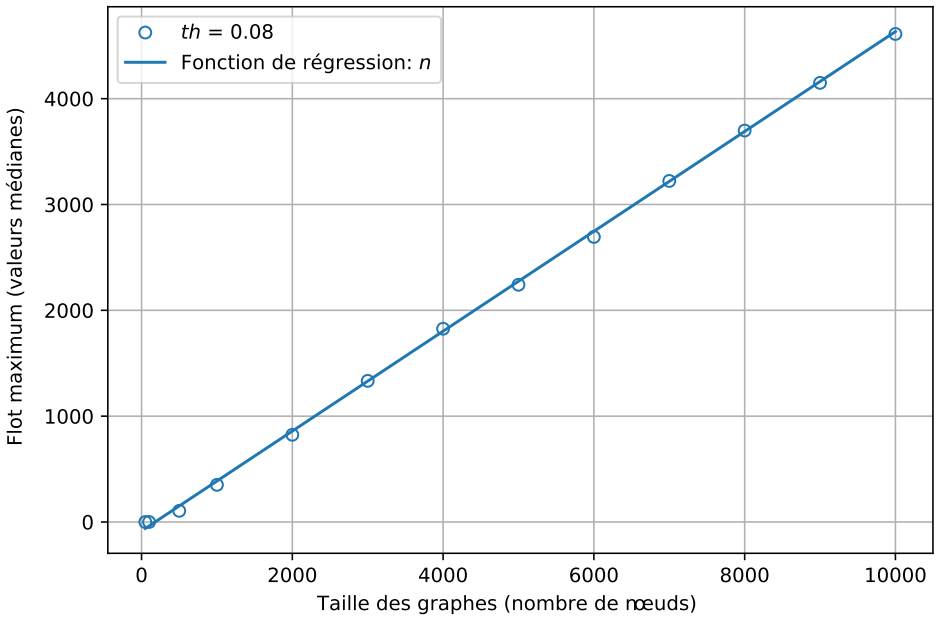


Figure 3.19 : Valeur du flot maximum pour $T = 1$ et $th = 0,08$ (soit 1% de densité) en fonction du nombre de nœuds du graphe. Fonction de régression linaire avec $R^2 = 0,966$. Les capacités ont été générées comme indiqué dans le tableau 3.7

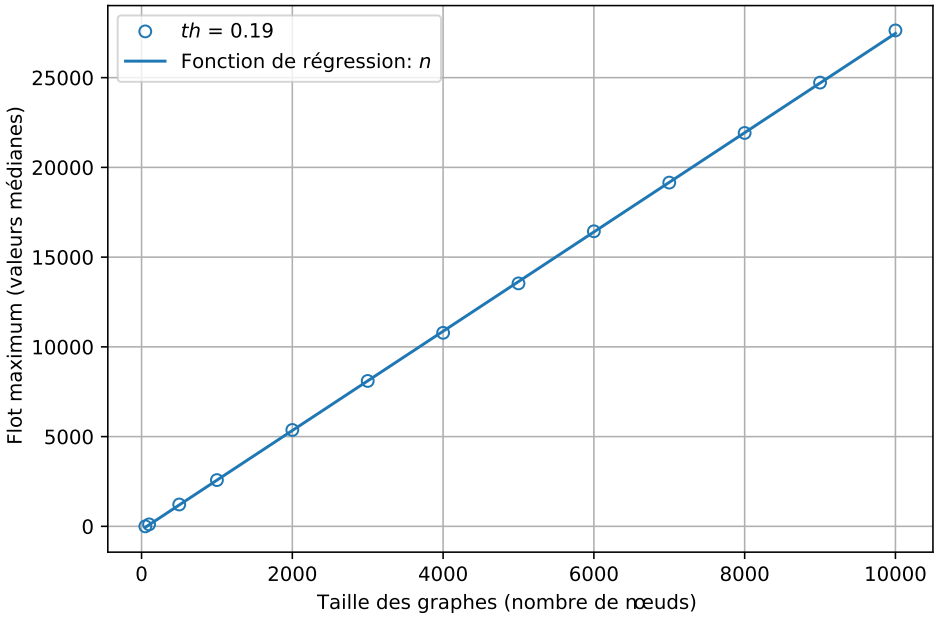


Figure 3.20 : Valeur du flot maximum pour $T = 1$ et $th = 0,19$ (soit 5% de densité) en fonction du nombre de nœuds du graphe. Fonction de régression linaire avec $R^2 = 0,994$. Les capacités ont été générées comme indiqué dans le tableau 3.7

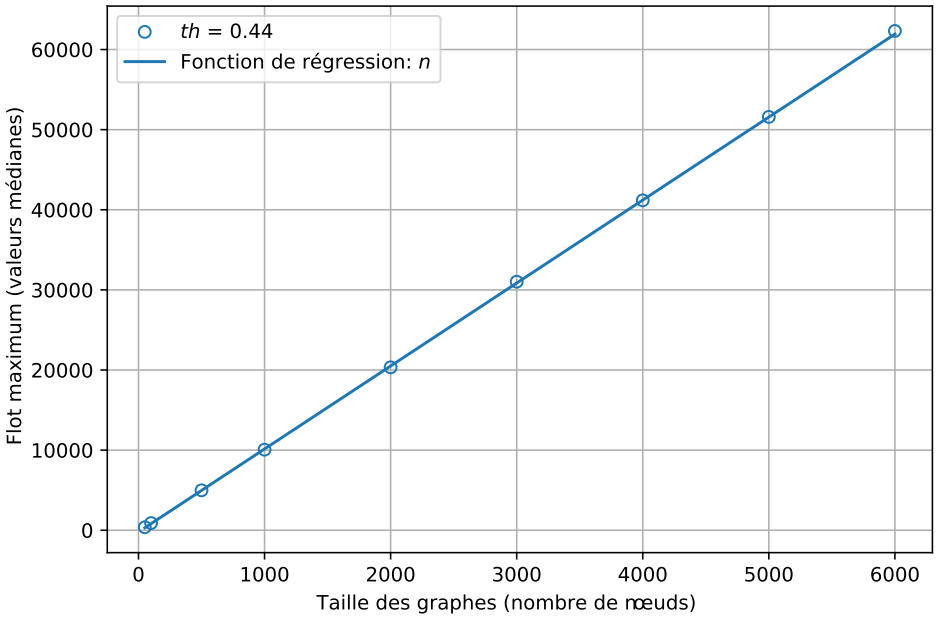


Figure 3.21 : Valeur du flot maximum pour $T = 1$ et $th = 0,44$ (soit 20% de densité) en fonction du nombre de nœuds du graphe. Fonction de régression linaire avec $R^2 = 0,998$. Les capacités ont été générées comme indiqué dans le tableau 3.7

Nous avons choisi de situer la source du graphe à la position $(0,25;0,75)$ et de situer le puits du graphe à la position $(0,75;0,25)$.

Nous avons choisi d'envoyer dans le graphe 80% du flot maximum estimé. Avoir une grande quantité de flot à envoyer dans le graphe assure que l'algorithme exécute un nombre important d'itérations avant d'atteindre la solution optimale.

Pour la première expérience, le nombre de nœuds n est fixé à 500 et l'horizon de temps T varie entre 10 et 1000. Pour la seconde expérience, l'horizon de temps est cette fois-ci fixé à 100 et le nombre de nœuds varie de 500 à 2000.

Le tableau 3.6 récapitule les paramètres d'expérience fixés.

	Expérience sur T	Expérience sur n
n	500	Dans $[500;2000]$
T	Dans $[10;1000]$	100
th	$0,08$	$0,08$
Q	80% du flot max estimé	80% du flot max estimé
Position de la source	$(0,25;0,75)$	$(0,25;0,75)$
Position du puits	$(0,75;0,25)$	$(0,75;0,25)$

Table 3.6 : Paramètres utilisés pour générer aléatoirement les instances de flot de cout minimum pour l'étude expérimentale

Afin de générer les couts et les capacités des arcs, nous avons utilisé la méthode décrite en section 3.3.4.1. Les couts, comme les capacités, peuvent varier de 0 à 100. Lorsqu'une valeur a été générée pour un pas de temps donnée, elle augmente de 10 avec une probabilité de 25% et diminue de 10 avec une probabilité de 25%. Elle a trivialement une chance sur deux de ne pas être modifiée.

Ces paramètres de génération permettent d'avoir des capacités nulles. Cela signifie que certains arcs peuvent devenir indisponibles à certains pas de temps. Ces paramètres permettent également d'avoir des couts unitaires nuls, ce qui signifie que l'envoi de flot sur certains arcs peut être gratuit à certains pas de temps.

Ces paramètres assurent aussi que les t-graphes qui composent le graphe dynamique ne sont pas totalement indépendants les uns des autres, en effet, chaque t-graphe évolue pour donner le t-graphe suivant. Le graphe dynamique obtenu est quand même très dynamique, au sens où il évolue énormément.

Le tableau 3.7 résume les paramètres choisis pour la génération des couts et des capacités sur les arcs du graphe dynamique.

	Vecteurs de capacités	Vecteurs de couts
min	0	0
max	100	100
inc	10	10
p_{inc}	0,25	0,25
dec	10	10
p_{dec}	0,25	0,25

Table 3.7 : Paramètres utilisés pour générer les vecteurs de capacités et de couts

Pour chaque jeu de paramètres, l'expérience se déroule de la même façon. Le graphe dynamique est d'abord généré aléatoirement grâce à notre générateur. L'algorithme DSSP est exécuté sur ce graphe et on mesure le temps d'exécution. On crée ensuite le graphe développé correspondant au graphe dynamique généré puis on y applique l'algorithme SSP en mesurant le temps de calcul. Ces étapes sont répétées 15 fois pour chaque jeu de paramètres. Cela nous permet d'obtenir le temps de calcul moyen et le temps de calcul médian de chaque méthode de résolution du flot de cout minimum.

Notre but étant de comparer les deux méthodes, nous calculons aussi des fonctions de régressions en utilisant les temps de calcul médians. La forme de ces fonctions est définie à partir du comportement théorique. Nous évaluons ces fonctions de régression avec la valeur R^2, décrite par l'expression (3.21) (page 70). On rappelle que, dans cette expression, k est la nombre de valeurs sur lesquelles la régression a été calculée, ce qui correspond, dans le cas présent, au nombre de jeux de paramètres (nombre de pas de temps dans la première expérience, nombre de nœuds du graphe dans la seconde expérience).

3.3.4.4 Expérience en fonction de T

Pour cette expérience, nous souhaitons observer le temps de calcul de l'algorithme DSSP, décrit dans l'algorithme 1, en fonction de l'horizon de temps T. C'est la raison pour laquelle nous faisons varier T et fixons tous les autres paramètres. Ici nous avons fixé le nombre de nœuds n à 500, et faisons varier

n	500
T	{10;50;100;200;300;400; 500;600;700;800;900;1000}
th	0,08
Q	80% du flot max estimé
Position de la source	(0,25;0,75)
Position du puits	(0,75;0,25)

Table 3.8 : Paramètres de l'expérience faisant varier T

T entre 10 et 1000 de la sorte : $T \in \{10 ; 50 ; 100 ; 200 ; 300 ; 400 ; 500 ; 600 ; 700 ; 800 ; 900 ; 1000\}$. Tous les paramètres de cette expérience sont rappelés dans le tableau 3.8.

Notre algorithme DSSP est comparé à l'algorithme SSP sur le graphe développé. Cette méthode, que nous nommons SSP_{DVP}, a une complexité en temps de calcul au pire de $O(Q \cdot T \cdot (m + n \cdot \log(n \cdot T)))$. Cette écriture peut être grandement simplifié puisque dans le cas présent, beaucoup de paramètres sont fixés. En effet, la quantité Q est linéaire en T, n est fixé, et puisque le paramètre th est fixé aussi alors m est constant. En prenant en compte ces éléments dans l'écriture de la complexité, on obtient alors $O(T^2 \cdot \log(T))$.

Notre algorithme DSSP a une complexité au pire en temps de calcul de $O((Q+T) \cdot (m + n \cdot \log(n) + \log(T)))$. En simplifiant l'écriture de la complexité de la même manière que précédemment, on obtient alors pour l'expérience présente $O(T \cdot \log(T))$.

Afin de comparer aisément ces complexités théoriques, elles sont rappelées dans le tableau 3.9. Une métrique intéressante à observer est le ratio de complexité entre la méthode SSP_{DVP}, sur le graphe développé, et l'algorithme DSSP, qui travaille directement sur le graphe dynamique. Ce ratio vaut T.

Nous nous attendons donc à retrouver ce ratio théorique linéaire en comparant les temps de calcul en pratique entre les deux méthodes.

Complexité	Cas général	Dans l'expérience présente
SSP_{DVP}	$O(Q \cdot T \cdot (m + n \cdot \log(n \cdot T)))$	$O(T^2 \cdot \log(T))$
$DSSP$	$O((Q+T) \cdot (m + n \cdot \log(n) + \log(T)))$	$O(T \cdot \log(T))$

Table 3.9 : Complexités théoriques des deux algorithmes comparés lorsque T est fixé

Les résultats auxquels nous nous intéressons ici sont les temps de calcul des différentes méthodes. Ils sont présentés sous forme de graphiques dans les figures 3.22, 3.23 et 3.24. Pour chaque valeur de T, le graphique montre les valeurs moyennes et médianes obtenues pour la série d'instances testées. Une courbe de régression est aussi tracée sur ces graphiques permettant de comparer les résultats expérimentaux avec les valeurs théoriques attendues. Afin d'évaluer la qualité de cette régression, nous donnons les valeurs R^2, qui sont aussi rassemblées dans le tableau 3.10.

La figure 3.22 montre le temps de calcul moyen et médian, en secondes, de la méthode SSP_{DVP} lorsque l'on fait varier T. Une fonction de régression de la forme $T^2 \cdot \log(T)$, comme la complexité théorique, est tracée. La valeur R^2 de cette régression est $0,99$. Pour citer quelques chiffres, on note que le calcul est quasiment instantané lorsque le graphe dynamique n'a que 10 pas de temps alors que lorsqu'il en a 1000, le calcul prend jusqu'à 20000 secondes, soit presque 6 heures.

La figure 3.23 montre le temps de calcul moyen et médian, exprimé en secondes, de l'algorithme DSSP en fonction de la valeur de T. La fonction de régression, dont la forme $T \cdot \log(T)$ a été déterminée grâce à la complexité en temps théorique, est tracée sur le graphique. La valeur R^2 de cette régression est $0,99$. Exécuter l'algorithme DSSP sur un graphe dynamique ne prend pas plus d'une seconde lorsque le graphe a 10 pas de temps et lorsque le graphe a 1000 pas de temps, l'exécution de l'algorithme prend moins de 20 secondes.

	R^2
SSP_{DVP}	$0,99$
$DSSP$	$0,99$
Ratio	$0,94$

Table 3.10 : Valeur R^2 des courbes de régressions sur les temps de calcul pour l'expérience faisant varier l'horizon de temps T.

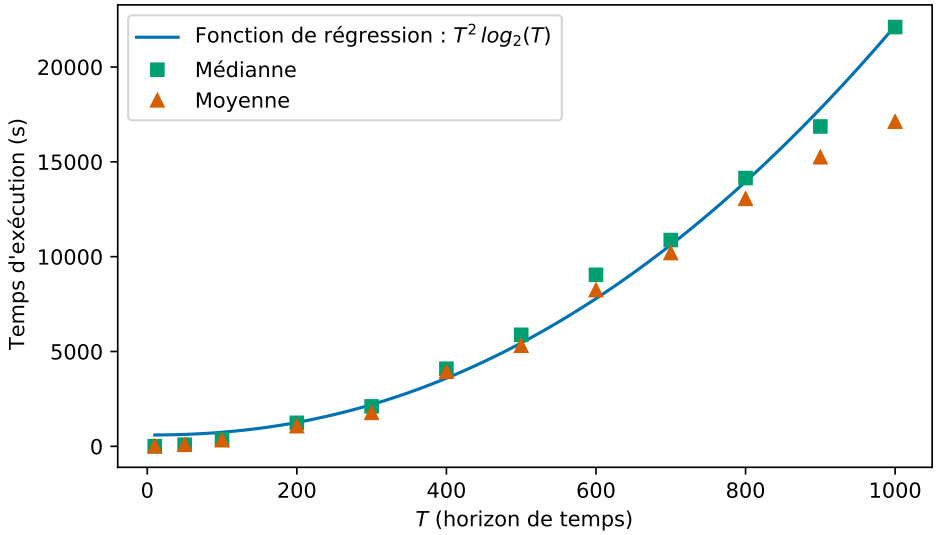


Figure 3.22 : Temps d'exécution de SSP_{DVP} en fonction de T, exprimé en secondes.

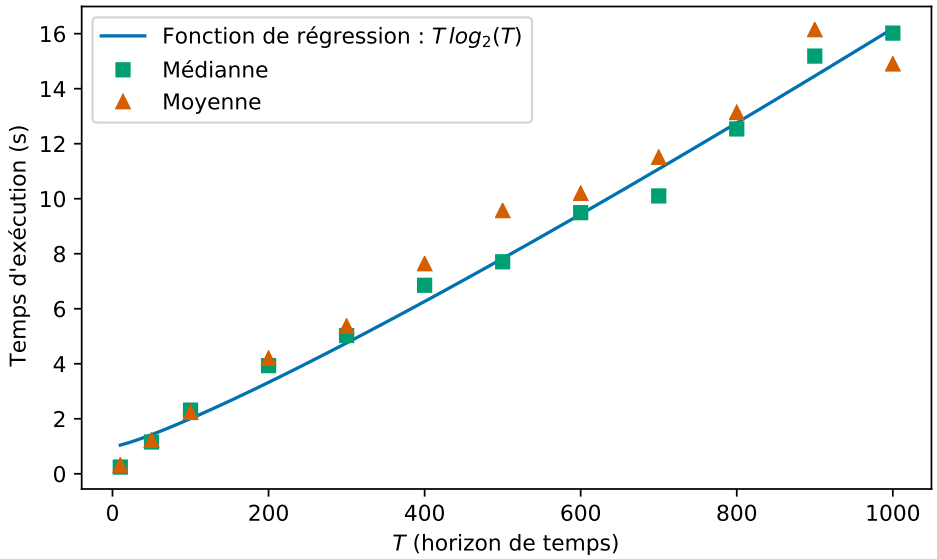


Figure 3.23 : Temps d'exécution de $DSSP$ en fonction de T, exprimé en secondes.

Le ratio entre le temps de calcul de SSP_{DVP} et DSSP est montré sur la figure 3.24. Les valeurs moyennes et médianes, sans dimension, sont affichées pour chaque valeur de T. Nous avons tracé une fonction de régression linéaire en accord avec la valeur théorique du ratio des complexité en temps de calcul. La valeur R^2 de cette régression est $0,94$.

Les valeurs R^2 associées aux fonctions de régression des figures 3.22, 3.23 et 3.24 nous indiquent que les régressions sont de très bonne qualité. Ces fonctions correspondent extrêmement bien aux résultats expérimentaux.

Le comportement des algorithmes en pratique est fidèle au comportement théorique que nous attendions. Cela signifie que la complexité en moyenne est du même ordre que la complexité au pire.

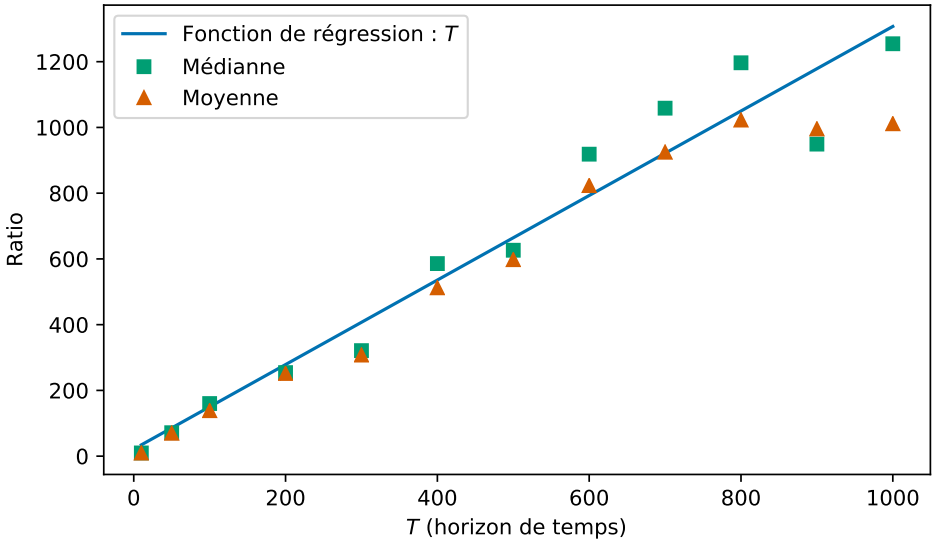


Figure 3.24 : Ratio des temps d'exécution de SSP_{DVP} et $DSSP$ en fonction de T.

La complexité théorique au pire de DSSP est meilleure que la complexité théorique au pire de SSP_{DVP}. En effet, il y un facteur T de plus pour SSP_{DVP}. Avec cette expérience, on a montré que DSSP est aussi bien plus efficace en pratique.

Le ratio des temps de calcul, représenté sur la figure 3.24, nous donne un moyen simple pour comparer les deux méthodes de résolution. Plus le graphe a un horizon de temps lointain, plus notre algorithme se démarque de SSP_{DVP}. On peut observer qu'il est environ 600 fois plus rapide lorsque le graphe a 500 pas de temps et plus de 1000 fois plus rapide lorsqu'il en a 1000.

3.3.4.5 Expérience en fonction de n

n	{500; 700; 900; 1000; 1100; 1300; 1500; 1700; 1900; 2000}
T	100
th	0,08
Q	80% du flot max estimé
Position de la source	(0,25; 0,75)
Position du puits	(0,75; 0,25)

Table 3.11 : Paramètres de l'expérience faisant varier n

De la même manière que pour l'expérience précédente, nous nous intéressons ici au temps de calcul de l'algorithme 1, mais cette fois-ci, en fonction du nombre de nœuds n du graphe. Nous faisons donc varier n en ayant, au préalable, fixé tous les autres paramètres. Le nombre de pas de temps du graphe est fixé à 100 et n varie entre 500 et 2000 dans l'ensemble suivant : {500 ; 700 ; 900 ; 1000 ; 1100 ; 1300 ; 1500 ; 1700 ; 1900 ; 2000}. Le tableau 3.11 synthétise tous les paramètres utilisés pour cette expérience.

Comme dans l'expérience précédente, nous comparons notre algorithme à la méthode SSP_{DVP}. La complexité en temps au pire de cette méthode est $O(Q \cdot T \cdot (m + n \cdot \log(n \cdot T)))$. Comme il a été possible de le faire précédemment, nous pouvons, ici aussi, simplifier cette écriture. Nous savons que Q est linéaire

en n. L'horizon de temps T est fixé et le nombre d'arcs m est de l'ordre de n^2. En reportant tous ces éléments dans l'écriture de la complexité de cette méthode, cela devient $O(n^3)$.

Notre algorithme DSSP a une complexité en temps au pire de $O((Q+T)\cdot(m+n\cdot\log(n)+\log(T)))$. Comme nous venons de le faire pour la complexité de la méthode SSP_{DVP}, nous pouvons aussi simplifier l'écriture de la complexité de DSSP. Cela devient $O(n^3)$. Ces complexités sont rassemblées dans le tableau 3.12.

Comme précédemment, nous utilisons le ratio des temps de calcul pour comparer les deux méthodes. Dans l'expérience présente où le seul paramètre variable est le nombre de nœuds, ce ratio est une constante.

Nous souhaitons montrer expérimentalement qu'il existe bien un facteur constant entre les temps de calcul de la méthode SSP_{DVP} et les temps de calcul de DSSP.

Complexité	Cas général	Dans l'expérience présente
SSP_{DVP}	$O(Q\cdot T\cdot(m+n\cdot\log(n\cdot T)))$	$O(n^3)$
$DSSP$	$O((Q+T)\cdot(m+n\cdot\log(n)+\log(T)))$	$O(n^3)$

Table 3.12 : Complexités théoriques des deux algorithmes comparés lorsque n est fixé

Nous nous intéressons maintenant aux temps de calcul des deux méthodes comparées en fonction de la taille du graphe en terme de nombre de nœuds. Ces résultats sont présentés sous forme de graphiques dans les figures 3.25, 3.26 et 3.27. Les valeurs médianes et moyennes obtenues pour chaque valeur de n sont indiquées sur le graphique ainsi qu'une courbe de régression dont la forme a été déterminée grâce à l'expression théorique des complexités. Les régressions, dont la qualité est évaluée avec la valeur R^2, permettent de comparer les résultats expérimentaux avec les attendus théoriques. Le tableau 3.13 rassemble les valeurs de R^2 obtenus pour cette expérience.

Le temps de calcul de SSP_{DVP} est montré dans la figure 3.25. Le temps de calcul moyen et médian obtenu sur la série d'instances testées pour chaque valeur de n, exprimé en secondes, est affiché sur le graphique. On affiche aussi une fonction de régression de la forme $O(n^3)$, comme la complexité théorique. La valeur R^2 de cette régression est $0,98$. Sur un graphe à 500 nœuds, l'exécution de l'algorithme est quasiment instantanée alors que sur un graphe à 2000 nœuds, l'exécution prend jusqu'à 120000 secondes, soit plus de 33 heures de calcul.

Les temps d'exécution de notre algorithme DSSP sont montrés sur la figure 3.26. On retrouve, comme précédemment, les temps moyens et médians ainsi qu'une fonction de régression. La forme de cette fonction est $O(n^3)$, comme la complexité théorique, et sa valeur R^2 est $0,98$. Dans un graphe dynamique avec 500 nœuds, le calcul est quasiment instantané, et il prend environ 700 secondes, soit à peine 12 minutes, dans un graphe qui a 2000 nœuds.

Le ratio entre le temps de calcul de SSP_{DVP} et celui de DSSP est l'objet de la figure 3.27. On trouve sur la figure les valeurs moyennes et médianes obtenues pour chaque valeur de n. Ces valeurs n'ont pas de dimension, puisqu'il s'agit d'un ratio. Une fonction de régression constante est tracée sur le graphique. La valeur de cette constante est environ 157, ce qui correspond exactement au temps d'exécution moyen.

	R^2
SSP_{DVP}	$0,98$
$DSSP$	$0,98$
Ratio	*Non pertinent*

Table 3.13 : Valeur R^2 des courbes de régressions sur les temps de calcul pour l'expérience faisant varier le nombre de nœuds n.

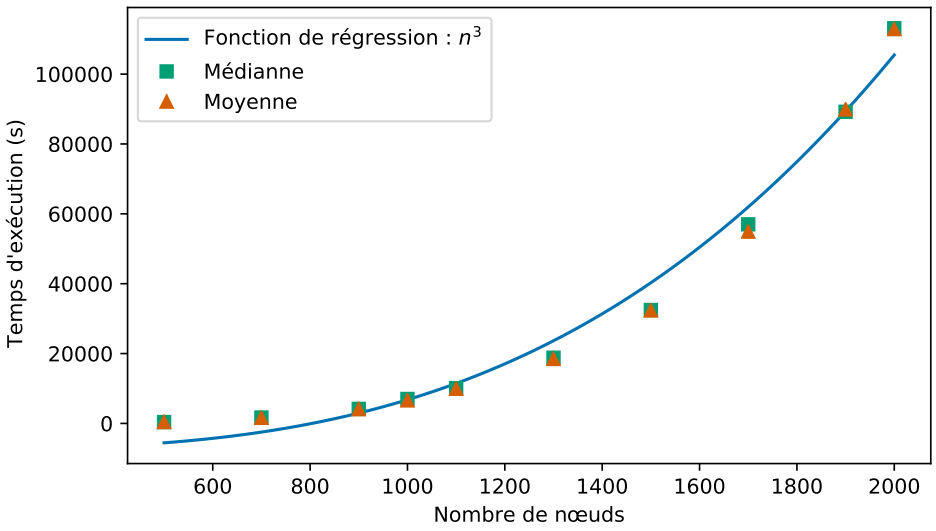


Figure 3.25 : Temps d'exécution de SSP_{DVP} en fonction de n, exprimé en secondes.

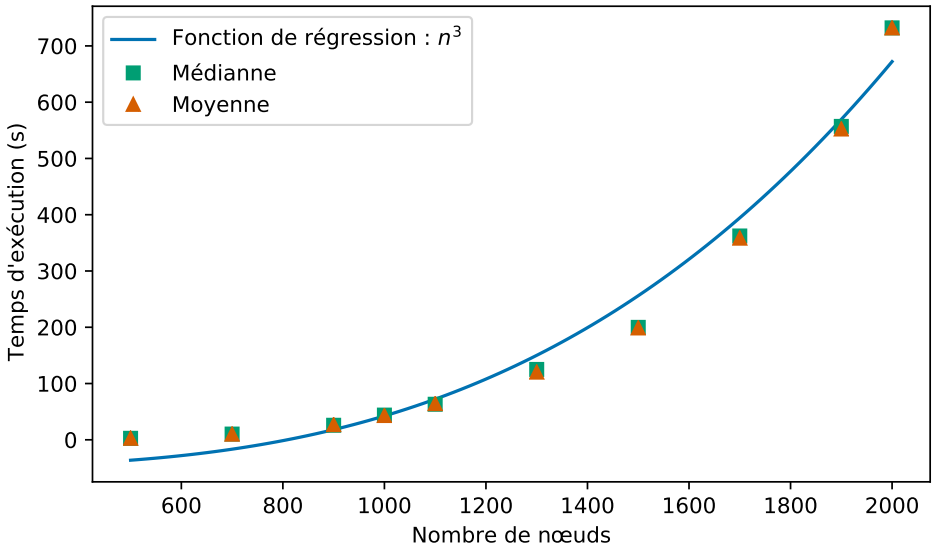


Figure 3.26 : Temps d'exécution de $DSSP$ en fonction de n, exprimé en secondes.

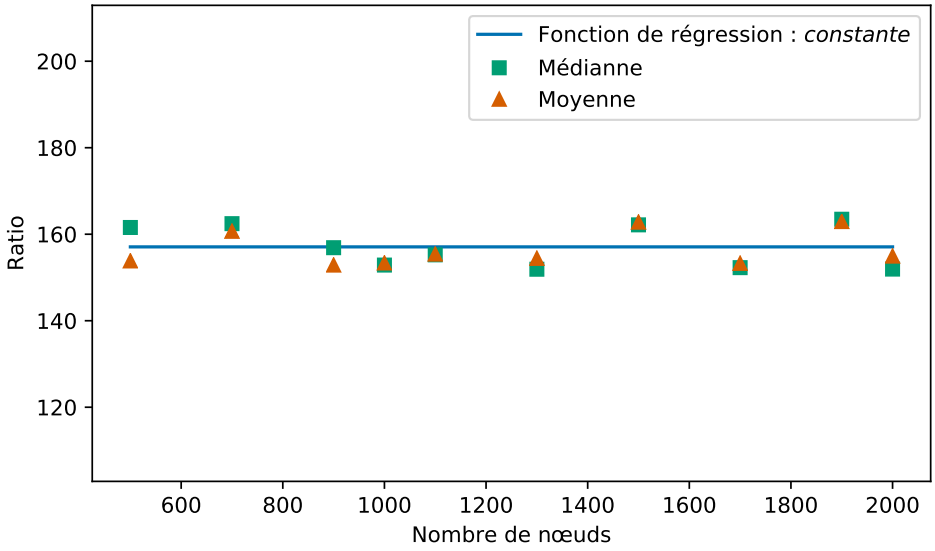


Figure 3.27 : Ratio des temps d'exécution de SSP_{DVP} et $DSSP$ en fonction de n, $constante \approx 157$.

Les valeurs R^2 associées aux fonctions de régression que nous avons tracées dans les figures 3.25, 3.26 montrent que celles-ci correspondent très bien aux données expérimentales.

Pour le cas particulier du ratio entre les temps de calcul de SSP_{DVP} et $DSSP$ (figure 3.27), nous ne donnons pas la valeur R^2 car celle-ci n'a pas de sens ici. En effet, avec une fonction de régression constante, la valeur R^2 est toujours nulle. Cela ne permet donc pas de se prononcer sur la qualité de la régression.

Les données expérimentales, exprimées par les temps de calcul, correspondent bien au comportement théorique, exprimé par les fonctions de régression, auquel nous nous attendions. Pour cette expérience en fonction du nombre de nœuds, comme pour l'expérience précédente en fonction de l'horizon de temps du graphe dynamique, le comportement expérimental correspond à la complexité au pire.

Dans ce cas particulier où, excepté n (et m indirectement), tous les paramètre sont fixés, on a pu constater que la complexité en temps au pire de DSSP était équivalente à la complexité en temps au pire de SSP_{DVP}. On a montré qu'en pratique, DSSP est malgré tout beaucoup plus rapide. En effet, pour rappel, avec 2000 nœuds, il y a moins de 12 minutes de calcul dans le premier cas contre environ 33 heures en moyenne dans le second.

Le ratio théorique des temps de calcul entre SSP_{DVP} et DSSP est constant, et c'est aussi le cas en pratique, comme le montre la figure 3.27. Expérimentalement, ce ratio vaut approximativement 157. Cela signifie que sur les graphes construits, avec 100 pas de temps, l'exécution de DSSP est 157 fois plus rapide que l'exécution de SSP_{DVP} sur les mêmes graphes.

Il est intéressant de noter que cette valeur obtenue dans cette expérience se retrouve aussi dans la première expérience où n était fixé est T variait. En effet, en observant la figure 3.24, pour la valeur $T = 100$ (comme dans la présente expérience), on a un ratio similaire.

La valeur T influe sur le ratio des temps de calcul, mais celui-ci ne dépend pas du nombre de nœuds du graphe comme le montre cette seconde expérience.

3.4 Conclusion

Nous avons consacré ce chapitre aux problèmes de flot dans les graphes dynamiques. Les flots sont bien connus dans les graphes statiques. Plusieurs problèmes de flot, dont les objectifs à optimiser sont

différents, existent dans la littérature.

Lorsque l'on s'intéresse aux graphes dynamiques, on peut s'interroger sur la définition des problèmes de flot dans ce contexte. La dimension temporelle a en plus fait naitre de nouveaux problèmes de flots inexistants dans les graphes statiques.

Il existe une littérature assez foisonnante sur les problèmes de flot dans les graphes dynamiques. Le nombre important de modèles de graphes possibles pour l'étude des flots, les différents problèmes existants ainsi que les différentes méthodes de travail proposant des démonstrations de résultats théoriques ou des méthodes de résolution des problèmes, font que de nombreux travaux existent dans la littérature. Ces travaux traitent différents problèmes sous différents angles.

Nous nous sommes focalisés sur des modèles spécifiques, sans temps de traversée, et où les autres paramètres varient dans le temps.

Nous nous sommes d'abord intéressés au problème du flot maximum. Nous avons, pour cela, travaillé sur un graphe dynamique dans lequel une quantité infinie de flot pouvait être stockée sur les nœuds.

Dans un second temps, nous avons travaillé sur un modèle dans lequel le stockage d'unités de flot sur les nœuds était interdit. Sur ce modèle, nous avons étudié le problème du flot de cout minimum.

Dans les deux cas, notre objectif était le même. Nous souhaitions proposer une méthode exacte de résolution du problème qui soit efficace. Cela impliquait donc de ne pas utiliser le graphe développé qui, bien qu'il permette de résoudre le problème de façon optimale, est de taille beaucoup trop importante.

Dans notre travail sur le problème du flot maximum dans un graphe dynamique avec stockage et sans temps de traversée, nous avons souhaité exploiter la connaissance que l'on a du graphe. Puisque nous connaissons le graphe dans son ensemble, c'est-à-dire tous les nœuds et tous les arcs, et puisque nous savons comment le graphe évolue dans le temps, c'est-à-dire, comment vont évoluer les capacités précisément, nous voulions utiliser ces informations afin de déterminer par quels arcs le flot doit être envoyé.

Dans ce but, nous avons mis au point une méthode de capacités cumulées sur les arcs qui indiquent une quantité maximum de flot qui peut traverser l'arc à partir d'une date donnée ou jusqu'à une date donnée. Cela nous a permis de déterminer une borne sur la quantité de flot pouvant traverser chaque arc pendant l'intervalle d'étude du graphe, et donc d'en déduire une borne sur le flot maximum. Cette borne se calcule en $O(m \cdot T + m \cdot n^2)$.

Malheureusement, cette démarche ne nous a pas permis d'obtenir un flot maximum sur le graphe dynamique. En effet, nous avons pu générer expérimentalement des contre-exemples dans lesquels la valeur de flot renvoyée par cette méthode n'était pas faisable. Nous obtenons, grâce à cette méthode, une borne supérieure du flot maximum.

Cette borne supérieure se calcule en un temps linéaire en T contrairement à la valeur exacte sur le graphe développé qui se calcule en un temps de l'ordre de T^3. Et la borne obtenue est supérieure à la valeur exacte de moins de 5% pour les trois quarts des instances testées.

Malgré nos tentatives, nous n'avons pas été en mesure de concevoir un nouvel algorithme exact. Notons cependant que la résolution du flot maximum sur un modèle sans stockage, donc en résolvant T flots maximum indépendants, donne une borne inférieure évidente. Nous avons donc un encadrement de la valeur exacte du flot maximum.

Nous avons aussi étudié le problème du flot de cout minimum. Nous avons travaillé sur un modèle de graphe assez restreint dans lequel il n'y a pas de temps de traversée et où le stockage sur les nœuds est interdit.

Nous avons su exploiter la particularité du modèle pour proposer un algorithme polynomial exact, de complexité $O((Q+T)\cdot(m+n\cdot\log(n)+\log(T)))$, pour résoudre le problème. Nous avons prouvé que

la complexité de notre algorithme est significativement meilleure que celle de la méthode classique qui consiste à utiliser le graphe développé. En se basant sur SSP, il semble difficile d'obtenir une complexité significativement meilleure que celle que nous avons : comment faire l'économie de $O(Q)$ appels à l'algorithme de Dijsktra ?

Nous avons validé ces résultats théoriques par une étude expérimentale. Nos expériences ont montré, d'une part, que le comportement pratique de notre algorithme est fidèle à son comportement théorique et d'autre part que l'exécution est suffisamment rapide pour que cet algorithme puisse être utilisé en pratique.

L'algorithme que l'on propose et son efficacité reposent intégralement sur le modèle que l'on considère. Il n'est donc malheureusement pas possible d'étendre l'utilisation de cet algorithme à d'autres modèles de graphes dynamiques.

Ce travail a été présenté à plusieurs reprises lors de conférences et a fait l'objet de la rédaction d'un article qui a été publié dans *Discrete Applied Mathematics* (Vernet et al., 2020).

Nous nous sommes focalisés sur des modèles de graphes sans « inconnues », c'est à dire des modèles où nous connaissions tout le graphe et où nous connaissions son évolution dans le temps. Nous pouvons donc à présent nous interroger sur la manière dont les problèmes de flot pourraient être traités sur des modèles d'autres types, comme des modèles où la structure du graphe et son évolution ne seraient pas connues à l'avance.

Si l'on ne connait pas l'évolution du graphe, alors on ne peut pas décider à l'avance des arcs par lesquels on envoie du flot. On risquerait dans ce cas de faire de très mauvais choix qui nous donneraient une solution très éloignée de la solution optimale. Il faudrait donc définir une politique nous permettant de prendre une décision à chaque pas de temps en fonction du graphe à ce même pas de temps.

Si l'on ne connait pas le graphe dans son ensemble, cela implique nécessairement de faire des choix locaux, au niveau des nœuds, pour savoir vers quels arcs envoyer le flot. En ce sens, cela se rapprocherait d'un problème de routage dans un réseau télécoms.

En résumé, nous avons travaillé sur les problèmes de flot dans des graphes connus à l'avance en ayant l'objectif de résoudre les problèmes de manière exacte, en évitant l'utilisation du graphe développé. Nos travaux sur le flot maximum ne nous ont pas permis de remplir cet objectif, mais nous avons tout de même pu mettre en évidence une borne supérieure du flot maximum. En revanche, nos travaux sur le flot de cout minimum nous ont permis de développer un algorithme efficace aussi bien théoriquement qu'expérimentalement. Une direction que pourrait prendre ce travail serait de s'intéresser à des modèles différents où l'on a une connaissance partielle du graphe pour proposer des algorithmes en ligne. De plus, il serait pertinent de s'interroger de manière plus globale sur les problèmes de flots et d'identifier, pour chaque variante, selon le problème étudié et le modèle de graphe considéré, la meilleure complexité d'algorithme, théoriquement et pratiquement, qu'il est possible d'obtenir.

Chapitre 4

Connexité

Ce chapitre traite des questions de connexité dans un graphe dynamique en se focalisant sur la notion de composante connexe et sur les problèmes associés. C'est une problématique bien connue dans les graphes statiques.

Identifier les composantes connexes d'un graphe peut s'avérer utile dans de nombreux domaines d'application. Cela peut apporter de l'information sur les réseaux de communication, les réseaux logistiques ou encore les réseaux sociaux.

Mais outre les applications, la décomposition d'un graphe en composantes connexes peut aussi permettre de décomposer certains problèmes de graphes afin de les résoudre séparément sur les différentes composantes du graphe. On pourra citer par exemple les problèmes de coloration, de couplage ou de tournées de véhicules.

On notera de plus l'importance que peut avoir la connexité d'un graphe en tant que pré-requis. Afin de résoudre certains problèmes, il peut être nécessaire de s'assurer en amont que le graphe est connexe. Un problème de flot, par exemple, ne pourra être résolu que si la source et le puits du graphe sont dans la même composante connexe.

La grande importance de la notion de connexité dans les graphes statiques amène à s'interroger sur cette notion dans un graphe dynamique. Qu'est-ce qu'une composante connexe dans un graphe dynamique ? Comment peut-on identifier ces composantes ?

De plus, d'autres problèmes dans les graphes statiques sont fortement liés à la notion de connexité. C'est par exemple le cas du problème de l'arbre de Steiner. Nous nous interrogeons également sur la manière de traduire ces problèmes dans un contexte dynamique.

C'est l'objet du présent chapitre. Nous proposons une extension de la notion de composante connexe dans un graphe dynamique que nous appelons *composante connexe persistante* ainsi qu'un algorithme permettant d'identifier celles qui sont non-dominées. Ce travail a fait l'objet d'un article soumis (Vernet et al., 2020). Nous présentons aussi nos premières réflexions sur le problème de Steiner, en collaboration avec Stefan Balev (LITIS, équipe RI2C). Elles ont fait l'objet d'un stage de master (Moal, 2020) et d'un article soumis (Balev et al., 2020).

La section 4.1 fait une revue de la littérature sur les problèmes de connexité et autres problèmes liés dans un contexte dynamique. Nous présentons en section 4.2 l'environnement dynamique dans lequel se place notre étude ainsi que nos définitions de connexité. L'algorithme que nous avons conçu est présenté en section 4.3. Nous avons réalisé une étude expérimentale sur cet algorithme que nous présentons en section 4.4. La section 4.5 présente les possibles extensions de notre définition de composante connexe. Nous traitons du problème de Steiner dans un contexte dynamique en section 4.6. Enfin la section 4.7 apporte une conclusion à nos travaux sur les problèmes de connexité dans les graphes dynamiques.

4.1 État de l'art

La notion de connexité et les problèmes liés à la connexité dans les graphes sont des questions connues qui ont été bien étudiées dans le contexte statique. Nous nous intéressons au cas dynamique et explorons dans cette section les travaux existants.

4.1.1 La connexité en contexte dynamique

Dans le cas statique, une composante connexe dans un graphe est définie comme un ensemble maximal de nœuds tel que pour chaque paire de nœuds, il existe un chemin entre eux. Comme cela a été défini dans le chapitre 2 dans la définition 7, un analogue au chemin, dans un graphe dynamique, est le trajet qui prend en compte les temps de traversée des arêtes en plus de leur présence ou absence aux différents pas de temps successifs de l'intervalle d'étude. Cette définition reste valable lorsque les temps de traversée sont nuls, autrement dit lorsque nous ne considérons pas de temps de traversée.

On trouve donc assez logiquement dans la littérature des définitions de composantes connexes basées sur la notion de trajet. Nous discutons de ces travaux en premier. Nous étudions les autres types de définitions dans un second temps.

4.1.1.1 Définitions utilisant les trajets

L'existence d'un chemin entre chaque paire de nœuds distincts dans un ensemble de nœuds, maximal pour l'inclusion, d'un graphe statique non orienté définit une composante connexe. Dans les graphes orientés maintenant, la définition s'étend de deux manières différentes. On peut considérer des composantes fortement ou faiblement connexes.

Une composante fortement connexe désigne un ensemble maximal de nœuds tel que pour chaque paire de nœuds distincts i et j, il existe un chemin orienté de i vers j et un chemin orienté de j vers i.

Une composante faiblement connexe est un ensemble maximal de nœuds tel que pour chaque paire de nœuds distincts i et j il existe un chemin entre i et j lorsque l'on ignore l'orientation des arcs.

Dans les graphes dynamiques, le fait que les liens entre les nœuds peuvent apparaitre et disparaitre impose une orientation sur les trajets, même dans les graphes non orientés. L'existence d'un trajet allant du nœud i vers le nœud j ne présume pas de l'existence d'un trajet de j vers i. Que l'on étudie des graphes dynamiques orientés ou non orientés, les trajets, eux, sont orientés.

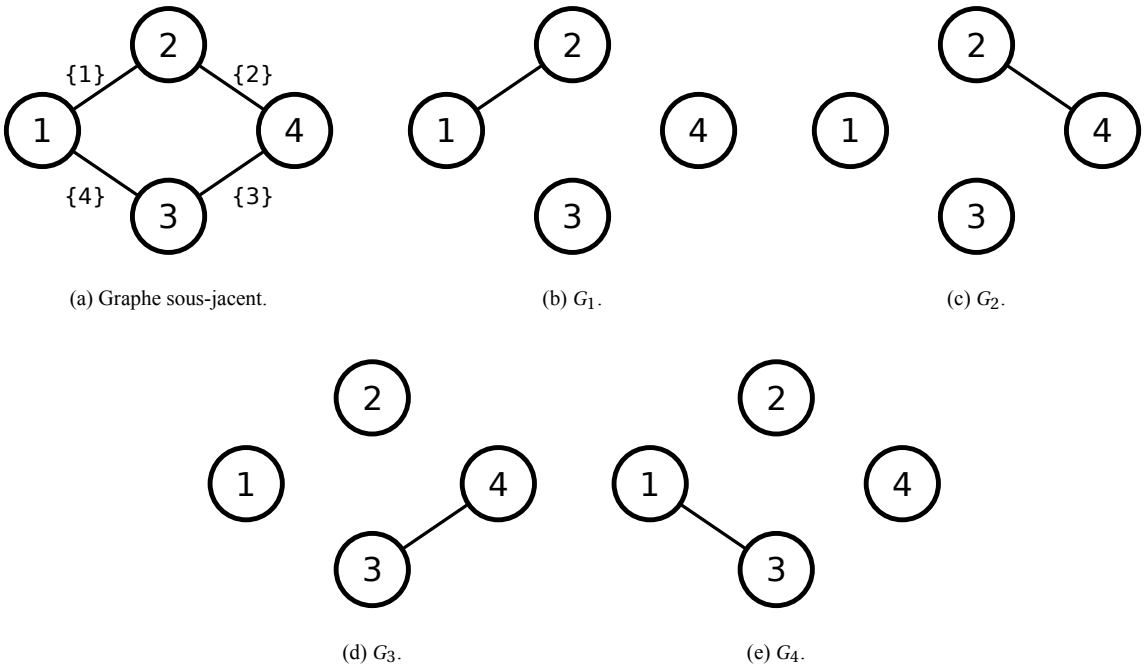
On peut observer ce phénomène sur le graphe présenté dans la figure 4.1. Il s'agit d'un graphe dynamique non orienté. La figure 4.1a montre le graphe sous-jacent où sont indiqués sur chaque arête les pas de temps où ladite arête est présente. Les figures 4.1b à 4.1e montrent chaque t-graphe afin de voir l'évolution du graphe. Dans ce graphe, il y a un trajet du nœud 1 vers le nœud 4 passant par le nœud 2, et un trajet du nœud 4 vers le nœud 1 passant par le nœud 3. Il existe un trajet du nœud 2 vers le nœud 3 passant par le nœud 4 mais il n'existe pas de trajet du nœud 3 vers le nœud 2. En effet, le nœud 3 est déconnecté du reste du graphe pendant les deux premiers pas de temps, et à partir du troisième pas de temps, il est connecté uniquement à des nœuds ne permettant pas d'atteindre le nœud 2 après cette date.

On devine facilement que ce phénomène va impacter les notions de connexité dans les graphes dynamiques. Les définitions basées sur les trajets devront prendre en compte l'orientation des trajets.

Bhadra et Ferreira (2003) étudient des graphes dynamiques orientés. En se basant sur la définition de trajet (en anglais *journey*) explicitée par Bui-Xuan et al. (2003), ils proposent une définition de composante fortement connexe. On notera que ces définitions peuvent tout à fait s'appliquer à des graphes dynamiques non-orientés.

De façon similaire à la définition qu'on l'on connait dans un graphe statique, une composante fortement connexe dans un graphe dynamique est un ensemble maximal de nœuds tel que pour chaque paire de nœuds i et j de l'ensemble, il existe un trajet de i vers j et un trajet de j vers i.

La spécificité de cette définition dans les graphes dynamiques repose sur le fait qu'il existe deux cas de figures des composantes fortement connexes dans les graphes dynamiques. On parle de composantes fortement connexes fermées ou ouvertes. Dans les composantes fermées, les trajets qui relient un nœud i de la composante à un nœud j de la composante ne passent que par des nœuds de la composante. À



(a) Graphe sous-jacent. (b) G_1. (c) G_2.

(d) G_3. (e) G_4.

Figure 4.1 : Graphe dynamique sur 4 pas de temps. Sur le graphe sous-jacent, les pas de temps de présence sont notés à côté des arêtes. Chaque t-graphe est représenté.

l'inverse, dans les composantes ouvertes, les trajets qui relient un nœud i de la composante à un nœud j de la composante peuvent passer par des nœuds qui ne font pas partie de la composante.

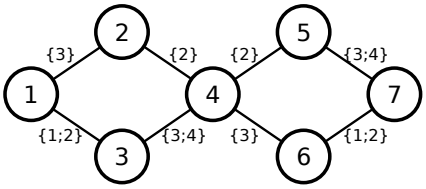
Observons le graphe de la figure 4.1. Les nœuds $\{1;2;4\}$ forment une composante fortement connexe ouverte. En effet, il existe un trajet dans chaque sens entre chaque paire de nœuds de l'ensemble, ce qui définit la composante fortement connexe. Le nœud 3 ne peut pas être ajouté à l'ensemble car il n'existe pas de trajet du nœud 3 vers le nœud 2 comme on l'a vu précédemment. On observe de plus que le trajet qui permet d'aller du nœud 4 vers le nœud 1 passe par le nœud 3 qui ne fait pas partie de l'ensemble. Donc cette composante est une composante fortement connexe ouverte.

Cette vision de la connexité dans un graphe dynamique a une conséquence particulière. Contrairement au contexte statique où les composantes connexes forment une partition des nœuds du graphe, dans le contexte dynamique, ce n'est pas le cas. En effet les composantes connexes peuvent s'intersecter et certains nœuds peuvent faire partie de plusieurs composantes fortement connexes.

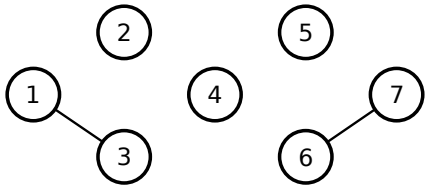
Prenons comme exemple le graphe dynamique de la figure 4.2. La figure 4.2a montre le graphe sous-jacent et les figures 4.2b à 4.2e montrent chaque t-graphe. L'ensemble $\{1;2;3;4\}$ est une composante fortement connexe fermée. En effet, il existe un trajet dans chaque sens entre chaque paire de nœuds et ces trajets ne passent que par des nœuds de l'ensemble. Pour les mêmes raisons, l'ensemble $\{4;5;6;7\}$ est aussi une composante fortement connexe fermée. Le nœud 4 fait donc partie de deux composantes fortement connexes fermées distinctes.

On peut aussi noter que certains nœuds peuvent se trouver dans la même composante fortement connexe, qu'elle soit ouverte ou fermée, et pourtant ne jamais être connectés à aucun pas de temps. En d'autres termes, deux nœuds qui ne sont jamais dans la même composante connexe statique dans aucun t-graphe peuvent se trouver dans la même composante fortement connexe dynamique. C'est le cas par exemple dans la figure 4.1 où il n'existe jamais de chemin ou d'arête entre les nœuds 1 et 4 mais ils se trouvent bien dans la même composante fortement connexe ouverte.

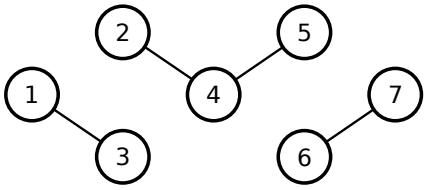
En plus de définir ces notions de connexité dans des graphes dynamiques, Bhadra et Ferreira s'intéressent au problème qui consiste à trouver de telles composantes. Les auteurs prouvent que pour un



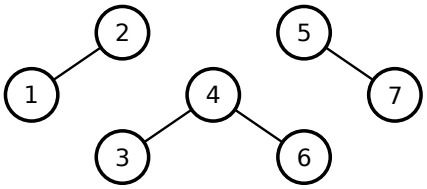
(a) Graphe sous-jacent.



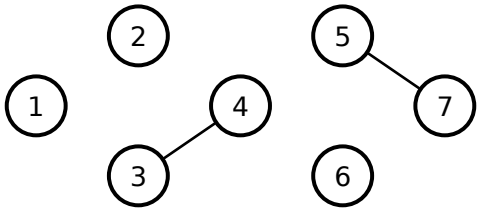
(b) G_1.



(c) G_2.



(d) G_3.



(e) G_4.

Figure 4.2 : Graphe dynamique sur 4 pas de temps. Sur le graphe sous-jacent, les pas de temps de présence sont notés à côté des arêtes. Chaque t-graphe est représenté. Il y a deux composantes connexes fermées : $\{1;2;3;4\}$ et $\{4;5;6;7\}$.

entier k donné, trouver une composante connexe, ouverte ou fermée, de taille k est un problème NP-complet (pour rappel, le problème analogue dans un graphe statique est polynomial, et plus précisément, linéaire en m).

Les travaux de la littérature qui s'intéressent à la connexité dans les graphes dynamiques basée sur les notions de trajet s'appuient sur les définitions de composantes connexes développées par Bhadra et Ferreira.

Nous modifions parfois légèrement les notations choisies par les auteurs afin de ne pas confondre les définitions proposées dans des travaux différents. Nous le précisons lorsque c'est le cas.

Jarry et Lotker (2004) s'intéressent au problème qui consiste à trouver les composantes fortement connexes définies par Bhadra et Ferreira. Ces derniers ont prouvé que le problème était NP-complet, mais Jarry et Lotker considèrent des graphes particuliers. Ils ont été en mesure de prouver que le problème est aussi NP-complet pour des graphes qu'ils nomment *two-layer grids* (qui sont des grilles sur lesquelles un arc est ajouté entre les nœuds à distance 2), mais qu'il est polynomial dans le cas des arbres. Ils proposent en plus un algorithme de résolution pour ce dernier cas.

Nicosia et al. (2012) travaillent sur des graphes dynamiques qui sont vus comme une séquence de graphes statiques sur un même ensemble de nœuds. Il s'agit aussi du modèle que nous utilisons, tel que nous l'avons établi dans la définition 4 (donnée page 13). Les auteurs précisent que l'intervalle de

temps discret sur lequel est défini un graphe n'est pas forcément uniforme, c'est-à-dire que le temps réel écoulé entre deux étapes successives n'est pas toujours le même et ne reste pas constant dans le temps.

Ils travaillent sur une définition de la connexité qui correspond aux composantes fortement connexes ouvertes définies par Bhadra et Ferreira (2003).

Ils définissent un graphe statique non-orienté qu'ils appellent en anglais *affine graph*. Ce graphe est construit à partir d'un graphe dynamique donné. Il possède les mêmes nœuds que le graphe dynamique et il existe une arête entre deux nœuds s'il existe un trajet dans chaque sens entre ces nœuds. Trouver une composante fortement connexe ouverte dans le graphe dynamique est équivalent à trouver une clique maximum dans le graphe statique défini, ce qui n'est pas polynomial.

Gómez-Calzado et al. (2015) travaillent sur des graphes dynamiques non-orientés en temps continu. Les arêtes peuvent apparaitre et disparaitre dans le temps. Un temps de traversée des arêtes est considéré. Lors d'un trajet, il est possible d'attendre sur un nœud entre la traversée d'une arête et d'une autre.

Ils étudient des définitions de composantes connexes basées sur des trajets, et comme Bhadra et Ferreira, ils font la différence entre les composantes ouvertes ou fermées. Ils proposent la définition des Δ-composantes dans lesquelles les trajets connectant deux nœuds durent au plus Δ, ce qu'ils nomment des Δ-trajets. Et il existe un Δ-trajet entre chaque paire de nœuds de la Δ-composante sur chaque fenêtre de temps de longueur Δ de l'intervalle d'étude du graphe.

Huyghues-Despointes et al. (2016) travaillent sur ce qu'ils appellent des flots de liens (*link stream* en anglais), qui correspondent à des graphes dynamiques. Dans le modèle qu'ils décrivent, une seule arête peut être traversée à chaque pas de temps dans un trajet. C'est équivalent à un modèle où chaque arête aurait un temps de trajet de valeur 1.

Ils définissent les δ-composantes (qu'ils nomment Δ-composantes). Il s'agit d'un ensemble de nœuds qui est connecté, au sens de la composante fortement connexe ouverte définie par Bhadra et Ferreira (2003), sur toutes les fenêtres de temps de longueur δ de l'intervalle d'étude du graphe. Lorsque δ est égal à l'horizon de temps, alors il s'agit exactement de la composante fortement connexe ouverte de Bhadra et Ferreira.

Ils définissent aussi un graphe d'accessibilité, à la manière du *affine graph* de Nicosia et al., qui est un graphe statique dans lequel deux nœuds sont reliés s'ils sont connectés au sens de la δ-composante. Et donc, toujours à la manière de Nicosia et al., une δ-composante est une clique maximale dans ce graphe d'accessibilité. Cela leur permet de prouver que la recherche d'une δ-composante maximum est NP-difficile.

Ces définitions sont basées sur la notion de trajets, correspondant aux chemins dans un graphe dynamique. L'objectif sous-jacent de la plupart de ces travaux est la transmission de message dans un réseau. C'est-à-dire que les composantes connexes, au sens dynamique, donnent une indication sur les nœuds entre lesquels l'envoi d'un message est possible dans un délai imparti.

Ces définitions présentent des points communs intéressants. Les ensembles de nœuds présentés comme des composantes connexes ne sont pas disjoints contrairement aux composantes connexes dans les graphes statiques. De plus, l'identification des composantes maximum est NP-difficile alors qu'il s'agit d'un problème polynomial dans un graphe statique.

D'autres définitions, ne prenant pas en compte l'existence d'un trajet entre les nœuds, sont possibles. C'est l'objet de la section suivante.

4.1.1.2 Autres définitions

Casteigts et al. (2015) utilisent la notion de T-intervalle connexité, originellement définie par Kuhn et al. (2010), et que nous appellerons τ-intervalle connexité par soucis de clarté entre toutes les notations utilisées.

Un graphe dynamique est τ-intervalle connexe lorsque le graphe statique résultant de l'intersection des t-graphes $G_\theta \dots G_{\theta+\tau-1}$, pour tout $\theta \in [1, T-\tau+1]$, T étant l'horizon de temps, est connexe au sens statique. En d'autres termes, un graphe dynamique est τ-intervalle connexe lorsque tous les t-graphes d'une fenêtre de temps de longueur τ de l'intervalle d'étude possèdent un sous-graphe commun couvrant tous les nœuds du graphe. Ou encore, n'importe lesquels des δ t-graphes successifs ont au moins un arbre couvrant commun.

Casteigts et al. s'intéressent au problème de décision qui consiste à déterminer si un graphe dynamique est τ-intervalle connexe pour une valeur τ donnée. Ils s'intéressent aussi au problème d'optimisation correspondant visant à trouver la plus grande valeur de τ telle que le graphe dynamique est τ-intervalle connexe.

Ils montrent que ces problèmes peuvent être résolus en réalisant $\Theta(T)$ opérations d'intersection de deux graphes et de test de connexité de graphes qui sont des opérations linéaires en m. Ils proposent des algorithmes optimaux permettant de résoudre ces problèmes.

Ce travail ne propose pas de définition de composante connexe basée sur la définition de τ-intervalle connexité.

Akrida et Spirakis (2019) définissent un modèle de graphes pour lesquels un ensemble d'intervalles de présence est donné pour chaque arête. C'est-à-dire qu'une arête donnée va être présente pendant les intervalles définis et absente le reste du temps. Le temps est défini comme continu dans leur modèle, ce qui est une différence majeure avec les autres travaux que l'on a étudiés jusque là.

Ils proposent un premier algorithme qui permet de trouver le plus long intervalle de temps, commençant à une date x donnée pendant lequel le graphe est continuellement connexe. Il s'agit d'un algorithme polynomial.

Ils s'intéressent ensuite au problème consistant à trouver un ensemble de nœuds, de taille supérieure à une borne donnée, qui reste continuellement connecté pendant un intervalle de temps, commençant à une date x donnée, le plus long possible. Ils proposent un algorithme polynomial permettant de résoudre ce problème.

À l'opposé des définitions de composantes connexes dans des graphes dynamiques basées sur les trajets vus en section 4.1.1.1, les ensembles de nœuds recherchés par Akrida et Spirakis ne s'intersectent pas et contrairement au travail de Casteigts et al. (2015), la connexion entre les nœuds ne doit pas nécessairement se faire via les mêmes arêtes.

Pour chacun des algorithmes présentés dans ce travail, le choix du paramètre x, qui est la date de départ des ensembles connexes recherchés, détermine totalement le résultat de l'algorithme. Si le graphe est connexe seulement à partir d'une date $x+\epsilon$, $\epsilon > 0$, le premier algorithme présenté n'est pas en mesure de le détecter. De la même manière, si de grandes composantes connexes apparaissent dans le graphe à partir d'une date $x+\epsilon$, $\epsilon > 0$, le second algorithme, appelé avec le paramètre x, n'est pas en mesure de trouver ces composantes.

4.1.2 Problèmes liés

D'autres travaux traitent de problèmes liés à la notion de connexité sans s'interroger sur les composantes connexes. On peut par exemple citer le problème de l'arbre couvrant de poids minimum. Si l'on considère un arbre unique sur tout l'intervalle d'étude, et dont le poids total serait minimum, alors le problème est très simple puisqu'il suffit de considérer le graphe statique résultant de l'intersection de tous les t-graphes avec, sur chaque arête, la somme des poids de l'arête à chaque pas de temps. On se ramène de cette façon à un problème d'arbre couvrant de poids minimum dans un graphe statique.

Si l'on considère en revanche une autre manière de définir un arbre couvrant dans un graphe dynamique, alors le problème peut ne pas s'avérer aussi simple. C'est par exemple l'objet du travail de Huang et al. (2015) qui s'interrogent sur ce qu'est un arbre couvrant de poids minimum dans un graphe dynamique. Ils considèrent des graphes dynamiques orientés avec des temps de traversée et des poids

sur les arcs, ces paramètres étant variables. Ils considèrent un nœud racine r et définissent un arbre couvrant comme étant une arborescence de racine r avec un trajet, au sens dynamique, de r vers tous les autres nœuds du graphe. Ils considèrent un premier problème dont l'objectif est de minimiser les dates d'arrivée aux différents nœuds du graphe avec l'arbre couvrant et proposent une résolution exacte de ce problème. Ils considèrent ensuite un second problème dont l'objectif est de minimiser le poids de l'arbre couvrant et proposent une méthode de résolution grâce à une transformation vers un graphe statique.

Un problème assez similaire à celui de l'arbre couvrant de poids minimum consiste à rechercher, dans un graphe dynamique connexe au sens de Bhadra et Ferreira (2003), un sous-graphe couvrant dynamique lui aussi connexe. Ce problème, explicité par Kempe et al. (2002), a été traité récemment par Casteigts et al. (2019) qui montrent qu'un graphe dynamique connexe au sens ci-dessus, et dont le graphe sous-jacent est complet, admet toujours un sous-graphe couvrant dynamique connexe de densité $O(n \cdot \log(n))$.

On peut aussi s'intéresser au problème de Steiner, assez proche du problème de l'arbre couvrant de poids minimum dans sa définition. Le problème de l'arbre de Steiner (Karp, 1972 ; Garey et Johnson, 1979) est un problème d'optimisation dans les graphes dont le but est de trouver comment connecter certains nœuds identifiés du graphe. Nous rappelons ce problème dans la définition 9.

Définition 9 (Rappel du problème de l'arbre de Steiner)**.** *Soient $G = (V, E)$ un graphe (statique) et $w_e \geqslant 0$ un poids positif sur chaque arête $e \in E$. Pour un ensemble de nœuds $S \subset V$ donné, appelés nœuds terminaux, on cherche un arbre $G' = (V', E')$, sous-graphe de G où $S \subset V' \subset V$ et $\sum_{e' \in E'} w_{e'}$ est minimum.*

Ce problème présente deux cas particuliers polynomiaux. Lorsque le nombre de terminaux est exactement 2, alors le problème de Steiner devient un problème de plus court chemin dans un graphe, donc ce cas est polynomial. Et lorsque tous les nœuds du graphe sont des terminaux alors le problème de Steiner devient un problème d'arbre couvrant de poids minimum, lui aussi polynomial. Dans le cas général, l'arbre de Steiner est un problème NP-complet (Karp, 1972), y compris lorsque les poids sont unitaires (Garey et Johnson, 1979).

La littérature ne foisonne pas de travaux traitant du problème de Steiner dans un contexte dynamique. On peut néanmoins en trouver quelques uns.

Imase et Waxman (1991) travaillent sur un graphe auquel est associé un ensemble de changements. Chaque changement correspond à l'ajout ou à la suppression d'un nœud donné. C'est donc comme si on observait le graphe sur un certain nombre de pas de temps, et à chaque pas de temps, un nœud disparait ou un nœud apparait alors que d'autres nœuds restent fixes.

Ils s'intéressent à un premier problème qui consiste à trouver une séquence d'arbres, un pour chaque pas de temps, dont le poids total est minimum, tel que chaque arbre couvre les nœuds apparus à une date antérieure et encore présents à cette date. Aucune modification des arêtes de l'arbre n'est possible d'un pas de temps au suivant, si ce n'est concernant le nœud apparu ou disparu à cette date. Ils prouvent que pour tout algorithme d'approximation, il existe une instance du problème pour laquelle le ratio d'approximation ne peut pas être arbitrairement petit.

Ils s'intéressent ensuite à un second problème qui est très similaire au premier mais cette fois-ci, les modifications sont autorisés entre un arbre et le suivant dans la séquence. Ils proposent un algorithme d'approximation pour résoudre ce problème.

Khodaverdian et al. (2016) travaillent sur des graphes dynamiques vus comme une succession de graphes statiques sur le même ensemble de nœuds. Chaque arête possède un poids, qui ne dépend pas du temps.

Ils définissent le problème du k-réseau de Steiner temporel (k-*Temporal Steiner Network* en anglais). Ce problème vise à satisfaire k demandes distinctes avec un graphe statique (un sous-ensemble de nœuds du graphe dynamique de départ et un sous-ensemble des arêtes de son graphe sous-jacent) de poids minimum. Une demande est définie par un couple de nœuds $(a;b)$ et une date θ, et il doit exister un chemin entre a et b à la date θ dans le graphe. Ils définissent aussi la version de ce problème dans un graphe dynamique orienté, la différence réside dans le fait que les demandes sont orientées.

Khodaverdian et al. considèrent aussi un modèle de graphes où les arêtes ne peuvent pas disparaitre. C'est-à-dire qu'une fois qu'une arête est apparue, alors elle reste présente. Ils s'intéressent au problème de k-réseau de Steiner temporel dans ce cas-là.

Les auteurs prouvent que le problème de k-réseau de Steiner temporel, dans le premier cas, est NP-difficile à approximer par un facteur $k-\epsilon$. Ils prouvent, pour le second cas, qu'il existe un algorithme d'approximation polynomial permettant de le résoudre.

Ce problème cherche à connecter des nœuds deux par deux plutôt que de connecter un ensemble de nœuds. En ce sens, il est moins proche du problème de Steiner classique dont nous avons rappelé la définition précédemment (définition 9) que du problème de Steiner généralisé décrit par Agrawal et al. (1995), dans lequel l'objectif est de connecter un ensemble de paires de nœuds par une forêt de poids minimum.

Rozenshtein et al. (2016) s'intéressent à un problème de propagation d'information ou de virus et modélisent cela en définissant des arbres de Steiner dynamiques.

Ils travaillent sur des graphes dynamiques, orientés, en temps continu. Les arcs peuvent apparaitre et disparaitre plusieurs fois. Certains nœuds peuvent être marqués comme « activés ». Une fois activé, un nœud n'est jamais désactivé.

Ils s'intéressent à la notion de trajet. Puisqu'ils considèrent un graphe orienté, ils considèrent aussi des trajets orientés. Dans un trajet, il est possible d'avoir un temps d'attente non nul sur les nœuds. Le poids d'un trajet est déterminé en fonction du graphe et des nœuds actifs.

Ils considèrent des terminaux comme étant des nœuds à un pas de temps donné. Puis ils définissent un arbre de Steiner temporel comme étant un ensemble de trajets de poids minimum connectant les terminaux.

Rozenshtein et al. s'intéressent à un problème qui consiste à trouver un ensemble d'arbres de Steiner temporels pour connecter des terminaux. L'idée n'est pas de trouver un seul arbre pour connecter les terminaux mais un ensemble d'arbre en minimisant à la fois le poids des arbres et le nombre d'arbres utilisés. Ils utilisent pour cela des algorithmes d'approximation.

L'objectif de ce travail est applicatif. Les auteurs cherchent à déterminer comment une épidémie s'est propagée en utilisant les données connues de personnes infectées et des contacts entre les personnes.

4.2 Contexte

Comme nous l'avons vu, il existe, dans la littérature, des travaux portant sur la connexité dans les graphes dynamiques. Nous proposons une nouvelle définition permettant de généraliser la notion de composante connexe afin de la rendre adaptée aux graphes dynamiques.

Une composante connexe (dans un graphe statique) étant un ensemble de sommets connectés entre eux, nous étendons cela en parlant à présent d'un ensemble de sommets qui restent connectés entre eux pendant un certain intervalle de temps.

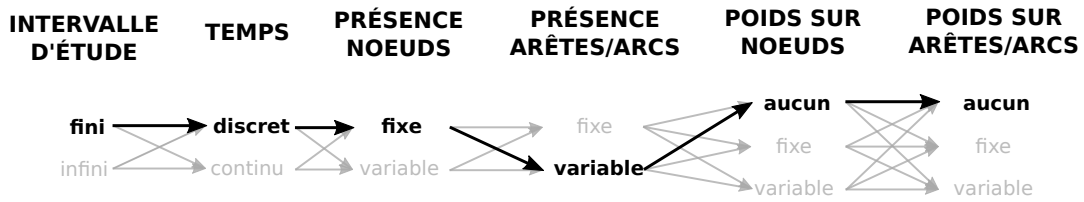


Figure 4.3 : Illustration du modèle de graphe dynamique étudié pour la question de la connexité.

4.2.1 Modèle

Comme nous l'avons expliqué dans le chapitre 2, nous observons, ici encore, les graphes sur un intervalle d'étude fini et discret.

On peut tout de même noter que les t-graphes peuvent correspondre aux instants où le graphe est observé. Ce qui signifierait alors que le temps effectif écoulé entre G_θ et $G_{\theta+1}$, pour $\theta \in \mathcal{T}$, n'est pas nécessairement le même que celui écoulé entre $G_{\theta'}$ et $G_{\theta'+1}$, pour $\theta' \in \mathcal{T}$. Cela n'a aucune influence sur nos définitions et le traitement des données.

Dans les graphes auxquels nous nous intéressons à présent, nous ne considérons aucun type de poids, ni sur les arcs ni sur les nœuds. L'ensemble des nœuds du graphe est constant tout au long de l'intervalle d'étude.

La dynamique agit uniquement sur les arêtes du graphe. En effet, celles-ci peuvent apparaitre et disparaitre pendant l'intervalle d'étude.

La figure 4.3 schématise le modèle de graphe étudié dans cette partie en reprenant les notations de la figure 3.1

Nous nous intéressons ici à l'évolution de la structure du graphe. C'est la raison pour laquelle nous pouvons travailler sur un modèle de graphe plus basique que celui étudié dans le chapitre 3.

4.2.2 Définitions et problème

Nous introduisons la notion de *composante connexe persistante*. Il s'agit d'une extension de la notion de composante connexe aux graphes dynamiques. Nous avons souhaité une définition qui prenne en compte la dimension temporelle du graphe. La définition 10 définit des composantes caractérisées par leur taille dans l'espace, en termes de nombre de nœuds qui les composent, et par leur durée dans le temps, en termes de nombre de pas de temps de présence consécutifs. Cette définition donne des composantes qui dépendent de la dynamique du graphe.

Définition 10 (Composante connexe persistante)**.** *Une composante connexe persistante (PCC pour persistant connected component en anglais) p dans un graphe dynamique G est un ensemble de k sommets dans V qui restent connectés dans le graphe pendant l pas de temps consécutifs. Ces sommets peuvent être connectés directement via des arêtes ou bien indirectement via des chemins passant par d'autres nœuds du graphe.*
Les nœuds $u_1 \dots u_k$ forment une composante connexe persistante de taille k et de durée l si et seulement si il existe $G_i \dots G_{i+l-1}$ tels que $u_1 \dots u_k$ sont dans la même composante connexe au sens statique dans chaque G_j $(i \leq j < i + l)$.
On note une PCC p de la façon suivante : $p = (K, k, l, f)$ où K est l'ensemble des nœuds qui la composent, k est la taille de cet ensemble, l est la durée de la composante (en d'autres termes, le nombre de pas de temps consécutifs pendant lesquels la composante est présente), et f est le dernier pas de temps de présence.

Définition 11 (Composante connexe persistante maximale)**.** *Une PCC maximale est une PCC, maximale au regard de sa taille et de sa durée. C'est-à-dire que pour une PCC maximale* $p = (K, k, l, f)$*, avec* $K = \{u_1, \ldots, u_k\}$*, alors :*

$$\nexists p' = (K \cup \{u_{k+1}\}, k+1, l, f) \quad et \tag{4.1}$$

$$\nexists p'' = (K, k, l+1, f) \quad et \tag{4.2}$$

$$\nexists p''' = (K, k, l+1, f+1) \tag{4.3}$$

Une PCC maximale $p = (K, k, l, f)$ est donc maximale au regard de l'inclusion. En effet, il n'existe pas d'ensemble de nœuds plus grand que K et contenant K connecté aux mêmes pas de temps que les nœuds de K (condition 4.1). Une PCC maximale $p = (K, k, l, f)$ est aussi maximale au regard de sa durée. En effet, l'ensemble K est connecté du pas de temps $f - l + 1$ au pas de temps f inclus. Cet ensemble n'est pas connecté au pas de temps $f - l$ (condition 4.2) ni au pas de temps $f + 1$ (condition 4.3). Dans la suite de ce document, nous ne considèrerons que des PCC maximales et par abus de langage le terme PCC sera utilisé pour désigner des PCC maximales.

Définition 12 (Composante connexe persistante vivante)**.** *On dit qu'une composante connexe persistante* $p = (K, k, l, f)$ *est vivante à une date* θ *si* $f - l + 1 \leqslant \theta \leqslant f$*.*

Dans le contexte statique, une composante connexe est définie comme étant un ensemble maximal au regard de l'inclusion. Par analogie à cette propriété, notre objectif est d'identifier les composantes connexes persistantes qui sont les plus importantes en termes de taille et de durée.

Nous introduisons donc la notion de *dominance* d'une composante connexe persistante. Une PCC dominante est définie comme suit :

Définition 13 (Composante connexe persistante dominante)**.** *La composante connexe persistante* $p = (K, k, l, f)$ *est une PCC dominante si et seulement si pour chaque PCC* $p' = (K', k', l', f') \neq p$*,*

$$k > k' \; ; \; l \geq l' \quad ou \tag{4.4}$$

$$l > l' \; ; \; k \geq k' \quad ou \tag{4.5}$$

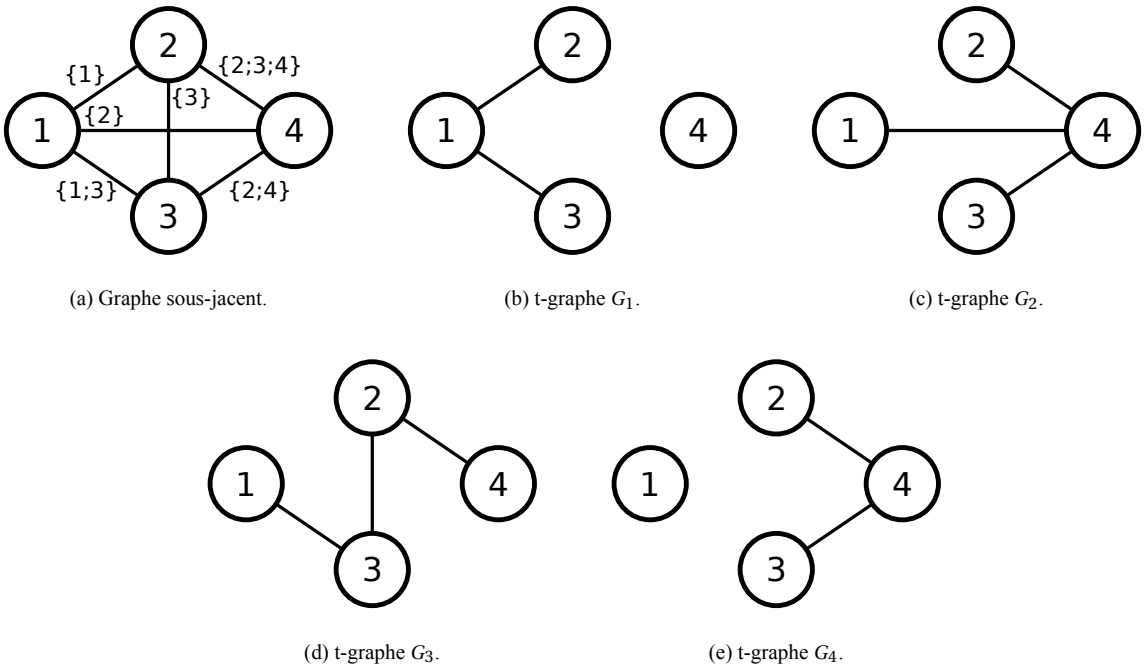
$$k = k' \; ; \; l = l' \; ; \; f < f' \quad ou \tag{4.6}$$

$$k = k' \; ; \; l = l' \; ; \; f = f' \; ; \; K \prec K' \tag{4.7}$$

Nous cherchons à optimiser deux critères : la taille et la durée des composantes. C'est la raison pour laquelle nous souhaitons construire un front de Pareto avec toutes les composantes non-dominées telles que définies par la définition 13.

On définit la dominance par la taille d'une composante (condition 4.4) et sa durée (condition 4.5). Cela fait encore beaucoup de composantes non dominées. On donne donc priorité aux composantes qui apparaissent en premier. La condition 4.6 s'assure donc que les composantes qui vivent avant dominent les composantes qui vivent après. Si deux composantes ont la même taille, la même durée, et la même date de fin, alors on considère un ordre arbitraire sur l'ensemble des nœuds (qui peut être l'ordre lexicographique par exemple) pour qu'une des deux composantes domine l'autre. C'est le rôle de la condition 4.7.

L'objectif que nous nous sommes fixés est d'identifier, dans un graphe dynamique, toutes les composantes connexes persistantes non-dominées telles que définies dans la définition 13.



(a) Graphe sous-jacent. (b) t-graphe G_1. (c) t-graphe G_2.

(d) t-graphe G_3. (e) t-graphe G_4.

Figure 4.4 : Graphe dynamique sur 4 pas de temps.

4.2.3 Exemple

Afin que les définitions de composantes connexes persistantes et de dominance soient claires, observons un exemple de graphe dynamique. La figure 4.4 présente un graphe dynamique de 4 nœuds sur 4 pas de temps. La figure 4.4a représente le graphe sous-jacent, avec sur chaque arête l'ensemble des pas de temps auxquels l'arête est présente. Les figures 4.4b à 4.4e montrent chacun des 4 t-graphes.

Ce graphe n'est pas connexe tout au long de l'intervalle d'étude. On notera que s'il l'était, cela aurait peu d'intérêt de l'observer puisqu'il ne présenterait qu'une seule composante connexe persistante de taille 4 et de durée 4.

Le t-graphe G_1 a deux composantes connexes. Une première de taille 3 composée de l'ensemble de nœuds $\{1;2;3\}$ et une seconde de taille 1 composée de l'ensemble $\{4\}$.

Les t-graphes G_2 et G_3 ont tous les deux une seule composante connexe composée de l'ensemble de nœuds $\{1;2;3;4\}$.

Le t-graphe G_4 a deux composantes connexes, le singleton $\{1\}$ et l'ensemble $\{2;3;4\}$.

Les nœuds 1, 2 et 3 sont connectés depuis le premier pas de temps et jusqu'au troisième inclus. La connexion entre ces trois nœuds n'est pas directe. Ces nœuds seuls ne forment pas une composante connexe dans chacun des trois premiers pas de temps du graphe. En effet, au pas de temps 2, le sous-graphe induit par ces trois nœuds est un stable car ils sont connectés via le nœud 4 qui est en dehors de l'ensemble. Ces nœuds forment une composante connexe persistante $p_1 = (\{1;2;3\},3,3,3)$ dans le sens donné par la définition 10.

De la même manière, les nœuds 2, 3 et 4 sont connectés depuis le deuxième pas de temps et jusqu'à l'horizon de temps du graphe. Ils forment aussi une PCC au sens de la définition 10 que l'on note $p_2 = (\{2;3;4\},3,3,4)$.

Ce graphe est connexe pendant deux pas de temps consécutifs (pas de temps 2 et 3). Cela signifie donc que l'ensemble des nœuds du graphe forme une PCC sur ces deux pas de temps. On note $p_3 =$

$(\{1;2;3;4\},4,2,3)$.

On remarque également que les nœuds 2 et 3, bien que connectés directement par une arête seulement au troisième pas de temps, restent connectés par un chemin pendant toute la durée de l'intervalle d'étude. Ces deux nœuds forment donc aussi une composante connexe persistante que l'on note $p_4 = (\{2;3\},2,4,4)$.

Chacune des composantes connexes persistantes que l'on vient de citer est bien maximale d'après la définition 11. En effet, on ne peut pas y ajouter un nœud ou les prolonger sur un pas de temps supplémentaire.

On peut les comparer grâce au critère de dominance défini dans la définition 13. Nous pouvons de cette manière déterminer quelles sont les PCC non-dominées du graphe que l'on étudie.

La composante p_3 est plus grande que toutes les autres. De la même manière, la composante p_4 dure plus longtemps que toutes les autres composantes.

Les composantes p_1 et p_2 quant à elles, ont la même taille et la même durée. En les comparant, on observe que p_1 se termine avant p_2. D'après la condition 4.6 de la définition 13, p_2 est dominée par p_1.

En résumé, ce graphe a trois PCC non-dominées, p_1, p_3 et p_4. Ce sont donc ces composantes là que l'on souhaite pouvoir identifier dans le graphe.

Contrairement au cas statique où la décomposition du graphe en composantes connexes forme une partition des nœuds du graphe, on remarque que ce n'est plus une propriété valide dans le cas dynamique. En effet, les nœuds peuvent faire partie de plusieurs composantes connexes persistantes différentes, y compris à la même date.

Par exemple, le nœud 2 est présent dans chacune des PCC identifiées. Les PCC peuvent aussi être vivantes à la même date avec des nœuds communs. Par exemple p_1 et p_4, qui contiennent toutes les deux le nœud 2, existent en même temps sur les trois premiers pas de temps du graphe.

Afin de visualiser le chevauchement des PCC, nous avons représenté les PCC non-dominées du graphe de la figure 4.4 par un graphique en figure 4.5. La composante p_1 est représentée en orange, p_3 est en bleu et p_4 en vert. Les nœuds contenus dans les composantes sont en ordonnée et les pas de temps de présence des composantes sont en abscisse. Il est très clair avec ce schéma que les PCC ne sont pas disjointes et qu'un nœud peut appartenir à plusieurs composantes.

Néanmoins, comme nous le verrons plus tard, le nombre de composantes connexes persistantes non-dominées est malgré tout polynomialement borné.

4.2.4 Comparaison avec la littérature

Nous nous intéresserons à des ensembles de sommets qui restent connectés pendant un certain intervalle de temps, comme une extension naturelle des composantes connexes dans des graphes statiques qui correspondent à des ensembles de sommets connectés. Il s'agit de conserver les ensembles qui restent connectés dans la durée et non pas d'identifier les nœuds accessibles depuis un nœud donné. Dans cette optique, l'absence de temps de traversée sur les arêtes est pertinent. En cela, nos travaux sur les composantes connexes persistantes sont différents de la plupart des travaux sur la connexité dans les graphes dynamique que l'on trouve dans la littérature.

À notre sens, il est important de noter que le fait que deux nœuds soient accessibles entre eux à un moment donné de l'intervalle d'étude ne les rend pas accessibles à n'importe quel moment. Dans le cas d'un modèle de graphe sans temps de trajet sur les arêtes, la définition de Bhadra et Ferreira (2003) qui considère que deux nœuds sont dans la même composante dès lors qu'il existe un trajet entre eux masque le fait que ces deux nœuds peuvent être inaccessibles pendant quasiment toute la durée de l'intervalle.

D'après notre définition des PCC, deux nœuds sont dans la même PCC s'ils sont connectés directement ou via un chemin sur plusieurs t-graphes successifs. Donc pour n'importe quelle paire de nœuds

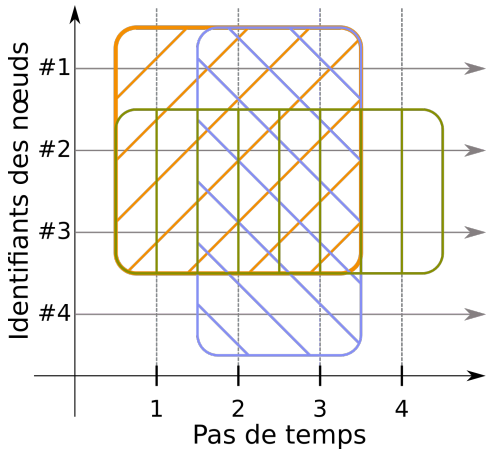
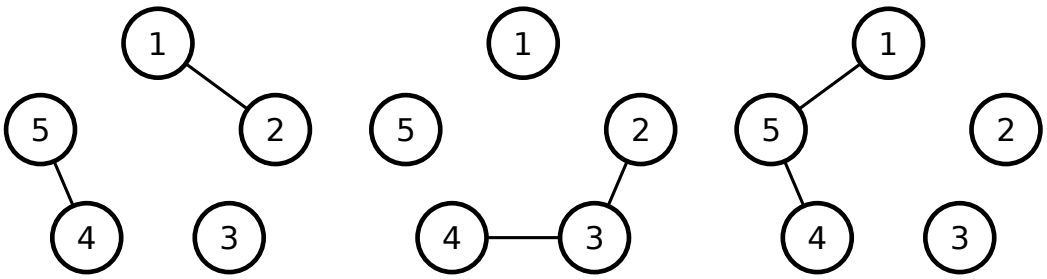


Figure 4.5 : Représentation graphique des composantes connexes persistantes du graphe de la figure 4.4. Chaque surface colorée représente une PCC en recouvrant les nœuds qui la composent et les pas de temps sur lesquels elle est présente.



(a) G_1 (b) G_2 (c) G_3

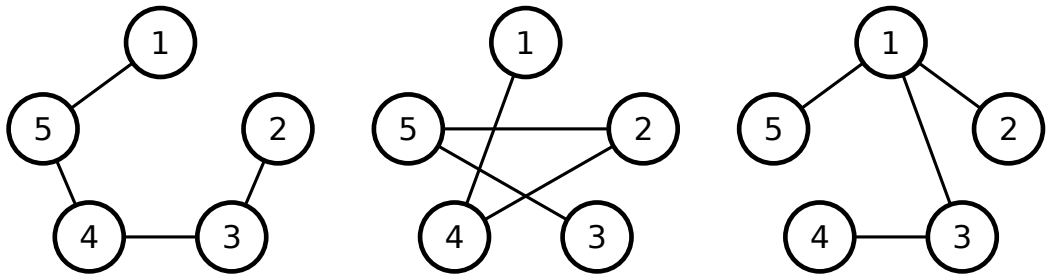
Figure 4.6 : Graphe dynamique de 5 nœuds sur 3 pas de temps.

dans la PCC, il existe un chemin entre eux. Une PCC fait alors toujours partie d'une composante fortement connexe ouverte telle que définie par Bhadra et Ferreira (2003) (voir section 4.1).

À l'inverse, il est possible d'avoir une composante fortement connexe (ouverte ou fermée) au sens de Bhadra et Ferreira et de ne pas avoir de composante connexe persistante. C'est ce que montre la figure 4.6. Dans ce graphe sur 3 pas de temps, on trouve une seule composante fortement connexe fermée (il existe un trajet dans chaque sens entre chaque paire de nœuds du graphe), mais il n'existe aucune PCC d'une durée au moins 2.

Notre définition des PCC, tout comme les travaux de Casteigts et al. (2015) et Akrida et Spirakis (2019), ne s'appuie pas sur les trajets définis par Bui-Xuan et al. (2003). Néanmoins nos travaux diffèrent.

Lorsqu'un graphe est τ-intervalle connexe, comme décrit par Casteigts et al. (2015), alors ce graphe est nécessairement connexe sur $\mathcal{T}$. Ce graphe a donc une composante connexe persistante formée des n nœuds du graphe et qui dure pendant tout l'intervalle d'étude $\mathcal{T}$. La réciproque est bien sûr fausse. Même si un graphe est connexe pendant tout l'intervalle d'étude, puisque la connexité n'est pas forcément assurée via les mêmes arêtes, alors il ne sera pas τ-intervalle connexe pour $\tau > 1$. Prenons par exemple le graphe dynamique de la figure 4.7. Ce graphe est connexe sur tout l'horizon de temps. D'après notre définition des composantes connexes persistantes, il a une seule PCC de 5 nœuds et qui dure 3 pas de temps. En revanche, lorsque l'on applique la définition de τ-intervalle connexité, pour



(a) G_1 (b) G_2 (c) G_3

Figure 4.7 : Graphe dynamique de 5 nœuds sur 3 pas de temps.

$\tau = 2$, l'intersection entre G_1 et G_2 forme un stable, tout comme l'intersection entre G_2 et G_3. Ce graphe n'est pas 2-intervalle connexe.

Bien qu'ils ne définissent pas à proprement parler de « composantes connexes », il est possible d'utiliser la définition de connexité de Casteigts et al. pour décrire ce qu'on pourrait appeler des « composantes τ-intervalle connexes ». Il s'agirait d'un ensemble de nœuds V' (sous-ensemble des nœuds du graphe) tel que le sous-graphe dynamique engendré par V' est τ-intervalle connexe.

Akrida et Spirakis (2019) s'intéressent à des ensembles de nœuds connectés entre eux, ce qui est proche de notre définition des PCC. Cependant, ils considèrent un modèle avec le temps continu. De ce fait, les graphes sur lesquels ils travaillent ne peuvent pas être vus comme une succession de graphes statiques. Il est donc difficile de comparer nos travaux.

Observons le graphe de la figure 4.8. Il s'agit d'un graphe dynamique de 9 nœuds sur 4 pas de temps. Ce graphe nous permet de comparer les composantes au sens de Bhadra et Ferreira (2003) et de Casteigts et al. (2015) avec nos composantes connexes persistantes.

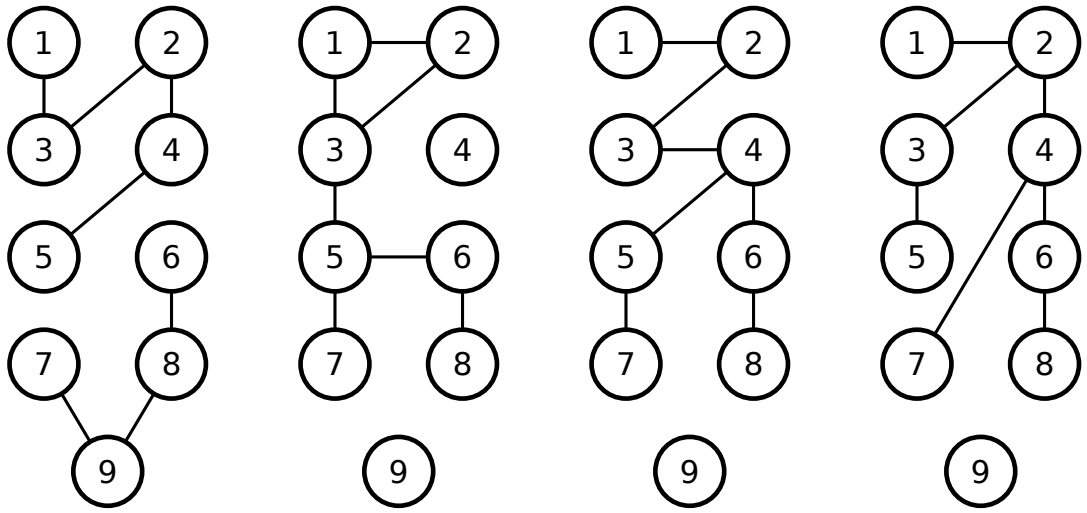
Dans ce graphe, l'ensemble de nœuds $V_1 = \{1\ ;\ 2\ ;\ 3\ ;\ 4\ ;\ 5\ ;\ 6\ ;\ 7\ ;\ 8\}$ forme une composante fortement connexe fermée au sens de Bhadra et Ferreira (2003). L'ensemble de nœuds $V_2 = \{6\ ;\ 7\ ;\ 8\ ;\ 9\}$ forme aussi une composante fortement connexe fermée. En effet, il existe un trajet dans chaque sens dans le graphe entre chaque paire de nœuds de l'ensemble V_1. Il existe un trajet depuis le nœud 9 vers tous les autres nœuds du graphe, mais il existe un trajet vers le nœud 9 seulement depuis les nœuds de l'ensemble V_2.

Si l'on considère à présent la définition de connexité donnée par Casteigts et al. (2015), on observe que l'ensemble $V_3 = \{1\ ;\ 2\ ;\ 3\}$ est 2-intervalle connexe. En effet, l'intersection de n'importe quelle paire de t-graphes successifs dans le sous-graphe induit par V_3 donne un graphe connexe. Les ensembles $\{2\ ;\ 3\}$ et $\{6\ ;\ 8\}$ sont tous les deux 4-intervalle connexes car les arêtes $(2;3)$ et $(6;8)$ sont présentes à chaque pas de temps.

Enfin, ce graphe présente 3 composantes connexes persistantes non dominées. L'ensemble V_1 qui formait une composante fortement connexe fermée est une composante connexe persistante non dominée de 8 nœuds et qui dure 2 pas de temps (du troisième au quatrième pas de temps de l'intervalle d'étude). L'ensemble $\{1\ ;\ 2\ ;\ 3\ ;\ 5\ ;\ 6\ ;\ 7\ ;\ 8\}$ $(V_1 \setminus \{4\})$ forme une composante connexe persistante non dominée de 7 nœuds et qui dure 3 pas de temps (du deuxième au quatrième pas de temps de l'intervalle d'étude). Et l'ensemble $\{1\ ;\ 2\ ;\ 3\ ;\ 5\}$ est une composante connexe persistante non dominée de 4 nœuds et qui dure 4 pas de temps, soit la totalité de l'intervalle d'étude.

Cet exemple permet de mettre en lumière les ensembles décrits par ces définitions. On constate de cette manière que ces définitions ne s'intéressent pas aux mêmes ensembles de nœuds.

Contrairement à la plupart des travaux présentés en section 4.1, l'identification des composantes connexes non-dominées dans un graphe dynamique est un problème qui se résout par un algorithme



(a) G_1 (b) G_2 (c) G_3 (d) G_4

Figure 4.8 : Graphe dynamique de 9 nœuds sur 4 pas de temps.

polynomial. La section suivante présente l'algorithme que nous avons mis en place pour cela.

4.2.5 Applications

Dans le contexte statique, la connexité a de nombreux cas d'application. Il en va naturellement de même dans le cas dynamique.

On rappellera que dans le modèle considéré ici, il n'y a aucun type de poids sur les arêtes. Il n'y a donc pas de temps de trajet associés aux arêtes. Un modèle de ce type est particulièrement adapté dans les cas où la dynamique du réseau considéré est lente comparée au temps nécessaire pour effectivement traverser une arête. Autrement dit, ce modèle est adapté lorsque le temps de trajet est négligeable devant le temps nécessaire pour que le graphe évolue.

Dans les réseaux de communication tels que les réseaux adhoc ou les réseaux de capteurs, la transmission d'information se fait de manière quasiment instantanée. Un modèle sans temps de trajet est donc adapté.

Dans ce type de réseaux, une composante connexe persistante est un sous-ensemble qui reste connecté. Lorsqu'il s'agit de communication, cela peut être essentiel. On citera par exemple (Koster et Muñoz, 2009).

Dans les réseaux de transport, la disponibilité de certaines routes peut dépendre du temps. L'inaccessibilité d'une route peut être temporaire, pour cause de travaux par exemple, de nouvelles routes peuvent être créées et d'autres peuvent être fermées à la circulation. Le temps nécessaire pour traverser une arête, de l'ordre de la minute ou de l'heure, peut être considéré comme étant négligeable devant l'évolution du réseau, de l'ordre du mois voire de l'année.

Dans ce contexte, une composante connexe persistante permet de mesurer l'accessibilité de différents lieux sur une période donnée. Démare et al. (2017) utilise par exemple les graphes dynamiques pour modéliser un réseau de transport dans la vallée de la Seine.

Dans les réseaux sociaux, un temps de trajet sur les arêtes n'aurait aucun sens puisque celles-ci représentent la relation entre deux personnes. On pourra voir les travaux de Newman (2003).

Parmi les nombreux travaux existant sur la détection de communautés dans les réseaux sociaux, certains considèrent la dimension temporelle comme (Nguyen et al., 2011). Si l'on suppose que les communautés doivent être connexes pendant un certain intervalle de temps, alors identifier les composantes connexes persistantes telles que nous les avons définies peut aider à identifier les communautés dans le graphe.

4.3 Algorithme PICCNIC

Nous avons développé un algorithme polynomial permettant d'identifier, dans un graphe dynamique, toutes les composantes connexes persistantes non dominées. Étant donné que nous recherchons les composantes les plus importantes au regard de deux critères, la taille et la durée des composantes, notre algorithme donne un ensemble de composantes qui forment un front de Pareto.

Cette section présente l'algorithme PICCNIC (*PersIstent Connected CompoNent InCremental Algorithm* en anglais que l'on traduit par *algorithme incrémental pour composantes connexes persistantes*). Nous déroulons ensuite l'algorithme sur le graphe de la figure 4.4. Nous apportons les preuves de correction et de complexité de l'algorithme.

4.3.1 Description

L'algorithme PICCNIC traite un graphe dynamique dont le modèle est décrit en section 4.2.1. L'objectif est de trouver toutes les composantes connexes persistantes non-dominées de ce graphe au sens de la définition 13.

Il s'agit d'un algorithme incrémental qui traite les t-graphes les uns après les autres par une nouvelle itération. Il présente donc le net avantage de ne pas nécessiter une connaissance complète de l'évolution du graphe à l'avance. On peut découvrir le graphe et son évolution au fur et à mesure de l'exécution de l'algorithme.

L'algorithme PICCNIC est décrit précisément dans l'algorithme 2. Chaque itération est composée de deux étapes distinctes décrites respectivement dans les algorithmes 3 et 4.

Algorithme 2 : PICCNIC

Input : Graphe dynamique G, intervalle d'étude $\mathcal{T} = \{1;\ldots;T\}$
Output : Ensemble des composantes connexes persistantes dominantes

```
/* Loop on the number of instants */
pour chaque θ ∈ {1,...;T+1} faire
    G' ← G_θ ;                          /* Le graphe à la date θ, rien si θ = T+1 */
    CC ← Ensemble des composantes connexes de G' de taille ≥ 2
    si θ = 1 alors
        PCC_c ← CC ;                    /* Prépare la prochaine itération */
        PCC_f ← ∅
        /* Toutes les composantes connexes sont des composantes connexes
           persistantes */
    sinon
        PCC_n ← PICCNIC_Step1 (CC, PCC_c)
        PCC_f ← PICCNIC_Step2 (PCC_n, PCC_c)
        PCC_c ← PCC_n ;                 /* Prépare la prochaine itération */
    fin
fin
retourner PCC_f
```

Algorithme 3 : PICCNIC$_{\text{Step1}}$

Input : CC, PCC$_c$
Output : PCC$_n$, l'ensemble des nouvelles PCC possibles

```
1  PCC_n ← ∅
2  pour chaque c ∈ CC&|c| > 1 faire
3      Pers ← FAUX ;                    /* Indique si c est déjà une PCC ou non */
4      PCC_t ← ∅ ;  /* Contient les PCC dont les ensembles sont présents dans c
        */
5      pour chaque p ∈ PCC_c faire
6          si p = c alors
7              Pers ← VRAI
8              p' ← p
9          sinon
10             p' ← p ∩ c
11         fin
12         si |p'| ≥ 2 alors
13             l (p') ← l (p)+1
14             f (p') ← i
15             ajouterPCC(PCC_t ; p')
16         fin
17     fin
18     si ¬Pers alors
19         l (c) ← 1
20         f (c) ← i
21         ajouterPCC(PCC_t ; c)
           /* Ajoute les nouvelles composantes qui ne sont pas déjà dans PCC_t
               */
22     fin
23     PCC_n ← PCC_n ∪ PCC_t
24 fin
25 retourner PCC_n
```

Le principe général de cet algorithme est de parcourir le graphe pas de temps par pas de temps en gardant en mémoire les composantes connexes persistantes qui existaient au pas de temps précédent puis de les comparer aux composantes connexes statiques qui existent au pas de temps présent. De cette manière on peut identifier les composantes connexes persistantes qui existent toujours, celles qui n'existent plus, celles qui viennent d'apparaitre et celles qui se sont scindées.

Pour cela, nous utilisons dans l'algorithme cinq ensembles qui contiennent des PCC. L'ensemble PCC_c conserve les PCC vivantes au pas de temps précédent. L'ensemble PCC_n contient, à chaque pas de temps θ, les PCC vivantes à la date θ et nécessite pour être construit l'ensemble PCC_t qui est un ensemble temporaire des PCC vivantes à la date θ. L'ensemble PCC_o contient, à chaque pas de temps, les composantes qui viennent de se finir, autrement dit, celles qui étaient présentes au pas de temps précédent et ne le sont plus au pas de temps présent. Et enfin, l'ensemble PCC_f contient les PCC non-dominées, c'est à dire celles qui sont terminées et ne sont dominées par aucune autre PCC terminée.

Chaque itération de l'algorithme traite un pas de temps θ et commence par récupérer les composantes connexes statiques de G_θ, le t-graphe à la date courante (ligne 3). Nous ignorons les composantes de taille 1 parce qu'un nœud étant toujours connecté à lui-même pendant tout l'intervalle d'étude, un graphe

Algorithme 4 : PICCNIC$_{\text{Step2}}$

Input : PCC_n, PCC_c, PCC_f
Output : PCC_f mis à jour

```
1  PCC_o ← PCC_c \ PCC_n ;                          /* Ensemble des PCCs terminées */
2  pour chaque p ∈ PCC_o faire
3      pour chaque p' ∈ PCC_o & p ≠ p' faire
4          si DomLaterEqual (p', p) alors
5              PCC_o ← PCC_o \ {p}
6              STOP
7          sinon
8              si DomLaterEqual (p, p') alors
9                  PCC_o ← PCC_o \ {p'}
10             fin
11         fin
12     fin
13 fin
14 pour chaque p ∈ PCC_o faire
15     pour chaque p' ∈ PCC_f faire
16         si DomEarlier(p', p) alors
17             PCC_o ← PCC_o \ {p}
18             STOP ;                    /* p est dominée par une PCC terminée */
19         sinon
20             si DomLaterEqual(p, p') alors
21                 PCC_f ← PCC_f \ {p'}
22             fin
23         fin
24     fin
25 fin
26 PCC_f ← PCC_f ∪ PCC_o
27 retourner PCC_f
```

Algorithme 5 : DomEarlier

Input : Deux PCC p_1 et p_2 telles que `f(`p_1`)` < `f(`p_2`)`
Output : VRAI si p_1 domine p_2

```
1  si (k (p1) ≥ k (p2)) & (l (p1) ≥ l (p2)) alors
2      retourner VRAI
3  fin
4  retourner FAUX
```

Algorithme 6 : DomLaterEqual

Input : Deux PCC p_1 et p_2 telles que $\texttt{f}(p_1) \geq \texttt{f}(p_2)$
Output : VRAI si p_1 domine p_2

si $(\texttt{k}\ (p_1) > \texttt{k}\ (p_2))$ & $(\texttt{l}\ (p_1) \geq \texttt{l}\ (p_2))$ **alors**
  retourner VRAI
fin
si $(\texttt{k}\ (p_1) \geq \texttt{k}\ (p_2))$ & $(\texttt{l}\ (p_1) > \texttt{l}\ (p_2))$ **alors**
  retourner VRAI
fin
si $(\texttt{k}\ (p_1) = \texttt{k}\ (p_2))$ & $(\texttt{l}\ (p_1) = \texttt{l}\ (p_2))$ & $(\texttt{f}\ (p_1) = \texttt{f}\ (p_2))$ & $(\texttt{K}\ (p_1) \prec \texttt{K}\ (p_2))$ **alors**
  retourner VRAI
fin
retourner FAUX

dynamique contient toujours n composantes connexes persistantes de taille 1 et de durée T. C'est donc inintéressant.

Chaque itération est composée de deux étapes. La première, décrite dans l'algorithme 3, a pour but d'identifier les composantes connexes persistantes vivantes à la date θ courante, c'est-à-dire d'une part celles qui commencent à la date θ, et d'autre part celle qui étaient déjà présentes à la date $\theta - 1$ et qui sont toujours là à la date θ.

Cette étape utilise la fonction `ajouterPCC`. Cette fonction permet d'ajouter une PCC à un ensemble de PCC en vérifiant qu'il n'existe pas déjà dans l'ensemble une PCC ayant le même ensemble de nœuds. Si c'est le cas, alors on conserve dans l'ensemble la plus vieille des deux.

La seconde étape de l'algorithme PICCNIC, décrite dans l'algorithme 4 traite les composantes connexes persistantes qui sont terminées à la date courante θ pour ne conserver que celles qui sont non-dominées par une autre PCC terminée. Pour cela, on utilise les fonctions `DomEarlier` et `DomLaterEqual`, décrites respectivement dans les algorithmes 5 et 6, qui vérifient, pour deux PCC données, si la première domine la seconde.

Enfin, la dernière itération de l'algorithme est particulière puisqu'il n'y a pas de t-graphe à traiter. Cette itération ne réalise que la seconde étape de l'algorithme et permet de traiter les composantes connexes persistantes qui étaient encore vivantes à l'horizon de temps.

4.3.2 Exemple

Observons comment s'exécute l'algorithme 2 sur le graphe de la figure 4.4. Nous décrivons ici chaque itération.

La première itération de l'algorithme s'intéresse à G_1, le premier t-graphe (voir figure 4.4b). Ce graphe est divisé en deux composantes connexes au sens statique. La composante $\{4\}$ est un singleton et n'est donc pas prise en compte. Il n'y a qu'une seule composante à traiter, $\{1;2;3\}$.

L'ensemble des composantes courantes PCC_c est initialisé avec cette composante connexe et $PCC_c = \{(\{1;2;3\},3,1,1)\}$. L'ensemble des PCC finales reste vide pour l'instant, $PCC_f = \emptyset$.

Pour la seconde itération de l'algorithme, le t-graphe courant est G_2 (voir figure 4.4c). Il a une seule composante connexe $\{1;2;3;4\}$.

À la fin de l'itération, l'ensemble de PCC courantes $PCC_c = \{(\{1;2;3\},3,2,2), (\{1;2;3;4\},4,1,2)\}$. La première composante était déjà présente au pas de temps précédent, donc sa durée est maintenant 2. L'ensemble PCC_f reste vide puisqu'aucune PCC ne s'est encore terminée.

Dans la troisième itération, le t-graphe courant est G_3 (voir figure 4.4d). Comme au pas de temps précédent, le graphe est connexe au troisième pas de temps, donc il n'y a qu'une seule composante connexe.

À la fin de l'itération, on a $PCC_c = \{(\{1;2;3\},3,3,3),\ (\{1;2;3;4\},4,2,3)\}$. Les deux composantes étaient déjà présentes au pas de temps précédent, donc elles ont vieilli d'une unité. Et aucune nouvelle composante n'est apparue à ce pas de temps. Aucune composante ne s'est encore terminée, donc on a encore $PCC_f = \emptyset$.

La quatrième itération traite le t-graphe G_4 (voir figure 4.4e). Il est composé de deux composantes connexes : $\{1\}$ et $\{2;3;4\}$. On ne s'intéresse qu'à la seconde puisque l'on ignore les singletons.

Par construction, on ajoute dans PCC_t d'abord la PCC $(\{2;3\},2,4,4)$ qui vient de l'ensemble $\{1;2;3\}$ qui est connecté depuis la premier pas de temps. On y ajoute ensuite la PCC $(\{2;3;4\},3,3,4)$ qui vient de l'ensemble $\{1,2,3,4\}$ qui est connecté depuis le deuxième pas de temps. On trouve ensuite la PCC $(\{2,3,4\},3,1,4)\}$ qui vient de la composante connexe $\{2;3;4\}$ de G_4. Lorsque l'on veut ajouter cette PCC à PCC_t avec la fonction `ajoutePCC`, on trouve qu'il existe déjà une PCC avec le même ensemble de nœuds. En comparant les deux on constate que celle qui y était déjà est vieille de 3 pas de temps alors que celle que l'on tente d'ajouter n'est vieille que d'un pas de temps, on conserve donc celle qui y était déjà sans ajouter la nouvelle. On peut ensuite ajouter tout le contenu de PCC_t dans PCC_n.

Au début de l'étape 2 de cette quatrième itération, on a donc $PCC_n = \{(\{2;3\},2,4,4)\ ;\ (\{2;3;4\},3,3,4)\}$. Et donc à la fin de la quatrième itération, on a $PCC_c = \{(\{2;3\},2,4,4)\ ;\ (\{2;3;4\},3,3,4)\}$.

L'ensemble des PCC terminées non-dominées $PCC_f = \{(\{1;2;3\},3,3,3)\ ;\ (\{1;2;3;4\},4,2,3)\}$. En effet, ces deux composantes sont terminées puisque le nœud 1, qui est présent dans les deux PCC, est maintenant déconnecté du reste du graphe. Ces deux PCC sont non dominées puisque l'une des deux dure plus longtemps que l'autre mais contient moins de nœuds, et la seconde dure moins longtemps mais contient plus de nœuds. Elles sont donc toutes les deux conservées dans l'ensemble PCC_f.

Dans la cinquième et dernière itération, il n'y a pas de t-graphe à traiter. Il s'agit ici de traiter les PCC qui se sont terminées à l'horizon de temps du graphe, ici à la date 4. On exécute donc seulement l'étape 2 de l'algorithme (décrite dans l'algorithme 4).

À la fin de l'itération, $PCC_f = \{(\{1;2;3\},3,3,3)\ ;\ (\{1;2;3;4\},4,2,3)\ ;\ (\{2;3\},2,4,4)\}$. La dernière composante dure 4 pas de temps, en effet, les nœuds 2 et 3 sont connectés pendant tout l'intervalle d'étude. On peut aussi noter que la PCC $(\{2;3;4\},3,3,4)\}$ a été supprimée de PCC_f parce qu'elle est maintenant dominée par $(\{1;2;3\},3,3,3)$ qui a le même nombre de nœuds et dure le même temps mais termine avant.

Le graphe dynamique de la figure 4.4 contient 3 composantes connexes persistantes non dominées : une de taille 4 et de durée 2, une de taille 3 et de durée 3 et une de taille 2 et de durée 4.

4.3.3 Correction et complexité

Cette section se concentre sur les preuves de correction et de complexité de l'algorithme PICCNIC (algorithme 2).

Théorème 11 (Correction de PICCNIC)**.** *Pour un graphe dynamique G sur l'intervalle d'étude $\mathcal{T}$, l'algorithme PICCNIC donne l'ensemble des composantes connexes persistantes non dominées, dans le sens de la définition 13.*

Démonstration. Afin de prouver la correction de l'algorithme 2, nous devons montrer que :

1. À chaque pas de temps $\theta \leq T$, chaque PCC dominante (au sens de la définition 13) qui termine au plus tard à la date θ est présente dans $PCC_f \cup PCC_c$.
2. À la fin de l'algorithme, seules les PCC dominantes sont présentes dans PCC_f.

Prouvons d'abord la première propriété. Pour $\theta = 1$, cette propriété est triviale puisque l'ensemble PCC_c contient toutes les composantes connexes de G à la date 1.

Supposons maintenant que la propriété est vraie pour un certain θ tel que $1 \leq \theta \leq T$, et intéressons-nous à l'itération $\theta + 1$. Soit $p = (K, k, l, f)$ une PCC dominante en $\theta + 1$.

Supposons d'abord que p n'est pas dominante pour chaque $\theta' \leq \theta$, d'où sa date de fin $f = \theta + 1$. Si p dure 1 pas de temps, alors c'est nécessairement une composante connexe de $G_{\theta+1}$. Dans l'algorithme, la variable booléenne qui lui est associée `Pers` est fausse et p est directement incluse dans PCC_t et donc ensuite dans PCC_n, puis est ajoutée à PCC_c.

Si la durée de p est strictement supérieure à 1, supposons d'abord que $p \in PCC_c$ au début de l'itération. Alors p doit être incluse dans une composante connexe de $G_{\theta+1}$. Dans ce cas, p est ajoutée à PCC_t et la fonction `ajouterPCC` prend soin d'avoir la PCC la « plus vieille » dans PCC_t. Puis p est ajoutée à PCC_n et ensuite à PCC_c. Même si p termine à la date $\theta + 1$, elle n'est pas supprimée après les tests de dominance parce qu'elle est supposée dominante par hypothèse.

Dans le cas où p n'est pas dans l'ensemble PCC_c au début de l'itération, cela veut dire qu'à toutes les itérations précédentes, p n'était pas conservée par l'algorithme (comme on ne retire jamais de composantes de PCC_c avant qu'elle ne termine). Elle était cependant incluse dans une composante connexe dans chacun des t-graphes précédents. Cela veut dire qu'à chacune de ces dates, p était strictement incluse (en termes d'ensemble de nœuds) dans une autre PCC, que l'on nomme q, présente dans PCC_c au moins à la date θ. Si q n'existait pas, alors p aurait été ajoutée à PCC_c avant et serait apparue comme l'intersection d'une PCC de PCC_c avec une composante connexe statique. La durée de q est égale à la durée de p avant la date $\theta + 1$ (de telle manière que p n'est pas dominée par q à $\theta + 1$). La PCC q n'est pas incluse dans une composante connexe à la date $\theta + 1$ puisque p est alors dominante, et l'intersection de q avec une composante connexe c à la date $\theta + 1$ est p (une intersection plus grande impliquerait un sous-ensemble de q plus grand présent à la date $\theta + 1$, mais p est dominante).

On en déduit donc que p est incluse dans PCC_t et donc dans PCC_n et ensuite dans PCC_c à l'itération $\theta + 1$. Si p est dominante pour $\theta' \leq \theta$, alors par hypothèse d'induction, p est présente à la date θ. Soit elle termine au plus tard à la date $\theta + 1$ et p reste dans PCC_f (elle n'est pas retirée par l'algorithme puisqu'elle est dominante), soit elle n'est pas terminée et reste dans PCC_c. Dans ce cas, p est incluse dans une composante connexe de G. Par induction, le résultat est vrai aussi pour $\theta = T$. À la dernière itération, seules les PCC dominées sont supprimées, donc p est dans l'ensemble PCC_f renvoyé par l'algorithme.

Prouvons à présent la seconde propriété. Supposons qu'il existe une PCC $p = (K, k, l, f)$ qui est présente avant la dernière itération et qui est dominée par une PCC $p' = (K', k', l', f')$.

Nous venons de prouver que p' est conservée par l'algorithme à l'itération T. Si $p \in PCC_c$, lors de la dernière itération, p est ajoutée à PCC_o comme l'ensemble CC est vide. Elle est ensuite comparée, selon le critère de dominance, aux PCC de PCC_c puis à celles de PCC_f. Elle sera donc comparée à p' et supprimée.

Si $p \in PCC_f$ et $p' \in PCC_c$ à l'itération T, alors pour la même raison, p' sera comparée selon le critère de dominance à p et p sera supprimée.

Supposons maintenant que $p \in PCC_f$ et $p' \in PCC_f$. Elles ont été comparées quand la dernière a été terminée, et si elles se sont terminées en même temps elles sont comparées par l'algorithme lorsqu'elles sont dans l'ensemble PCC_o. De ce fait, il est impossible que p et p' soient toutes les deux présentes. □

Maintenant que l'on a prouvé que l'algorithme PICCNIC est correct, montrons sa complexité. Nous passons, pour cela, par plusieurs lemmes.

Lemme 12 (Borne de $|PCC_c|$)**.** *À la fin de n'importe quelle itération $1 \leq \theta \leq T$, la cardinalité de l'ensemble PCC_c est bornée par $n - NbCC(G_\theta)$, où $NbCC(G_\theta)$ est le nombre de composantes connexes de G_θ.*

Démonstration. L'ensemble PCC_n ne contient aucunes PCC en double, c'est-à-dire qu'il ne contient pas deux PCC telles qu'elles auraient le même ensemble de sommets. Ceci est assuré par la construction de PCC_n puisqu'on y ajoute les PCC qui étaient présentes dans PCC_t.

L'ensemble PCC_t est construit pour chaque composante connexe statique du t-graphe G_θ courant. Il ne contient pas de doubles puisqu'on s'en assure au moment d'y ajouter les éléments grâce à la fonction `ajouterPCC`. Tous ses éléments sont ensuite ajoutés à PCC_n.

Les intersections entre une PCC de PCC_c et une composante connexe statique peuvent avoir le même ensemble de sommets uniquement lorsque l'on considère la même composante connexe statique. Dès lors que l'on change de composante connexe statique courante c, les intersections donneront forcément d'autres ensembles de sommets. Donc les nouvelles PCC ajoutées à PCC_t et ensuite à PCC_n ne contiennent pas les mêmes nœuds que des PCC déjà dans PCC_n.

Donc il n'y a pas d'ensembles doublons dans PCC_n. Et comme PCC_c est construit à partir de l'ensemble PCC_n à la fin de chaque itération, il ne contient pas de doublons non plus.

Soient $p = (K, k, l, i)$ et $p' = (K', k', l', i')$ deux éléments de PCC_n à la fin d'une itération θ tels que $K \neq K'$. Supposons que p commence avant p'. On sait que K et K' ont une cardinalité au moins égale à 2. Ces deux ensembles sont forcément inclus dans des composantes connexes de G_θ.

S'ils sont inclus dans des composantes connexes différentes, alors K et K' sont disjoints.

Supposons maintenant qu'ils soient inclus dans la même composante c de G_θ et que $K \cap K' \neq \emptyset$. Si l'un n'est pas inclus dans l'autre alors $K'' = K' \cup K$ est aussi inclus dans c. De plus, une PCC p'' associée à l'ensemble K'' est présente depuis la date $\theta_{p''}$ où les deux PCC p et p' étaient présentes en même temps pour la première fois. Puisque p commence avant p', cela signifie que p'' apparait à la même date que p'. Mais c'est impossible puisque K'' contient strictement K et K'. Alors p' n'apparait pas. Nous avons donc $K \subset K'$ (si $K' \subset K$ alors p' ne peut pas être incluse dans PCC_n).

On peut en déduire que les PCC présentes dans l'ensemble PCC_c sont soit strictement incluses soit disjointes.

Il est facile de vérifier par induction que le nombre de sous-ensembles non singletons strictement inclus ou disjoints d'un ensemble Ω donné est borné par la cardinalité de cet ensemble moins 1, quelle que soit la manière dont ces sous-ensembles sont choisis.

Il est évident que si l'ensemble a 2 éléments seulement, alors il n'y a qu'un seul sous-ensemble qui vérifie les hypothèses.

Considérons à présent un ensemble Ω ayant $\omega > 2$ éléments. Supposons qu'il ait $\omega - 1$ sous-ensembles vérifiant les hypothèses tel que ce nombre soit maximal (c'est-à-dire qu'il soit impossible de créer un autre sous-ensemble sans violer les contraintes de stricte inclusion ou de disjonction). Lorsqu'un élément e est ajouté à l'ensemble Ω, on peut soit conserver les mêmes sous-ensembles, et dans ce cas on a toujours $\omega - 1$ sous-ensembles, ou bien on peut créer un nouveau sous-ensemble en faisant une union entre $\{e\}$ et un sous-ensemble déjà existant. Dans ce cas, nous obtenons ω sous-ensembles. Puisque deux ensembles sont soit strictement inclus soit disjoints, il est impossible de créer un autre sous-ensemble, quelle que soit la manière dont ceux-ci avaient été initialement choisis.

On en déduit donc qu'un ensemble Ω ayant ω éléments a au plus $\omega - 1$ sous-ensembles tels que chaque sous-ensemble a une cardinalité supérieure ou égale à 2 et deux sous-ensembles sont strictement inclus ou disjoints.

G_θ est divisé en α composantes connexes de tailles $k_1, \ldots, k_i, \ldots, k_\alpha$. En appliquant le raisonnement précédent sur chaque composante connexe, on en déduit que chacune contient au plus $k_i - 1$ sous-ensembles. On obtient ainsi le résultat recherché. □

Lemme 13 (Borne de $|PCC_f|$)**.** *À la fin de n'importe quelle itération $1 \leq \theta \leq T$, la cardinalité de l'ensemble PCC_f est bornée par $min\{n-1, \theta\}$.*

Démonstration. C'est trivialement vrai pour $\theta = 1$ puisque l'ensemble est vide à ce moment-là.

Supposons à présent que c'est vrai pour une date θ telle que $1 \leq \theta \leq T$. Le théorème 11 dit que deux composantes liées par une relation de dominance ne peuvent pas se trouver toutes les deux dans PCC_f.

Supposons qu'il existe deux composantes p et p' de même durée l présentes dans PCC_f à la date θ. L'une des deux domine forcément l'autre. Donc pour une durée donnée telle que $1 \leq l \leq T$, il existe au plus une PCC de durée l dans PCC_f, soit au plus θ éléments dans PCC_f.

De la même manière, deux PCC contenues dans PCC_f sont de tailles différentes. Donc PCC_f contient au plus $n-1$ éléments (puisque les singletons sont ignorés). □

Lemme 14 (Complexité d'une itération de PICCNIC)**.** *La complexité d'une itération de l'algorithme 2, y compris la dernière, est* $O(n^2)$.

Démonstration. On notera tout d'abord que tous les ensembles de PCC utilisés dans une itération ont au plus n éléments. Les lemmes 12 et 13 le prouvent pour les ensembles PCC_c et PCC_f. C'est vrai pour PCC_n puisque cet ensemble est égal à PCC_c à la fin d'une itération. C'est vrai aussi pour PCC_o puisque cet ensemble est inclus dans l'ensemble PCC_c de l'itération précédente. C'est évident pour CC de par sa définition. On note $NbCC(G_\theta)$ le nombre de composantes connexes du t-graphe G_θ.

En utilisant des structures de données appropriées, comme des arbres binaires de recherche, il est possible de réaliser les opérations d'ajout ou de suppression d'élément en temps $O(\log(n))$ et d'effectuer des comparaisons et des intersections de deux ensembles en temps linéaire.

Dans chaque itération de l'algorithme, on calcule les composantes connexes statiques du t-graphe courant G_θ. Cette opération se fait en $O(m+n)$, soit $O(n^2)$.

La première itération de l'algorithme se contente de mettre dans l'ensemble PCC_c les composantes connexes statiques du premier t-graphe. Cela se fait en temps de l'ordre de $n \cdot \log(n)$.

La première étape de l'algorithme PICCNIC (algorithme 3) contient une boucle sur l'ensemble des composantes connexes CC ($|CC| \leqslant n$) de G_θ dans laquelle se trouve une boucle sur l'ensemble PCC_c ($|PCC_c| \leqslant n$). Chaque élément de chaque ensemble est lui-même borné par n, mais les éléments de CC forment une partition des nœuds du graphe, donc en réalité, un ensemble C_α de CC est de taille k_α tel que $\sum_\alpha k_\alpha = n$.

Les opérations de comparaison (ligne 6 de l'algorithme 3) et d'intersection (ligne 10 de l'algorithme 3) entre c de l'ensemble CC et p de l'ensemble PCC_c se font en $O(\min\{|c| ; |p|\})$. Pour chaque composante C_α de CC, chacune de ces opérations coute $O(n \cdot k_\alpha)$. Donc au total, cela coute $O(n \cdot (k_1 + \cdots + k_{|CC|}))$, soit $O(n^2)$.

Par le raisonnement de la preuve du lemme 12, on sait que les ensembles de PCC_t sont soit disjoints soit strictement inclus les uns dans les autres. Les PCC de PCC_t ne contiennent que des nœuds de la composante C_α courante de taille k_α. Cet ensemble ne contient pas de doublons puisque le rôle de la fonction d'ajout `ajouterPCC` (utilisée aux lignes 15 et 21) est d'ajouter des PCC sans créer de doublons. Donc la taille de l'ensemble PCC_t est borné par k_α

On a rappelé ci-dessus que l'ajout d'un élément dans PCC_t pouvait se faire en temps logarithmique en la taille de l'ensemble. La fonction `ajouterPCC` compare les âges des composantes identiques en temps constant donc l'ajout d'une PCC à PCC_t avec la fonction `ajouterPCC` se fait en temps $\log(k_\alpha)$.

L'instruction ligne 15 coute $O(\log(k_\alpha))$ pour chaque p dans PCC_c. Donc cette instruction coute $O(n \cdot \log(k_\alpha))$ pour chaque C_α dans CC. Donc au total, sur toute l'exécution de l'algorithme, cela coute $O(n \cdot (\log(k_1) + \cdots + \log(k_{|CC|})))$. Comme $\log(k_\alpha) < k_\alpha$ alors la complexité totale de l'instruction ligne 15 est bornée par $O(n^2)$.

L'instruction ligne 21 coute $O(\log(k_\alpha))$ pour chaque C_α dans CC. Donc au total, sur toute l'exécution de l'algorithme, cela coute $O(\sum_\alpha \log(k_\alpha))$, ce qui est borné par $O(n)$.

Enfin, à la ligne 23, on ajoute tous les éléments de PCC_t dans PCC_n. La taille de l'ensemble PCC_n est bornée par n donc l'ajout d'un élément se fait en $O(\log(n))$. Comme l'ensemble PCC_t contient au plus k_α éléments, les ajouter tous coute $O(k_\alpha \cdot \log(n))$ pour chaque composante C_α dans CC. Donc au total, pour toute l'exécution de l'algorithme, cela coute $O(\sum_\alpha k_\alpha \cdot \log(n))$, ce qui est borné par $O(n\log(n))$.

Au total, l'algorithme 3 coute $O(n^2)$.

Dans la seconde étape de PICCNIC (algorithme 4), les deux boucles contiennent au plus deux tests de domination réalisés dans une boucle interne en temps constant.

Cette étape de PICCNIC retire aussi des éléments de PCC_o ou PCC_f. Puisque la taille de chacun de ces ensembles est bornée par n, alors il n'est pas possible de leur retirer plus de n éléments. Donc ces opérations coutent au total $O(n \cdot \log(n))$.

Le nombre de boucles internes est au plus $|PCC_o|^2$ pour la première boucle et $|PCC_o| \cdot |PCC_f|$ pour la seconde. Les deux boucles ont donc une complexité de $O(n^2)$.

Enfin, cette étape contient des opérations de différence et d'union entre ensembles de PCC, en dehors de toute boucle. Ces opérations sont réalisées en $O(n\log(n))$.

La complexité de l'algorithme 4 est aussi $O(n^2)$.

Enfin, l'instruction ligne 10 de l'algorithme 2 est une opération en temps constant. Donc une itération de PICCNIC coute $O(n^2)$. □

Théorème 15 (Complexité de PICCNIC)**.** *La complexité de l'algorithme PICCNIC est $O(n^2 \cdot T)$, où n est le nombre de nœuds du graphe dynamique et T son horizon de temps.*

Démonstration. Immédiat grâce au lemme 14 puisque l'algorithme PICCNIC a $T+1$ itérations. □

4.4 Étude expérimentale

Nous proposons maintenant d'étudier l'algorithme PICCNIC, décrit précédemment, de façon expérimentale. Il n'est pas question ici de comparer ses performances à un autre algorithme puisque celui que nous proposons est le premier permettant de calculer les composantes connexes non-dominées d'un graphe dynamique. Nous souhaitons observer la connexité de différents types de graphes dans le temps selon leurs composantes connexes persistantes. Nous nous proposons aussi d'observer le comportement de l'algorithme en pratique en le comparant au comportant théorique donné par sa complexité. Nous testons pour cela l'algorithme sur des graphes dynamiques générés de façon aléatoire.

Comme pour l'étude expérimentale de notre algorithme de flot de cout minimum présenté en section 3.3.4 du chapitre 3, nous utilisons la librairie de graphes pour Java GraphStream [1]. Les expériences ont été réalisées sur des machines virtuelles possédant un processeur Intel Core 64 bits avec 8 cœurs, 4Mo de mémoire cache et 96 Go de RAM. L'environnement d'exécution Java que nous avons utilisé est OpenJDK 9.

4.4.1 Générateur de graphes dynamiques

Comme nous l'avons fait pour l'étude expérimentale du chapitre 3, nous générons des graphes dynamiques en commençant par générer le graphe sous-jacent, c'est-à-dire l'ensemble des nœuds du graphe ainsi que l'ensemble des arêtes qui pourront être présentes dans le graphe à un moment de l'intervalle d'étude. Un fois ce graphe sous-jacent créé, nous le rendons dynamique en faisant apparaitre et disparaitre ses arêtes grâce à une chaine de Markov.

1. voir (Dutot et al., 2007) et http ://graphstream-project.org

Notre objectif étant de comparer la manière dont les nœuds du graphes sont connectés au cours du temps, nous allons comparer les résultats de l'algorithme sur différents types de graphes. Nous en testons quatre ayant différentes caractéristiques. Il existe, pour chaque type de graphes, un générateur GraphStream fonctionnel. Nous les décrivons ci-dessous :

1. Les graphes aléatoires, correspondant au modèle d'Erdős-Rényi (Erdős et Rényi, 1960). C'est la méthode la plus classique pour générer des graphes de façon aléatoire. La librairie GraphStream possède un générateur appelé *Random Graph generator* pour cet usage. Ce générateur fonctionne en ajoutant un nœud à la fois puis en le connectant aléatoirement à d'autres nœuds déjà présents dans le graphe. Cette opération est répétée autant de fois que l'on souhaite pour obtenir un graphe de la taille voulue.
2. Les graphes réguliers. Nous considérons ici des tores. Ils sont générés en utilisant le générateur *Grid Generator* de GraphStream. Ce générateur crée un tore régulier ayant le nombre de nœuds souhaité. Il est possible d'en construire aussi avec un degré 8 en connectant les nœuds d'une grille en plus aux 4 nœuds qui sont « en diagonale ».
3. Les graphes invariants d'échelle, qui sont particulièrement adapté quand il s'agit de modéliser des réseaux sociaux ou internet. La librairie GraphStream possède un générateur, *Barabasi-Albert generator*, qui modélise un graphe suivant le modèle décrit par Albert et Barabási (2002). Ce générateur ajoute un nœud à la fois et le connecte à un ou plusieurs nœuds du graphe choisi aléatoirement selon la règle d'attachement préférentiel de Barabasi-Albert. Cette opération est répétée autant de fois que nécessaire jusqu'à obtenir le nombre de nœuds souhaité.
4. Les graphes aléatoires géométriques, qui sont particulièrement adaptés pour toute application où la position spatiale des nœuds est importante, comme les réseaux de communication par exemple ou encore les réseaux logistiques. La librairie de GraphStream possède un générateur, appelé *Random Euclidean generator*, qui permet de créer ce type de graphes. Ce générateur ajoute les nœuds dans un espace $[0;1] \times [0;1]$, et lorsque deux nœuds sont suffisamment proches l'un de l'autre, au sens de la distance euclidienne, au regard d'un seuil donné, alors ils sont connectés par une arête.

Une fois le graphe sous-jacent créé selon l'un des modèles précédemment décrits, le graphe est rendu dynamique grâce à une chaine de Markov. Nous étudions des graphes sans poids sur les arêtes et les nœuds, la dynamique du graphe ne peut donc pas se faire uniquement sur ces éléments comme c'était le cas dans le chapitre 3. On considère ici que les arêtes apparaissent et disparaissent explicitement.

Cette dynamicité est modélisée par une chaine de Markov, à la manière des *edge-Markovian dynamic graphs* décrits par Clementi et al. (2008). La figure 4.9 montre la chaine utilisée ici. L'état 0 correspond à l'absence de l'arête, l'état 1 correspond à sa présence. Lorsqu'une arête est présente à la date θ, alors elle disparaitra à la date $\theta + 1$ avec la probabilité q. Lorsqu'une arête est absente à la date θ, alors elle apparaitra avec un probabilité p à la date $\theta + 1$. Les arêtes pouvant apparaitre pendant l'intervalle d'étude sont celles du graphe sous-jacent précédemment généré.

Nous caractérisons la vie des arêtes par leur présence. La présence représente le pourcentage de présence des arêtes pendant l'intervalle d'étude. Elle est égale à la probabilité stationnaire de présence des arêtes dans la chaine de Markov, soit la probabilité stationnaire de l'état 1. Cette valeur est généralement notée π_1 ($\pi_1 \in [0;1]$). Plusieurs couples de probabilités p et q permettent d'obtenir une valeur π_1 donnée. Nous nous intéressons ici au cas particulier où $p = \pi_1$ et $q = \pi_0$ (avec $\pi_0 = 1 - \pi_1$, la probabilité stationnaire de l'état 0).

4.4.2 Remarques sur l'implémentation

La version de l'algorithme que nous avons implémentée n'est pas exactement identique à celle présenté en section 4.3. En effet, en pratique dans l'implémentation que nous avons faite, nous n'avons

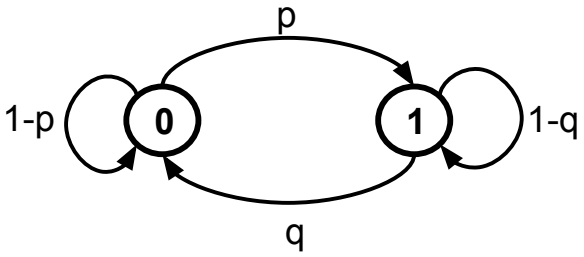


Figure 4.9 : Chaine de Markov représentant les états (présence ou absence) d'une arête et les probabilités d'aller d'un état à un autre. L'état 0 correspond à l'absence de l'arête, l'état 1 à sa présence.

pas utilisé d'ensemble temporaire PCC_t. Dans la première étape de l'algorithme PICCNIC, présentée dans l'algorithme 3, nous ajoutons directement les nouvelles PCC trouvées dans l'ensemble PCC_n sans utiliser la fonction `ajouterPCC`.

En conséquence, il peut y avoir des ensembles doublons dans l'ensemble PCC_n. En d'autres termes, cet ensemble peut contenir plusieurs PCC ayant le même ensemble de nœuds. Cela peut arriver lors des intersections entre une composante connexe statique à une date donnée et une PCC qui était déjà présente à la date précédente. Dans ce cas, ces PCC ont certes le même ensemble de nœuds, mais elles n'ont pas le même âge.

Afin de ne pas conserver d'ensembles doublons, nous ajoutons une instruction entre la fin de la première étape et le début de la seconde. Nous exécutons une fonction qui a pour but de supprimer les doublons de PCC_n. Le principe est simple, lorsque deux PCC ont le même ensemble de nœuds, seule la plus vielle des deux est conservée (on rappelle que cet ensemble ne contient que des PCC vivantes, donc elles ont toutes la même date de fin).

On comprend que cette implémentation n'a théoriquement pas la même complexité que celle de l'algorithme présenté précédemment. En effet, dans le cas présent, l'ensemble PCC_n peut terminer la première étape de l'algorithme PICCNIC (décrit dans l'algorithme 3) avec un nombre de PCC de l'ordre de n^2. Cela implique donc une suppression des doublons en temps $O(n^4)$ au pire.

C'est loin d'être le cas en pratique, d'une part car les doublons sont peu nombreux et d'autre part car nous utilisons pour PCC_n une structure de données très efficace en moyenne. Nous verrons en section 4.4.5 que la complexité observée est bien quadratique en n. Nous avons donc conservé cette implémentation pour les expérimentations.

4.4.3 Paramètres d'expérience

Afin d'évaluer notre algorithme, autant du point de vue des résultats renvoyés par celui-ci que du point de vue de son temps d'exécution, nous avons mené une étude expérimentale où nous avons fait varier certains paramètres des graphes sur lesquels nous étudions l'algorithme, et fixé d'autres paramètres.

Nous réalisons deux expériences, une première dans laquelle nous nous intéressons aux résultats de l'algorithme, et une seconde dans laquelle nous nous intéressons au temps d'exécution de l'algorithme.

Comme nous l'avons expliqué précédemment, nous nous sommes intéressés à quatre types de graphes. Nous testons donc quatre générateurs de graphes nous permettant d'obtenir des graphes dynamiques ayant la structure souhaitée.

L'horizon de temps du graphe doit être suffisamment lointain afin que l'on puisse observer une tendance dans le comportement de l'algorithme, et qu'on puisse de ce fait, interpréter ce que l'on observe. C'est la raison pour laquelle nous fixons l'horizon de temps à 1000 pour chacune des deux expériences.

	Résultats de l'algorithme PICCNIC	Temps d'exécution de l'algorithme PICCNIC
Type de graphes	invariants d'échelle, aléatoires, aléatoires géométriques, réguliers	invariants d'échelle, aléatoires, aléatoires géométriques, réguliers
n	1000	100, 250, 400, 550, 700, 850, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500
T	1000	1000
Degré moyen	4, 8, 12	4
Présence	0,7 ; 0,9	0,9
Nombre d'instances testées	10	10

Table 4.1 : Paramètres utilisés pour les expériences

Dans la première expérience pour laquelle on observe les résultats de l'algorithme, nous avons fixé le nombre de nœuds du graphe à 1000 afin de traiter des graphes de taille importante. Dans la seconde expérience où l'on observe le temps d'exécution de l'algorithme en fonction de la taille du graphe, nous avons fait varier le nombre de nœuds dans l'ensemble suivant : {100 ; 250 ; 400 ; 550 ; 700 ; 850 ; 1000 ; 1500 ; 2000 ; 2500 ; 3000 ; 3500 ; 4000 ; 4500}.

Les graphes que nous générons de façon aléatoire ici ne sont pas définis par leur nombre d'arêtes mais par leur degré moyen. Nous avons choisi d'observer les résultats de l'algorithme pour plusieurs valeurs de degrés moyens : 4, 8 et 12. On notera que les graphes réguliers que l'on génère ici ne sont pas disponibles pour des degrés moyen de 12. Les temps d'exécution de l'algorithme sont observés sur des graphes ayant un degré moyen de 4.

Deux valeurs de présence des arêtes sont testées pour observer les résultats de l'algorithme : $0,7$ et $0,9$ (c'est-à-dire que les arêtes du graphes sont présentes 70% ou 90% du temps). Seule la valeur 0.9 est testée lorsque l'on s'intéresse au temps d'exécution de PICCNIC.

Afin d'obtenir des résultats ayant un sens, statistiquement parlant, nous avons construit 10 instances de graphes pour chaque série de paramètres. De cette manière nous pouvons nous intéresser aux valeurs moyennes ou médianes.

L'ensemble des valeurs que nous avons choisi de tester est synthétisée dans le tableau 4.1.

4.4.4 Étude des résultats de l'algorithme

Nous nous intéressons ici aux résultats de l'algorithme PICCNIC afin d'observer le comportement des composantes connexes des graphes au cours du temps. Les figures 4.10, 4.11 et 4.12 montrent ce qu'on pourrait appeler les « fronts de Pareto moyens ». Ces fronts représentent les composantes connexes persistantes non dominées de chaque type de graphe, pour des graphes de 1000 nœuds et 1000 pas de temps. Chaque point donne la taille moyenne, en nombre de nœuds, de la composante connexe persistante d'une durée, en nombre de pas de temps, donnée pour un type de graphe. Chaque figure montre les résultats pour une valeur du paramètre de présence à $0,7$ sur la gauche de la figure et une valeur de $0,9$ sur la droite de la figure.

La figure 4.10 présente les résultats pour des graphes avec un degré moyen de 4. La figure 4.11 présente les résultats pour des graphes ayant un degré moyen de 8. Et enfin la figure 4.12 présente les résultats pour des graphes ayant un degré moyen de 12. Chaque front correspond à un type de graphe. Pour la figure 4.12, il n'y a pas de front correspondant aux graphes réguliers parce qu'une grille (ici un tore exactement) ne peut pas avoir un degré moyen égal à 12.

La figure 4.13 présente la distribution des degrés des nœuds des graphes pour chaque degré moyen observé et chaque type de graphe sur une échelle logarithmique. Cette figure est coupée car les graphes invariants d'échelle possèdent quelques nœuds avec un très grand degré. Les graphes réguliers étant des tores, tous les nœuds ont exactement le même degré qui est le degré moyen. Les graphes aléatoires et les graphes aléatoires géométriques ont une distribution de degré centrée sur le degré moyen du graphe.

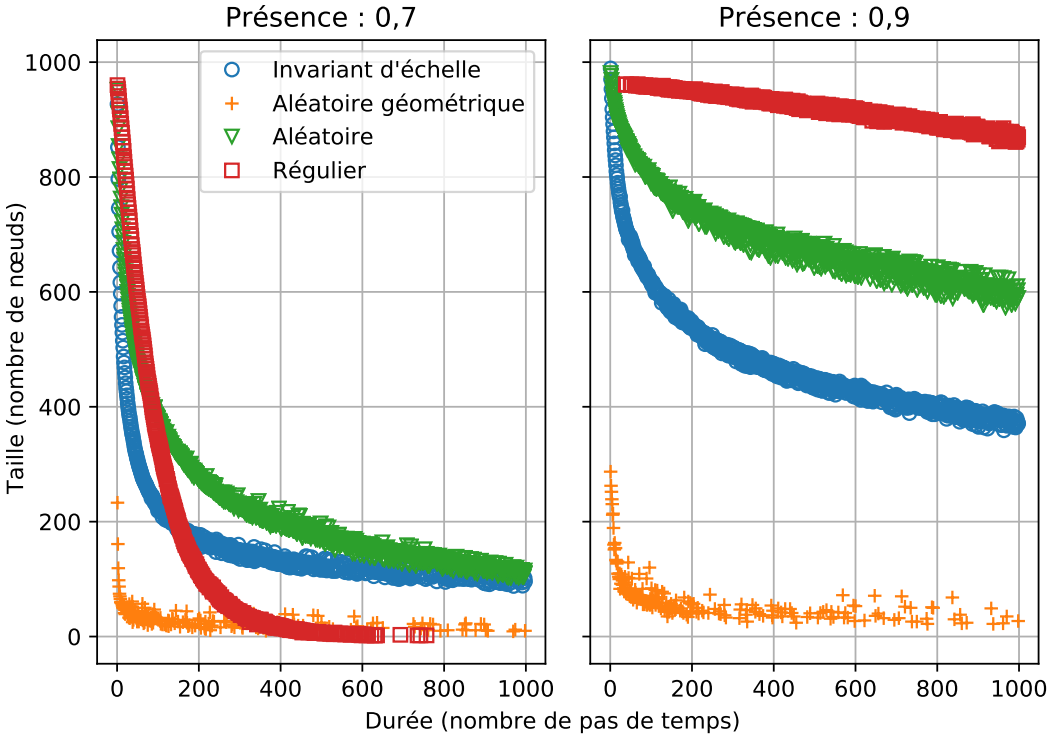


Figure 4.10 : Résultats de l'algorithme PICCNIC sur des graphes à 1000 nœuds, 1000 pas de temps, un degré moyen de 4. Chaque point correspond à la taille moyenne d'une composante connexe persistante non dominée d'une durée donnée. À gauche les résultats pour une valeur de présence de $0,7$, à droite pour une valeur de présence de $0,9$.

Le tableau 4.2 présente le coefficient de clustering moyen de chaque type de graphe pour chaque degré moyen. Pour un graphe donné, le coefficient de clustering moyen se calcule de la façon suivante :

$$\bar{C} = \frac{1}{n} \cdot \sum_{u=1}^{n} C_u \tag{4.8}$$

La valeur C_u correspond au coefficient de clustering local pour chaque nœud u. Il se calcule de la façon suivante :

$$C_u = \frac{\text{Nombre de triangles comprenant } u}{\text{Nombre de paires de voisins de } u} \tag{4.9}$$

La distribution des degrés ainsi que le coefficient de clustering moyen de chaque type de graphe pour chaque degré moyen nous permettent d'expliquer les résultats obtenus avec l'algorithme PICCNIC.

Pour plus de détails sur la notion de coefficient de clustering, on pourra se référer aux travaux de Watts et Strogatz (1998).

Degré	Graphe invariant d'échelle	Graphe aléatoire	Graphe aléatoire géométrique	Graphe régulier
4	0,0221	0,0037	0,5267	0,0000
8	0,0428	0,0084	0,5958	0,2857
12	0,0581	0,0117	0,6040	

Table 4.2 : Coefficient de clustering moyen du graphe sous-jacent pour chaque type de graphe et chaque degré moyen.

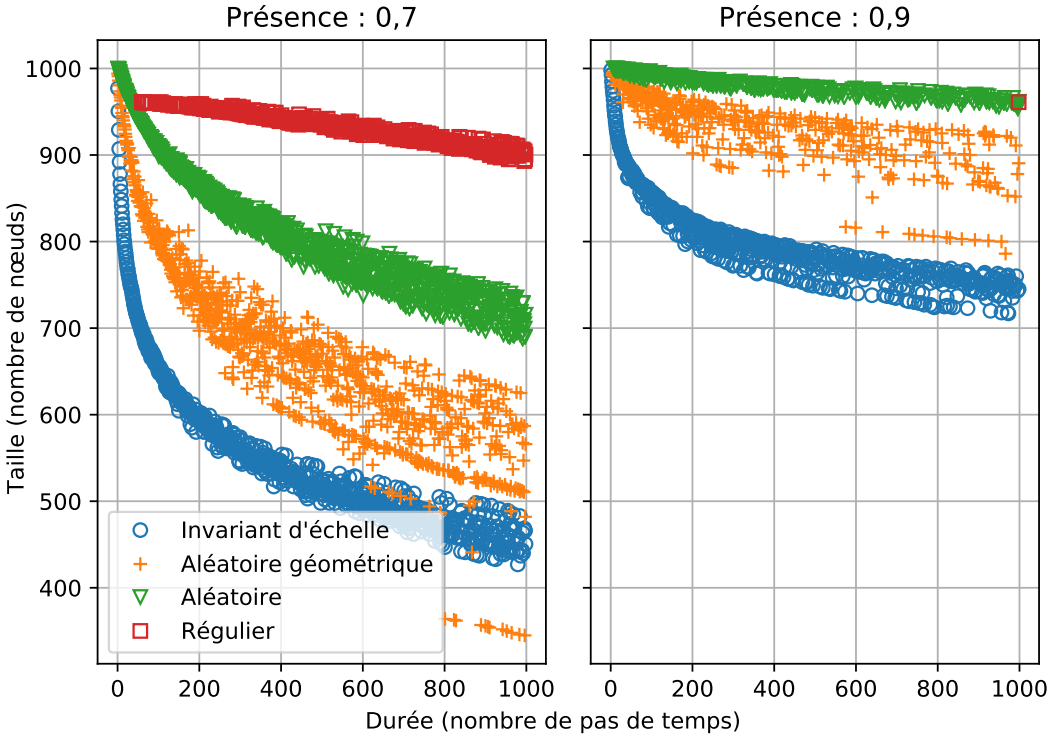


Figure 4.11 : Résultats de l'algorithme PICCNIC sur des graphes à 1000 nœuds, 1000 pas de temps, un degré moyen de 8. Chaque point correspond à la taille moyenne d'une composante connexe persistante non dominée d'une durée donnée. À gauche les résultats pour une valeur de présence de $0,7$, à droite pour une valeur de présence de $0,9$.

Le premier élément à noter concernant ces résultats est le fait que pour une valeur plus élevée de présence, les fronts sont plus élevés pour chaque type de graphes. Cela signifie que plus la valeur de présence des arêtes est élevée, plus les composantes connexes persistantes sont grandes en termes de nombre de nœuds. Cela s'explique logiquement. En effet, une valeur de présence plus élevée signifie que les arêtes vont être présentes plus souvent sur l'intervalle d'étude rendant de ce fait le graphe plus facilement connecté sur la durée, ce qui a pour conséquence de former des PCC de plus grande taille.

On peut observer de la même manière que les fronts sont plus élevés pour des degrés moyens plus élevés. Si les nœuds ont un degré moyen plus élevé, cela signifie qu'ils sont connectés à plus de nœuds dans le graphe, donc cela crée des composantes connexes de plus grande taille. Donc si le graphe a un degré moyen plus élevé alors ses composantes connexes persistantes ont plus de nœuds.

Les graphes aléatoires géométriques ont de petites composantes connexes persistantes qui ne durent pas beaucoup de temps. En effet, les fronts correspondant à ce type de graphes, représentés en orange sur les figures, chutent très rapidement, en particulier avec un degré moyen 4. Cela signifie que ces graphes n'ont pas de composantes « géantes ». Le coefficient de clustering de ce type de graphe (voir tableau 4.2) ainsi que la distribution des degrés des nœuds (voir figure 4.13) montrent que ces graphes ont un nombre important de clusters.

À partir du degré 8, on trouve des composantes de taille significative qui durent pendant un important intervalle de temps de l'intervalle d'étude. Malgré le fait que le graphe présente de nombreux clusters, ceux-ci restent très connectés entre eux, et donc les composantes connexes persistantes survivent plus facilement au passage d'un pas de temps au suivant.

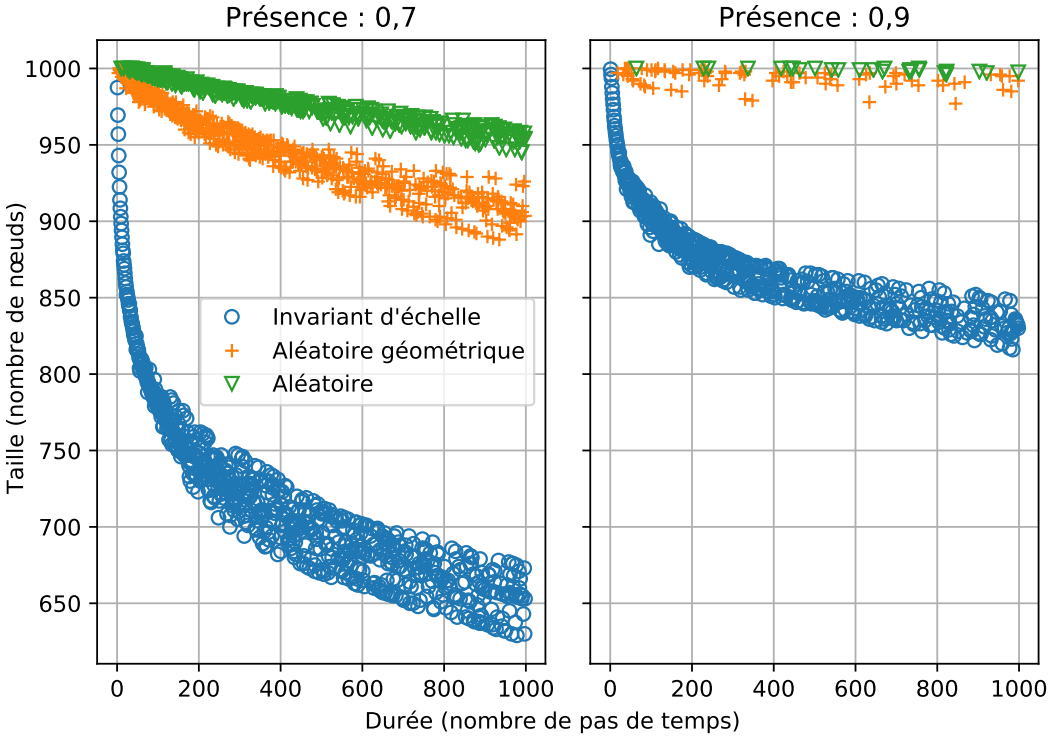


Figure 4.12 : Résultats de l'algorithme PICCNIC sur des graphes à 1000 nœuds, 1000 pas de temps, un degré moyen de 12. Chaque point correspond à la taille moyenne d'une composante connexe persistante non dominée d'une durée donnée. À gauche les résultats pour une valeur de présence de $0,7$, à droite pour une valeur de présence de $0,9$.

Les graphes aléatoires, représentés par les fronts verts sur les figures, ont, comme les graphes aléatoires géométriques à un degré moindre, des composantes connexes persistantes de petite taille et courte durée pour de faibles degrés moyens et faible présence d'arêtes. Le tableau 4.2 indique que ce type de graphes possède peu de clusters. Lorsque le degré moyen est plus grand et que les arêtes sont présentes plus souvent, alors on trouve des composantes de grande taille et longue durée.

Les graphes invariants d'échelle, représentés par les fronts bleus sur les figures, ont des composantes connexes persistantes qui ne durent pas pendant des intervalles importants. En effet, on observe que les fronts chutent assez rapidement. Cela s'explique par la distribution des degrés de ce type de graphes. Comme le montre la figure 4.13, ces graphes ont beaucoup de nœuds de très faible degrés et très peu de nœuds de degré plus élevé. C'est la raison pour laquelle les nœuds se retrouvent facilement déconnectés du reste du graphe. Il est donc peu probable de conserver un grand ensemble de nœuds connectés pendant une longue durée.

Pour chaque degré moyen et chaque valeur de présence des arêtes, le front correspondant aux graphes aléatoires est toujours plus élevé que celui correspondant aux graphes invariants d'échelle. Pour une durée donnée, une composante connexe persistante est plus grande dans des graphes aléatoires que dans des graphes invariants d'échelle. Pour de faibles degrés moyens, les fronts correspondant aux graphes aléatoires géométriques sont plus bas que les fronts de tous les autres types de graphe alors qu'avec un haut degré moyen ils sont plus hauts que les fronts des graphes invariants d'échelle. Les composantes connexes persistantes dans les graphes aléatoires géométriques sont bien plus grandes avec un haut degré. Les graphes invariants d'échelle ont des composantes connexes persistantes plus petites que les graphes aléatoires et aléatoires géométriques parce que, comme expliqué précédemment,

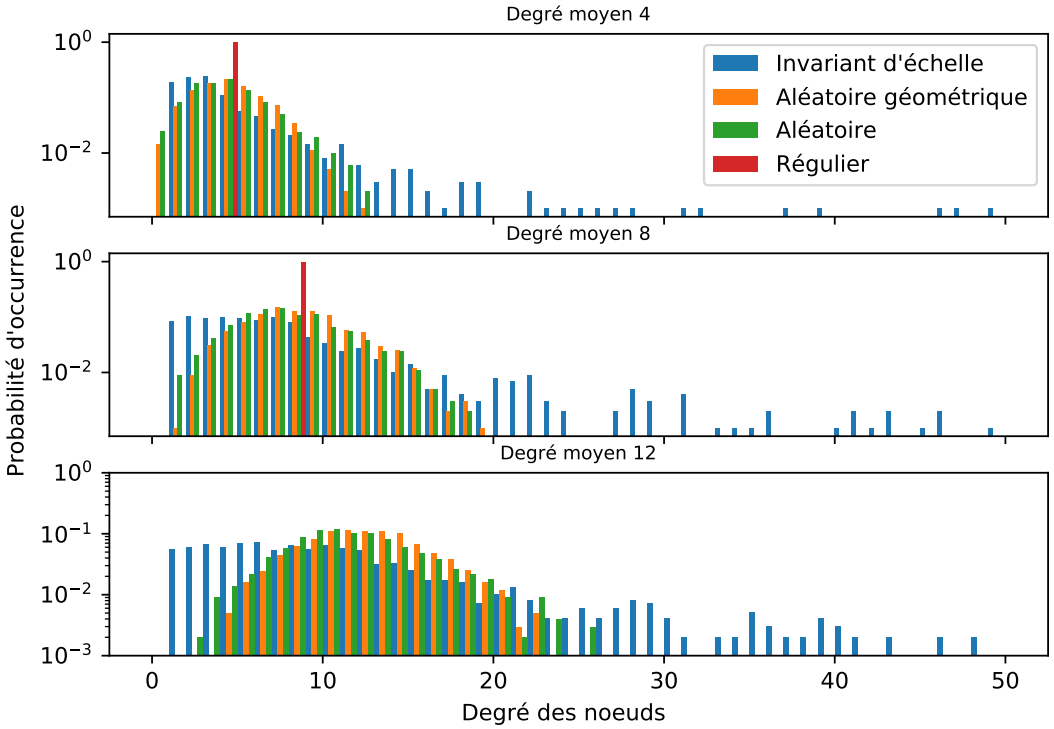


Figure 4.13 : Distribution des degrés des nœuds de chaque type de graphe de chaque degré moyen.

ils ont une majorité de nœuds de faible degré, de ce fait, la probabilité que de tels graphes se scindent en petites composantes reste grande.

Les fronts correspondant aux graphes réguliers sont représentés en rouge sur les figures 4.10 et 4.11. On peut observer sur la figure 4.10 que le front est très élevé pour une valeur de présence à 0,9 alors que pour une présence à 0.7 ce front chute très rapidement. Les tores présentent des composantes « géantes », avec beaucoup de nœuds et qui durent longtemps, lorsque la présence des arcs est suffisant grande. Dans le cas contraire, ces graphes sont facilement déconnectés.

Pour le degré moyen 8 (figure 4.11) et une présence 0,9, il n'y a qu'une seule composante connexes persistante non-dominée dans les graphes réguliers ayant en moyenne plus de 950 nœuds pour une durée de 1000 pas de temps. Cela signifie que 950 des 1000 nœuds que contient le graphe restent connectés pendant tout l'intervalle d'étude.

Avec un degré moyen suffisamment élevé ou une valeur de présence des arêtes suffisamment grande alors les graphes réguliers sont très résistants aux changements dans le graphe. Lorsque ses arêtes apparaissent et disparaissent le graphe reste malgré tout très connecté et de grands groupes de nœuds restent connectés pendant de longues périodes de temps.

4.4.5 Étude du temps d'exécution de l'algorithme

Nous souhaitons étudier le comportement pratique de l'algorithme PICCNIC et le comparer avec son comportement théorique. Pour cela, nous nous intéressons au temps de calcul de l'algorithme. Nous lançons l'algorithme sur chaque type de graphes, pour des graphes de 1000 pas de temps, avec un degré moyen de 4 et une valeur de présence égale à 0,9. Nous faisons varier le nombre de nœuds de 100 à 4500 afin d'observer le temps d'exécution de l'algorithme en fonction du nombre de nœuds du graphe. Les paramètres de cette expérience sont disponibles dans le tableau 4.1.

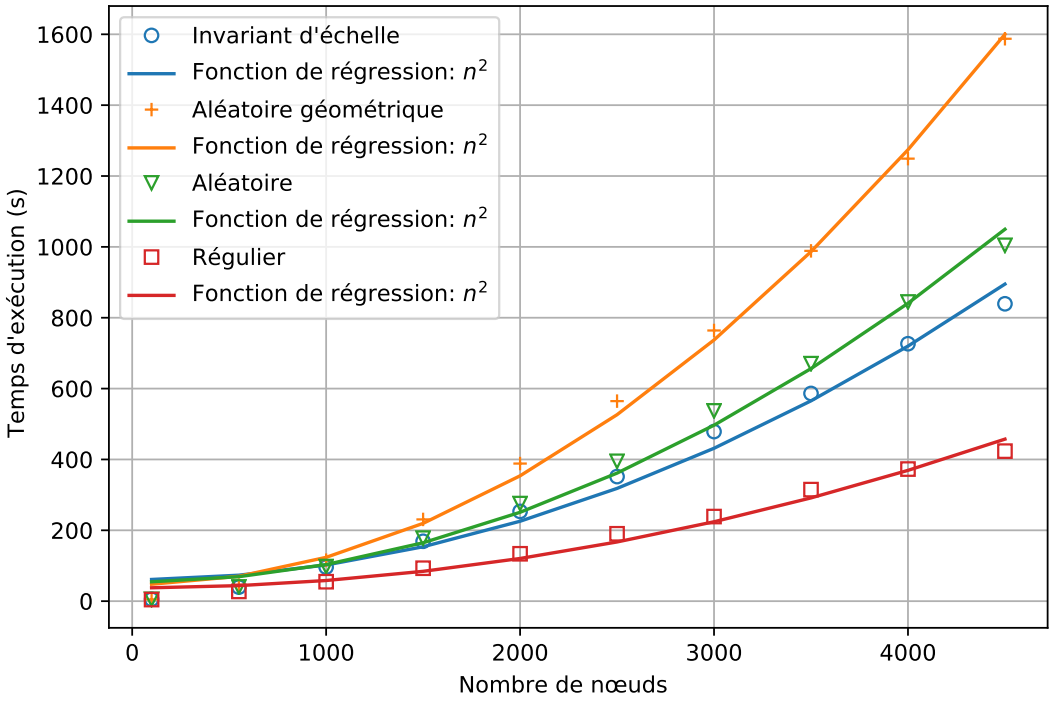


Figure 4.14 : Temps d'exécution médian de l'algorithme PICCNIC en fonction du nombre de nœuds du graphe, pour chaque type de graphes.

La figure 4.14 présente les temps de calcul médians de l'algorihme PICCNIC pour chaque type de graphe et chaque taille de graphes testée. Pour chacun des 4 types de graphes, une fonction de régression, obtenue grâce à la méthode des moindres carrés et calculée sur les temps de calcul médians, est tracée. La complexité de l'algorithme PCCNIC étant $O(n^2 \cdot T)$, lorsque l'on fixe l'horizon de temps T (fixé à 1000 ici), cette complexité devient $O(n^2)$. Les fonctions de régression considérées sont donc de la forme n^2. Afin d'évaluer la qualité de cette régression, on donne pour chacune la valeur R^2. Pour rappel, l'équation (3.21) (page 70) donne le calcul de R^2. Les valeurs R^2 obtenues pour chaque type de graphe sont synthétisées dans le tableau 4.3.

	R^2
Graphe aléatoire	0,992
Graphe aléatoire géométrique	0,997
Graphe régulier	0,979
Graphe invariant d'échelle	0,984

Table 4.3 : Valeur R^2 des courbes de régressions sur les temps de calcul de l'algorithme PICCNIC en fonction de la taille du graphe.

La valeur R^2 de la fonction de régression pour les graphes invariants d'échelle vaut 0,984. Pour les graphes réguliers cette valeur R^2 vaut 0,979. Elle vaut 0,992 pour les graphes aléatoires et 0.997 pour les graphes aléatoires géométriques.

Ces valeurs sont très proches de 1, donc cela montre que la fonction de régression est très adaptée aux valeurs expérimentales obtenues. Cela veut dire que l'algorithme PICCNIC a un comportement

en pratique cohérent à ce que l'on attendait et qui correspond à la complexité théorique au pire de l'algorithme.

Pour des graphes de 4500 nœuds, le temps d'exécution de l'algorithme PICCNIC se situe entre 692 secondes, soit moins de 12 minutes, pour les graphes réguliers et 3045 secondes, soit environ 50 minutes, pour les graphes aléatoires géométriques. Ces temps restent raisonnables et cela montre que cet algorithme peut être utilisé en pratique pour des graphes de taille conséquente.

Dans cette expérience, le test de l'algorithme s'est fait sur 1000 itérations. Pour des graphes de 1000 nœuds par exemple, le temps d'exécution moyen d'une itération est de $0,056$ seconde pour les graphes réguliers, $0,096$ pour les graphes invariants d'échelle, $0,096$ pour les graphes aléatoires et $0,116$ pour les graphes aléatoires géométriques. Pour des graphes de 4500 nœuds, le temps d'exécution moyen d'une itération est de $0,422$ seconde pour les graphes réguliers, $0,853$ seconde pour les graphes invariants d'échelle, $1,054$ seconde pour les graphes aléatoires et $1,577$ seconde pour les graphes aléatoires géométriques. Même pour des graphes de taille importante, une itération se calcule en moyenne en un temps raisonnablement court.

L'algorithme PICCNIC ne nécessite pas de connaitre l'évolution du graphe à l'avance. Comme nous venons de le voir, le temps d'exécution d'une itération est, en moyenne, très court. Il est donc envisageable d'utiliser en pratique l'algorithme PICCNIC de façon incrémentale au fur et à mesure que l'on découvre les changements dans le graphe que l'on traite.

Plus les graphes observés sont de grande taille, plus il est clair que l'algorithme PICCNIC met plus de temps à s'exécuter sur des graphes aléatoires géométriques que sur les autres types de graphes étudiés. De la même manière, on observe aussi que l'algorithme PICCNIC s'exécute beaucoup plus rapidement sur les graphes réguliers que sur les autres types de graphes.

Lorsque l'on compare ces observations sur les temps d'exécution aux composantes connexes persistantes non dominées que montre la figure 4.10, on peut remarquer que l'algorithme s'exécute plus rapidement sur les graphes réguliers qui ont des composantes de grande taille et de longue durée, alors qu'il prend beaucoup plus de temps à s'exécuter sur des graphes aléatoires géométriques qui ont des composantes avec moins de nœuds et qui durent moins longtemps.

On remarque qu'il y a une différence importante entre les temps de calcul des différents types de graphes. Cependant ces temps de calcul se situent toujours dans le même ordre de grandeur. Cela ne pose pas de problème pour l'utilisation, en pratique, de l'algorithme.

4.5 Extensions des composantes connexes persistantes

Cette section est consacrée aux extensions possibles de notre définition de composantes connexes persistantes. Nous discutons de trois variantes.

4.5.1 Composantes fortement connexes persistantes

Une extension qui est évidente concerne les graphes orientés. Nous avons jusqu'ici étudié des graphes non-orientés. Il semble cependant relativement simple d'étendre notre définition de composantes connexes persistantes pour prendre en compte un graphe orienté.

De façon analogue aux composantes fortement connexes d'un graphe orienté statique, on peut définir des composantes fortement connexes persistantes.

Définition 14 (Composante fortement connexe persistante)**.** *Une composante fortement connexe persistante (SPCC pour strongly persistant connected component en anglais) p dans un graphe dynamique*

orienté G est un ensemble de k sommets dans V qui restent fortement connectés dans le graphe pendant l pas de temps consécutifs.
Les nœuds $u_1 \dots u_k$ forment une composante fortement connexe persistante de taille k et de durée l si et seulement si il existe $G_i \dots G_{i+l-1}$ tels que $u_1 \dots u_k$ sont dans la même composante fortement connexe au sens statique dans chaque G_j ($i \leqslant j < i + l$).

La définition est très similaire à la définition des composantes connexes persistantes (définition 10). En effet, celle-ci se basait sur la définition d'une composante connexe au sens statique, et la définition 14 se base sur la définition d'une composante fortement connexe au sens statique.

Le critère de dominance défini pour les composantes connexes persistantes (définition 13) est valable dans le cas d'un graphe dynamique orienté. Il peut donc être utilisé directement sans adaptation.

D'un point de vue algorithmique, l'algorithme PICCNIC est utilisable quasiment directement. En effet, l'unique différence entre les PCC et les SPCC se trouve sur la manière dont les nœuds sont connectés sur les t-graphes. Il suffit donc, dans chaque itération de l'algorithme de comparer les composantes fortement connexes de la date courante avec SPCC existantes à la date précédente.

Il n'y a pour cela qu'une seule ligne à modifier dans l'algorithme. À la ligne 3 de l'algorithme 2, l'ensemble *CC* doit recevoir l'ensemble des composantes fortement connexes de taille supérieure à 2 du graphe G_θ. Le reste de l'algorithme est inchangé.

On peut noter que les preuves de correction et complexité restent valables puisque la construction des composantes fortement connexes dans un graphe orienté statique se fait également en temps $O(m + n)$.

4.5.2 Composantes connexes éternelles

Les composantes connexes persistantes sont présentes dans le graphe dynamique pendant un intervalle de temps compris dans l'intervalle d'étude du graphe. Intéressons-nous à présent aux composantes connexes persistantes particulières qui sont présentes durant tout l'intervalle d'étude. Nous les nommons composantes connexes éternelles.

Définition 15 (Composante connexe éternelle)**.** *Une composante connexe éternelle p dans un graphe dynamique G est un ensemble de k nœuds dans V qui restent connectés dans le graphe durant tout l'intervalle d'étude.*
Les nœuds $u_1 \dots u_k$ forment une composante connexe éternelle de taille k si et seulement si $u_1 \dots u_k$ sont dans la même composante connexe au sens statique dans chaque G_θ ($1 \leqslant \theta \leqslant T$).

La particularité des composantes éternelles est qu'elles forment une partition de l'ensemble des nœuds du graphe. À l'inverse, les PCC (définition 10) ne forment pas une partition de l'ensemble V. En effet, nous avons vu que des nœuds pouvaient appartenir à plusieurs composantes connexes persistantes différentes à la même date.

Théorème 16 (À propos des composantes connexes éternelles)**.** *Les composantes connexes éternelles forment une partition des nœuds du graphe.*

Démonstration. Supposons qu'un nœud α fasse partie de deux composantes connexes éternelles différentes. On a $\{\alpha\ ;\ a\}$ et $\{\alpha\ ;\ b\}$ deux composantes connexes éternelles. Puisque les nœuds a et α sont dans la même composante connexe éternelle, cela signifie que a et α sont dans la même composante connexe statique dans chaque G_θ, $\theta \in \mathcal{T}$. De la même manière, si b et α sont dans la même composante connexe éternelle cela signifie qu'ils se trouvent dans la même composante connexe statique dans chaque G_θ, $\theta \in \mathcal{T}$.

Donc dans chaque G_θ, $\theta \in \mathcal{T}$, α se trouve dans la même composante connexe statique que a et b. Donc a et b se trouvent dans la même composante connexe statique dans chaque G_θ, $\theta \in \mathcal{T}$. Et donc a

et b se trouvent dans la même composante connexe éternelle. Donc les composantes éternelles $\{\alpha\ ;\ a\}$ et $\{\alpha\ ;\ b\}$ ne forment qu'une seule et même composante éternelle $\{\alpha\ ;\ a\ ;\ b\}$.

Donc un nœud ne peut pas se trouver dans plusieurs composantes connexes éternelles différentes. □

Pour identifier ces composantes dans un graphe dynamique, il est bien entendu possible d'utiliser l'algorithme PICCNIC que l'on aurait adapté pour l'occasion. Il suffirait pour cela de ne pas conserver les composantes qui se seraient terminées avant d'avoir atteint l'horizon de temps. Le désavantage de cette méthode est son cout. En effet, la complexité de PICCNIC est $O(n^2 \cdot T)$ mais pour ces composantes particulières, on peut atteindre une meilleure complexité.

En réalisant un parcours du graphe à chaque pas de temps, comme pour la recherche de composantes connexes ordinaires dans un graphe statique, on peut identifier les composantes connexes éternelles. La différence avec un parcours permettant de chercher des composantes connexes classiques se trouve dans la manière d'attribuer un numéro de composante aux nœuds.

L'algorithme 7 décrit l'algorithme permettant d'identifier dans un graphe dynamique toutes les composantes connexes éternelles. On considère que chaque nœud porte initialement un numéro fictif de composante 0. La numérotation se fait grâce à un numéro sur chaque nœud u, auquel on accède grâce à `numéro(u)`, et à une structure de données, nommée `matchIndex` dans l'algorithme, qui permet d'attribuer un numéro de composante donné à un nœud en fonction du numéro de la composante dans laquelle il se trouvait au pas de temps précédent. La structure `matchIndex` peut être un tableau avec, dans la case i, où i est un numéro de composante au pas de temps précédent, une nouvelle valeur correspondant au nouveau numéro de composante.

Algorithme 7 : Identification des composantes connexes éternelles

```
Input : Graphe dynamique G, intervalle d'étude 𝒯 = {1,...,T}, numéro(u) = 0 ∀u ∈ V
1  nbC ← 0 ;                 /* Compteur pour le nombre de composantes connexes */
2  pour θ ∈ {1,...,T} et nbC < n faire
3      nbC ← 0
4      matchIndex ←NULL ;  /* Ensemble associant l'ancien numéro de composante
        au nouveau numéro de composante */
5      u ← un nœud non visité de G_θ
6      tant que u ≠ NULL faire
7          numéroter(u,nbC,matchIndex)
8          visité(u)
9          u ← suivant(u) ;                  /* prochain nœud dans le parcours */
10         si u = NULL alors
               /* une composante connexe de G_θ a été parcourue intégralement
                  */
11             u ← un nœud non visité de G_θ ou NULL si tous les nœuds sont visités
12             matchIndex ←NULL ;        /* Visite d'une nouvelle composante donc
                réinitialisation de l'index de numérotation */
13         fin
14     fin
15 fin
```

Lemme 17 (Correction d'une itération de l'algorithme d'identification des composantes connexes éternelles)**.** *À la fin de chaque itération θ, les nœuds du graphe portent le numéro de la composante connexe persistante dans laquelle ils sont depuis le premier pas de temps et jusqu'à la date θ.*

Algorithme 8 : Fonction `numéroter`

```
Input : nœud u, nombre de composantes différentes rencontrées nbC, tableau matchIndex
1 si matchIndex(numéro(u)) = NULL alors
2 |   matchIndex(numéro(u)) ← nbC + 1
3 |   nbC ← nbC + 1
4 fin
5 numéro(u) ← matchIndex(numéro(u))
```

Démonstration. À chaque itération θ de l'algorithme 7, correspondant au pas de temps θ de l'intervalle d'étude du graphe, le graphe statique G_θ est parcouru composante connexe par composante connexe.

Grâce à `matchIndex`, les nœuds qui se trouvaient dans la même composante avant et sont dans la même composante à la date θ portent encore le même numéro. La structure de données `matchIndex` est réinitialisée juste avant de visiter une autre composante connexe de G_θ alors que NbC est conservé, donc on ne réutilise pas les mêmes numéros de composante éternelle.

Si, à la fin de l'itération θ, deux nœuds ont le même numéro de composante éternelle, alors ils étaient dans la même composante connexe persistante jusqu'à $\theta - 1$ et ont été visité dans G_θ sans réinitialisation de `matchIndex`, donc ils se trouvent dans la même composante connexe de G_θ, donc ils se trouvent bien dans la même composante connexe persistante jusqu'à la date θ.

Au contraire si, à la fin de l'itération θ, deux nœuds n'ont pas le même numéro de composante connexe éternelle, soit ils n'avaient pas le même numéro à la fin de l'itération précédente et donc ne se trouvaient pas dans la même composante connexe persistante jusqu'à la date $\theta - 1$, soit la structure de donnée `matchIndex` a été réinitialisée entre la visite de ces deux nœuds, et donc cela signifie qu'ils ne se trouvent pas dans la même composante connexe statique de G_θ. Dans un cas comme dans l'autre, ces deux nœuds ne se trouvent pas dans la même composante connexe persistante jusqu'à la date θ. □

Théorème 18 (Correction de l'algorithme d'identification des composantes connexes éternelles)**.** *Les nœuds du graphe dynamique portent le numéro de la composante éternelle dans laquelle ils se trouvent à la fin de l'algorithme.*

Démonstration. Grâce au lemme 17, on sait qu'à la fin de l'algorithme, les nœuds porteront le numéro de la composante connexe persistante dans laquelle ils se trouvent depuis le début de l'intervalle d'étude et jusqu'à l'horizon de temps. Il s'agit précisément de la définition d'une composante connexe éternelle.

La structure `matchIndex` est vidée au début de chaque itération s'assurant ainsi de toujours numéroter les composantes de 1 à NbC.

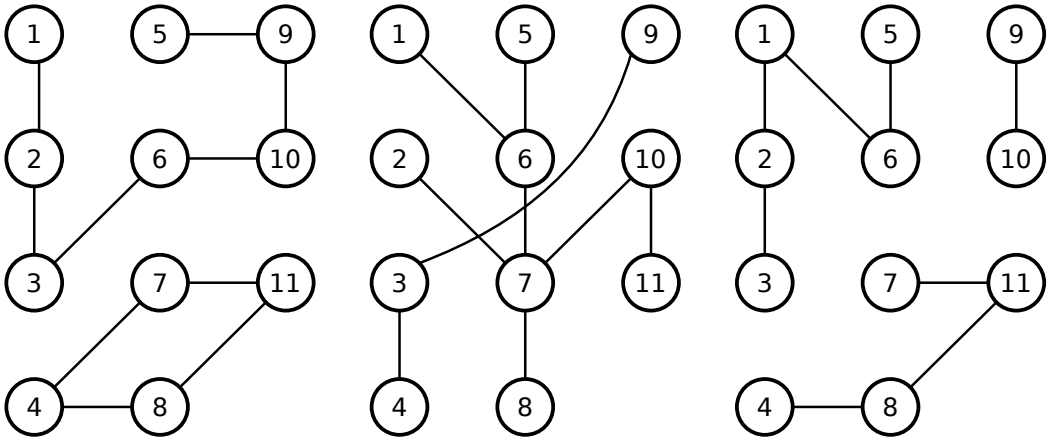
L'algorithme s'arrête lorsque l'on a réalisé une itération sur chaque t-graphe ou lorsque $NbC = n$. En effet, dans ce dernier cas, cela signifie que toutes les composantes éternelles sont des singletons, c'est donc inutile de réaliser d'autres itérations sur les t-graphes suivants puisqu'il en restera ainsi. □

Théorème 19 (Complexité de l'algorithme d'identification des composantes connexes éternelles)**.** *La complexité de l'algorithme 7 est* $O(m \cdot T)$.

Démonstration. L'algorithme comporte T itérations, une pour chaque pas de temps du graphe dynamique. Dans chaque itération, un parcours du graphe est effectué en $O(m)$. L'opération de numérotation est faite en temps constant puisque la structure de données `matchIndex` est un tableau.

Au total, l'algorithme 7 a une complexité $O(m \cdot T)$. □

Afin de clarifier l'algorithme 7, observons son exécution sur le graphe de la figure 4.15. Au début, chaque nœud porte le numéro de composante 0, l'ensemble `matchIndex` est vide, et $NbC = 0$.



(a) G_1 (b) G_2 (c) G_3

Figure 4.15 : Graphe dynamique de 11 nœuds sur 3 pas de temps.

Dans la première itération, on parcourt le t-graphe G_1. On parcourt, dans l'ordre, les nœuds 1, 2, 3, 6, 10, 9, 5. Puisqu'ils sont tous dans la même composante connexe, ces nœuds portent le numéro de composante 1. Puis il n'y a plus de nœud `suivant`, donc on vide `matchIndex` et on poursuit le parcours des nœuds du graphe à partir du nœud 4. Il porte initialement le numéro 0 on lui attribue le numéro 2 puisqu'on avait déjà une composante éternelle. L'ensemble `matchIndex` permet de se souvenir que maintenant les nœuds qui portaient le numéro de composante 0 doivent porter le numéro de composante 2. On parcourt ensuite les nœuds 7, 11, 8 en leur attribuant le numéro de composante 2.

Au début de la deuxième itération, le nombre de composantes éternelles rencontrées est remis à 0, et l'ensemble `matchIndex` est vidé. Le t-graphe G_2 a deux composantes connexes. On commence le parcours sur le nœud 1, on lui attribue le nouveau numéro de composante 1. On poursuit sur les nœuds 6 et 5 qui portaient le même numéro donc on leur attribue aussi le nouveau numéro 1. Le parcours arrive sur le nœud 7 qui portait le numéro 2, donc on lui attribue le numéro 2. On parcourt ensuite les nœuds 2, 8, 10 et 11 et ceux qui portaient le numéro 1 se voient attribuer le nouveau numéro 1 et ceux qui portaient le numéro 2 se voient attribuer le nouveau numéro 2. Il n'y a pas de nœud `suivant`, donc on reprend le parcours à partir d'un nœud du graphe non visité, soit le nœud 3. L'ensemble `matchIndex` est vidé et NbC vaut 2 ici. Le nœud 3 portait le numéro 1, on lui attribue le numéro 3, et le nœud 4 portait le numéro 2, on lui attribue le numéro 4.

Au début de la troisième itération, le nombre de composantes éternelles rencontrées est remis à 0, l'ensemble `matchIndex` est vidé. La t-graphe G_3 a trois composantes connexes. On parcourt le graphe à partir du nœud 1 qui portait le numéro de composante 1 et à qui on attribue le nouveau numéro de composante 1. Le nœud 2 portait le même numéro que le nœud 1, donc il porte aussi le nouveau numéro de composante 1. Le nœud 3 portait le numéro 3 et on lui attribue le nouveau numéro de composante 2. Le nœud 6 portait le numéro 1 donc on lui attribue le nouveau numéro 1, tout comme le nœud 5. Il n'y a pas de nœud `suivant` donc on reprend le parcours à partir du nœud 4 en vidant d'abord l'ensemble `matchIndex`. On a rencontré, à ce stade, 2 composantes éternelles, donc $NbC = 2$. Le nœud 4 portait le numéro 4 et se voit attribuer le nouveau numéro 3. Le nœud 8 portait le numéro 2 donc on lui attribue le numéro 4 car il s'agit de la quatrième composante éternelle que l'on rencontre. Les nœuds 11 et 7 portaient le numéro 2, donc on leur attribue aussi le numéro 4. Il n'y pas de nœud *suivant*, donc on reprend le parcours à partir du nœud 9 et on vide l'ensemble `matchIndex`. On a pour l'instant rencontré 4 composantes éternelles donc $NbC = 4$. Le nœud 9 portait le numéro 3 donc on lui attribue le numéro 5. Enfin le nœud 10 portait le numéro 1 donc on lui attribue le numéro 6.

À l'issue de l'algorithme, on a exactement 6 composantes connexes éternelles : une première com-

posée des nœuds {1 ; 2 ; 5 ; 6}, une deuxième composée des nœuds {7 ; 8 ; 11} et quatre singletons.

Les composantes connexes éternelles sont plus simples à identifier que les composantes connexes persistantes. En effet, l'algorithme 7 permet de toutes les identifier en $O(m \cdot T)$. On ne peut cependant pas se contenter de réaliser une intersection des t-graphes car les nœuds peuvent être connectés par des arêtes différentes, comme on a pu le voir dans l'exemple de la figure 4.7.

On note qu'il est aussi possible de définir les composantes fortement connexes éternelles dans des graphes dynamique orientés, qui s'appuient sur la notion de forte connexité dans les graphes statique orientés au lieu de s'appuyer sur la notion de connexité simple dans les graphes statiques non orientés.

Puisque les composantes fortement connexes d'un graphe statique se trouvent aussi grâce à un parcours de graphe, alors les composantes fortement connexes éternelles peuvent aussi se trouver de cette manière, en adaptant la manière de numéroter les composantes de chaque nœud. On obtient un algorithme de même complexité que l'algorithme 7.

4.5.3 Composantes connexes persistantes discontinues

Avec la définition des composantes connexes persistantes, nous nous sommes intéressés à un ensemble de nœuds qui restent connectés pendant un certain intervalle de temps. Mais nous pouvons nous interroger sur les ensembles de nœuds qui seraient connectés à plusieurs pas de temps de l'intervalle d'étude sans que ceux-ci soient consécutifs. C'est ce que nous appelons ici les composantes connexes persistantes discontinues.

Définition 16 (Composante connexe persistante discontinues)**.** *Une composante connexe persistante discontinue p dans un graphe dynamique G est un ensemble de k sommets dans V qui sont connectés dans le graphe à l pas de temps différents.*
Les nœuds $u_1 \dots u_k$ forment une composante connexe persistante discontinue de taille k et de durée l si et seulement si il existe $G_{i_1} \dots G_{i_l}$ tels que $u_1 \dots u_k$ sont dans la même composante connexe au sens statique dans chaque G_j, $j \in \{i_1 \dots i_l\}$.

Cette définition met en évidence les ensembles de nœuds qui sont connectés dans le graphe sans avoir de contraintes sur les dates où ils sont connectés. Cela permet d'identifier les ensembles de nœuds qui sont « souvent » connectés dans le graphe, ce qui n'est pas possible avec la définition des composantes connexes persistantes. Ce type d'ensembles peut représenter un intérêt applicatif (des réseaux de capteurs connectés périodiquement par exemple).

Prenons par exemple un ensemble K de k nœuds connectés un pas de temps sur deux. D'après la définition des composantes connexes persistantes, ces nœuds forment une PCC de taille k et de durée 1. Si maintenant on considère que les composantes peuvent être interrompues dans le temps, comme la définition 16 l'indique, alors l'ensemble K forme une composante connexe persistante discontinue de taille k et de durée $l = \frac{T}{2}$.

Malheureusement, malgré l'intérêt applicatif que présentent les composantes connexes persistantes discontinues, le nombre de telles composantes dans un graphe dynamique est potentiellement très important.

Théorème 20 (À propos des composantes connexes persistantes discontinues)**.** *Le nombre de composantes connexes persistantes discontinues dans un graphe dynamique peut être exponentiel en n, même pour un nombre de pas de temps dans l'intervalle d'étude linéaire en n.*

Démonstration. On construit un graphe dynamique G ayant un ensemble de nœuds V de taille n et où l'horizon de temps $T = n$. Chaque t-graphe G_α est composé d'exactement deux composantes connexes statiques : un singleton composé du nœud $\{\alpha\}$ et une composante ayant $n - 1$ nœuds composée des

nœuds $V \setminus \{\alpha\}$. L'ensemble des arêtes qui connectent les différentes composantes connexes à chaque pas de temps peut être quelconque.

Une représentation schématique du graphe que l'on a construit est proposée dans la figure 4.16.

Comptons le nombre de composantes connexes persistantes discontinues que l'on trouve dans le graphe G. Pour cela, on regarde dans quels t-graphes se trouvent chacune des composantes connexes persistantes possibles.

Commençons par les composantes de taille $n-1$. La composante $\{1;2;\ldots;n-1\}$ se trouve dans le t-graphe G_n, la composante $\{1;2;\ldots;n-2;n\}$ se trouve dans le t-graphe G_{n-1}, etc. Chacune des composantes possibles de taille $n-1$ apparait dans un unique t-graphe. Le graphe dynamique G possède donc n composantes connexes persistantes discontinues de taille $n-1$ et de durée 1.

Cherchons les composantes de taille $n-2$. La composante connexe persistante discontinue $\{1;2;\ldots;n-2\}$ est présente dans les t-graphes G_{n-1} et G_n. La composante $\{1;2;\ldots;n-3;n\}$ est présente dans les t-graphes G_{n-2} et G_{n-1}. On peut généraliser en disant que la composante de taille $n-2$ qui contient tous les nœuds de V sauf les nœuds i et j est présente dans les t-graphes G_i et G_j. Il y a $\binom{n}{2}$ ensembles de nœuds de taille 2.

De façon générale, la composante connexe persistante de taille $n-\alpha$ composée des nœuds de l'ensemble $V \setminus \{i_1;\ldots;i_\alpha\}$ est présente dans les t-graphes $G_{i_1},\ldots,G_{i_\alpha}$. Ou présenté autrement, en prenant α t-graphes au hasard, on obtient un ensemble de $n-\alpha$ nœuds qui sont connectés directement ou indirectement dans chacun des α t-graphes. On obtient donc $\binom{n}{\alpha}$ ensembles de nœuds de taille $n-\alpha$ dans V.

En résumé, le graphe dynamique G possède $\binom{n}{\alpha}$ composantes connexes persistantes discontinues de taille $n-\alpha$ et de durée α pour $1 \leqslant \alpha \leqslant n-2$. On ignore les composantes de taille 1, et la composante de taille n n'est pas présente dans ce graphe car il n'est jamais connexe.

Au total, le nombre de composantes connexes persistantes discontinues du graphe G est $\sum_{\alpha=1}^{n-2}\binom{n}{\alpha}$, soit $2^n-\binom{n}{0}-\binom{n}{n-1}-\binom{n}{n}=2^n-(n+2)$.

Le graphe dynamique G de taille n et durée n possède de l'ordre de 2^n composantes connexes persistantes discontinues. □

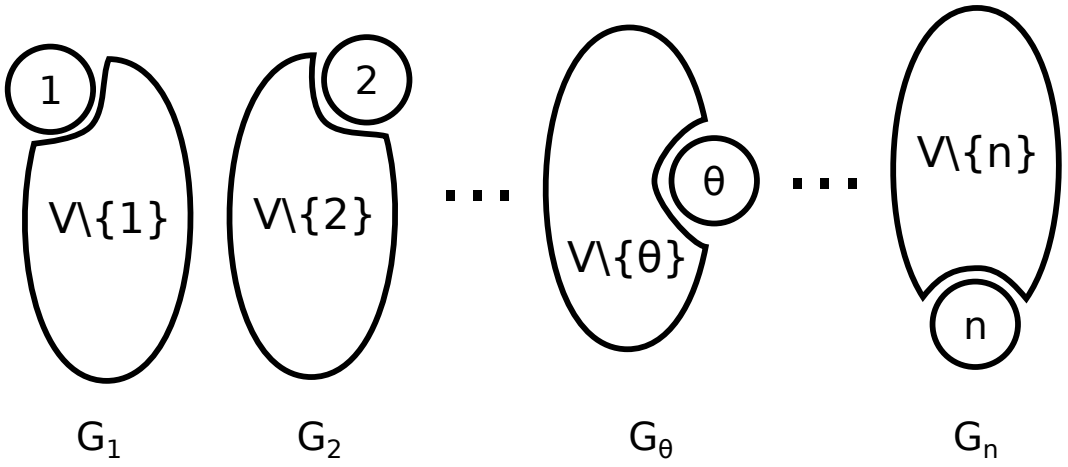


Figure 4.16 : Graphe dynamique sur n pas de temps possédant de l'ordre de 2^n composantes connexes persistantes discontinues.

Le cas du graphe dynamique décrit dans la figure 4.16 est extrême, mais de nombreux graphes peuvent appartenir à une famille de graphes semblables.

On peut noter que même en ne s'intéressant qu'aux composantes d'une taille supérieure à un seuil donné, le nombre de composantes connexes persistantes discontinues reste exponentiel. Par exemple, dans le graphe de la figure 4.16, si l'on cherche uniquement les composantes d'une taille supérieures à $\frac{n}{2}$, alors le nombre de telles composantes est $\sum_{\alpha=1}^{\frac{n}{2}} \binom{n}{\alpha} = 2^{n-1} - 1$.

De la même manière, si l'on cherche les composantes qui durent plus longtemps qu'un certain seuil, leur nombre est aussi exponentiel. Toujours dans le graphe de la figure 4.16, le nombre de composantes qui durent plus de la moitié de l'intervalle d'études est $\sum_{\alpha=\frac{n}{2}}^{n-2} \binom{n}{\alpha} = 2^{n-1} - n - 1$.

Le nombre de composantes connexes persistantes discontinues dans un graphe dynamique est beaucoup trop important pour que l'on puisse envisager de toutes les calculer.

De plus, comme chaque composante rencontrée peut réapparaitre plus tard, il est impossible de déterminer quelles composantes sont dominantes et lesquelles sont dominées avant d'avoir parcouru tout le graphe pour savoir combien de temps elles durent. Pour obtenir cette information, il faudrait donc au préalable trouver toutes les composantes connexes persistantes discontinues que l'on rencontre dans le graphe, mais comme énoncé par le théorème 20, il peut y en avoir un nombre exponentiel.

En bref, même s'il pourrait être intéressant de connaitre les composantes connexes persistantes discontinues d'un graphe dynamique donné, en pratique, c'est impossible à traiter, car ce serait beaucoup trop couteux.

4.6 Ensemble de Steiner

Le problème de Steiner, que nous avons rappelé en section 4.1, repose sur la notion de connexité. Nous nous interrogeons ici sur la manière d'étendre ce problème aux graphes dynamiques, en particulier la manière dont il peut être défini.

Nous nous intéressons ici à un modèle de graphes dynamiques évoluant sur un intervalle d'étude discret et fini dans lequel les nœuds restent présents pendant tout l'intervalle d'étude, les arêtes peuvent aller et venir et ont des poids positifs variant avec le temps. Selon le modèle de la figure 3.1 (page 27), la figure 4.17 schématise le modèle de graphe étudié ici.

À partir d'un graphe dynamique $G = (G_\theta)_{\theta \in \mathcal{T}}$ où $G_\theta = (V, E_\theta)$ et $\cup_{\theta \in \mathcal{T}} E_\theta = E$, d'un poids positif w_{e_θ}, $e_\theta \in E_\theta$, sur les arêtes et d'un ensemble de nœuds terminaux $S \subset V$, comment peut-on connecter ces nœuds terminaux de façon à minimiser le cout total ? Quels autres nœuds de graphe G choisir pour que les terminaux soient connectés sur tout $\mathcal{T}$? Nous proposons plusieurs pistes de réflexion sur de possibles définitions de ce problème dans un contexte dynamique. Nous nous concentrons ensuite sur un cas particulier du problème dont nous sommes en mesure de prouver la complexité.

Pour chaque proposition, nous souhaitons pouvoir connecter les terminaux à chaque pas de temps, cela signifie donc que les nœuds terminaux sont inclus dans une composante connexe éternelle (voir définition 15 page 117) du graphe dynamique G que l'on étudie.

Nous avons une approche différente de celle trouvée dans les rares travaux de la littérature abordant le problème de Steiner dans un contexte dynamique. En effet, comme rappelé en section 4.1.2, Imase et Waxman (1991) cherchent à conserver un arbre au cours du temps, Khodaverdian et al. (2016) cherchent à connecter des couples de nœuds à des pas de temps précis et Rozenshtein et al. (2016) souhaitent connecter des nœuds par des ensembles de trajets quand nous voulons conserver la connexité d'un ensemble de nœuds au cours du temps.

4.6.1 Définitions

Nous considérons un ensemble fixe de nœuds terminaux, l'idée étant de trouver la manière de les garder connectés de façon optimale au cours du temps. On cherche donc à définir ce que signifie exactement « connecter les terminaux de manière optimale ».

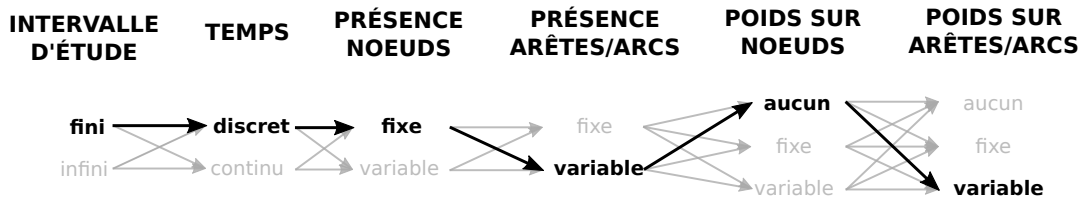


Figure 4.17 : Illustration du modèle de graphe dynamique étudié pour le problème de Steiner

En particulier, nous proposons de trouver un ensemble de nœuds du graphe qui permette de connecter les terminaux. Dans la suite, nous parlons donc d'ensembles de Steiner. Ces ensembles contiennent les nœuds terminaux. Nous proposons quatre visions possibles des ensembles de Steiner et discutons ces différentes définitions.

4.6.1.1 Arbre de Steiner constant

Une possibilité qui peut sembler naturelle serait de chercher l'ensemble de nœuds et d'arêtes qui permettent de connecter les nœuds terminaux de manière à ce que le poids total soit minimum sur tout l'intervalle d'étude. C'est ce que l'on définit formellement de la façon suivante :

Définition 17 (Arbre de Steiner constant dans un graphe dynamique)**.** *Soit* $G = (G_\theta)_{1 \leqslant \theta \leqslant T}$ *un graphe dynamique tel que :*

- $G_\theta = (V, E_\theta),\ 1 \leqslant \theta \leqslant T$
- $\cup_{1 \leqslant \theta \leqslant T} E_\theta = E$
- *Les arêtes sont pondérées :* $w_{e_\theta} \geqslant 0\ \forall e_\theta \in E_\theta$

Pour un ensemble donné de nœuds terminaux $S \subset V$*, on cherche un ensemble fixe de nœuds* $V' \subset V$*, avec* $S \subset V'$*, et un ensemble fixe d'arêtes* $E' \subset E_\theta \forall \theta \in \mathcal{T}$ *tels que :*

- $G'_\theta = (V', E')$ *est un graphe connexe* $\forall \theta \in \mathcal{T}$
- $\sum_{\theta \in \mathcal{T}} \sum_{e'_\theta \in E'} w_{e'_\theta}$ *est minimum*

Cette définition impose que la connexion entre les nœuds terminaux se fasse sur les mêmes liens au cours du temps, ce qui est très restrictif. De ce fait, il est très simple de ramener ce problème au problème de Steiner dans un graphe statique.

En effet, on peut s'intéresser au graphe statique formé par l'intersection de tous les t-graphes. Ce graphe possède les mêmes nœuds que le graphe dynamique G, et possède toutes les arêtes de G présentes à chaque pas de temps. Le poids de chaque arête sera alors la somme des poids de l'arête dans G à chaque pas de temps. Il suffit alors de chercher l'arbre de Steiner dans ce graphe statique.

Le problème n'est alors certes pas simple à résoudre, puisque le problème de Steiner est NP-complet. Mais le cas dynamique n'apporte rien de significatif puisqu'il est possible d'utiliser tous les algorithmes connus directement sur le graphe intersection. C'est la raison pour laquelle il est inutile d'étudier d'avantage ce cas.

4.6.1.2 Ensemble de Steiner variable

Une autre extension, qui est directe et peut sembler naturelle, consiste à chercher un arbre de Steiner (des nœuds et des arêtes) qui varie au cours du temps.

Définition 18 (Ensemble de Steiner variable)**.** *Soit* $G = (G_\theta)_{1 \leqslant \theta \leqslant T}$ *un graphe dynamique tel que :*

- $G_\theta = (V, E_\theta),\ 1 \leqslant \theta \leqslant T$
- $\cup_{1 \leqslant \theta \leqslant T} E_\theta = E$

— *Les arêtes sont pondérées :* $w_{e_\theta} \geqslant 0 \ \forall e_\theta \in E_\theta$

Pour un ensemble donné de nœuds terminaux $S \subset V$*, on cherche,* $\forall \theta \in \mathcal{T}$*, un ensemble de nœuds* $V'_\theta \subset V$*, avec* $S \subset V'_\theta$*, et un ensemble d'arêtes* $E'_\theta \subset E_\theta$ *tels que :*

— $G'_\theta = (V'_\theta, E'_\theta)$ *est un graphe connexe* $\forall \theta \in \mathcal{T}$

— $\sum_{e'_\theta \in E'_\theta} w_{e'_\theta}$ *est minimum* $\forall \theta \in \mathcal{T}$

Ce problème est donc équivalent à T problèmes de Steiner sur un graphe statique, un à chaque pas de temps de l'intervalle. En considérant les t-graphes séparément, c'est comme si on ne considérait pas un graphe qui évolue mais T graphes statiques différents sans tenir compte de la connaissance que l'on peut avoir sur l'évolution du graphe. Comme précédemment, le cas dynamique n'apporte rien de significatif ici.

De plus, il serait très couteux de résoudre ce problème puisqu'il s'agirait de résoudre T problèmes de Steiner différents. Et l'on sait qu'il est déjà couteux de résoudre un problème de Steiner.

On pourra aussi s'interroger sur la pertinence d'une définition où l'ensemble V', à savoir les nœuds qui permettent d'assurer la connexion des terminaux, change au cours du temps, potentiellement de manière très importante.

D'un point de vue applicatif, dans un réseaux de communication par exemple, il semble plus intéressant de conserver les mêmes nœuds intermédiaires au cours du temps.

4.6.1.3 Ensemble de Steiner entièrement connecté

Nous proposons une autre définition possible. Les nœuds intermédiaires sont les mêmes au cours du temps mais les arêtes permettant la connexion de ces nœuds et des terminaux varient dans le temps.

Définition 19 (Ensemble de Steiner entièrement connecté)**.** *Soit* $G = (G_\theta)_{1 \leqslant \theta \leqslant T}$ *un graphe dynamique tel que :*

— $G_\theta = (V, E_\theta)$, $1 \leqslant \theta \leqslant T$

— $\cup_{1 \leqslant \theta \leqslant T} E_\theta = E$

— *Les arêtes sont pondérées :* $w_{e_\theta} \geqslant 0 \ \forall e_\theta \in E_\theta$

Pour un ensemble donné de nœuds terminaux $S \subset V$*, on cherche un ensemble fixe de nœuds* $V' \subset V$*, avec* $S \subset V'$*, et,* $\forall \theta \in \mathcal{T}$*, un ensemble d'arêtes* $E'_\theta \subset E_\theta$ *tels que :*

— $G'_\theta = (V', E'_\theta)$ *est un graphe connexe* $\forall \theta \in \mathcal{T}$

— $\sum_{\theta \in \mathcal{T}} \sum_{e'_\theta \in E'_\theta} w_{e'_\theta}$ *est minimum*

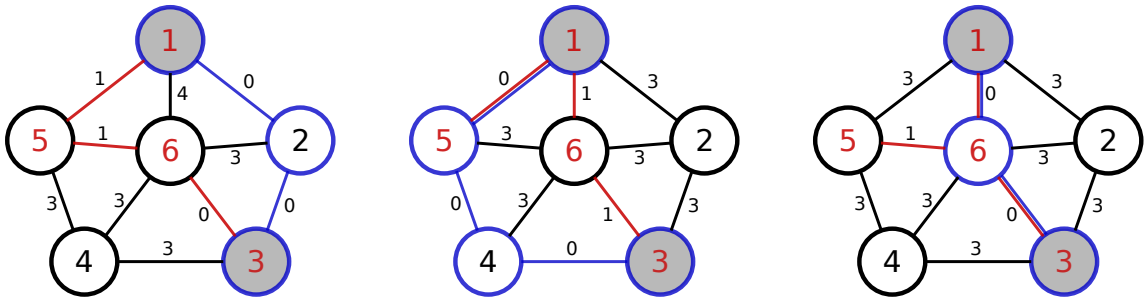
Nous cherchons ici un ensemble de nœuds intermédiaires ($V' \setminus S$) permettant d'assurer la connexion des terminaux de telle façon que le cout total de l'arbre couvrant V' à chaque pas de temps (ou le cout moyen de l'arbre couvrant V') soit minimum sur l'intervalle d'étude.

Dans cette définition, les nœuds terminaux et l'ensemble de Steiner V' sont connectés à chaque pas de temps.

Cette définition semble intéressante sur plusieurs points. En effet, une définition relativement simple donne un problème non trivial.

1. Tout d'abord, lorsque $T = 1$, il s'agit du problème de Steiner dans un graphe statique.
2. Les ensembles d'arête E'_θ permettent d'étudier le problème dans sa globalité au cours du temps, contrairement à la définition 17 où les arêtes choisies sont les mêmes à chaque pas de temps.
3. On peut noter que pour une date θ donnée, l'arbre de Steiner statique permettant de connecter les terminaux S dans G_θ ne fait pas nécessairement partie de la solution optimale du problème décrit par la définition 19.

4. Si l'on peut être tenté de faire l'union des arbres de Steiner optimaux à chaque pas de temps, cela n'apporte pas une solution optimale du problème auquel on s'intéresse ici.
5. On peut aussi être tenté de faire l'intersection des arbres de Steiner optimaux à chaque pas de temps. Dans le cas général, le sous-graphe obtenu après cette intersection n'est pas connexe.
6. Si l'on choisi $V' = V$, cela revient à chercher un arbre couvrant de poids minimum dans chaque t-graphe, et cela ne donne pas non plus la solution optimale de notre problème.



(a) G_1. (b) G_2. (c) G_3.

Figure 4.18 : Exemple de graphe dynamique sur 3 pas de temps. Les nœuds terminaux (l'ensemble S) sont en gris.

La figure 4.18 montre un exemple de graphe dynamique sur 3 pas de temps. Le poids des arêtes est indiqué à côté d'elles. Cet exemple permet de montrer les remarques faites ci-dessus.

Les nœuds gris (les nœuds 1 et 3) correspondent aux nœuds de l'ensemble de terminaux S. Les nœuds pour lesquels le numéro est indiqué en rouge (les nœuds 1, 3, 5 et 6) correspondent au sous-ensemble de nœuds V' de la solution optimale du problème décrit par la définition 19. Les arêtes en rouge correspondent aux sous-ensembles d'arêtes E'_θ de la solution optimale, pour $\theta \in \mathcal{T} = \{1;2;3\}$.

Les nœuds cerclés de bleu et les arêtes bleues correspondent aux arbres de Steiner optimaux à chaque pas de temps pour l'ensemble de terminaux S. Leurs couts sont toujours 0. Si l'on choisissait comme ensemble V' l'union des arbres de Steiner optimaux à chaque pas de temps de l'intervalle d'étude, on obtiendrait $V' = V$ dans cet exemple, et ces nœuds ne sont pas tous connectés à chaque pas de temps par des arêtes bleues de l'ensemble optimal.

Les nœuds de la solution optimale de cet exemple sont associés aux arêtes $(1;5)$, $(5;6)$ et $(3;6)$ au premier pas de temps pour un cout de 2, aux arêtes $(1;5)$, $(1;6)$ et $(3;6)$ au deuxième pas de temps pour un cout de 2 et aux arêtes $(1;6)$, $(5;6)$ et $(3;6)$ au dernier pas de temps pour un cout de 1. Le cout total optimal est 5.

Cet exemple illustre la remarque 3. En effet, les ensembles de nœuds bleus et rouges ne sont pas équivalents. Concernant la remarque 5, si l'on intersecte les arbres de Steiner dynamiques à chaque pas de temps (en d'autres termes, l'intersection des arêtes et nœuds bleus à chaque pas de temps), on obtient un stable. Il est donc impossible d'en déduire une solution au problème. Si à présent on fait l'union des nœuds des arbres de Steiner optimaux à chaque pas de temps (l'union des nœuds bleus), on obtient tous les nœuds du graphe, et comme on le dit dans la remarque 6, cette solution est très loin de la solution optimale. En effet, cette solution a un cout 15 dans cet exemple (arbre de cout 4 au premier et au deuxième pas de temps et de cout 7 au dernier pas de temps) alors que le cout de la solution optimale est 5.

Cette définition est certes pertinente et non triviale, elle pose cependant d'autres problèmes. Comme on vient de le voir, le problème de l'ensemble de Steiner entièrement connecté ne peut pas se réduire à

un problème de Steiner sur un graphe statique. C'est intéressant d'un point de vue algorithmique, mais cela implique de créer de toutes pièces une nouvelle méthode de résolution.

D'un point de vue pratique, cela semble relativement artificiel d'imposer que les t-graphes nommés G'_θ dans la définition 19 soient connexes. L'origine du problème étant de connecter les nœuds terminaux, si cet objectif est atteint, on peut s'interroger quant à la pertinence d'imposer en plus la connexion des nœuds intermédiaires à chaque pas de temps. Et la solution optimale de ce problème utilise des arêtes qui ne servent pas à connecter les nœuds terminaux (comme l'arête $(5;6)$ au troisième pas de temps de l'exemple figure 4.18), et cela augmente le cout de la solution.

4.6.1.4 Ensemble de Steiner partiellement connecté

En s'inspirant de la définition précédente, nous proposons les ensembles de Steiner partiellement connectés (définition 20 ci-dessous). Puisqu'il semblait artificiel de forcer tous les nœuds intermédiaires à être connectés, nous proposons cette fois-ci une définition où on cherche un ensemble de nœuds intermédiaires V' permettant de connecter les terminaux S à chaque pas de temps. On autorise les nœuds intermédiaires à être déconnectés. On cherche un ensemble d'arêtes E'_θ à chaque pas de temps θ de l'intervalle d'étude du graphe qui connectent les nœuds terminaux avec un poids minimum.

Définition 20 (Ensemble de Steiner partiellement connecté)**.** *Soit $G = (G_\theta)_{1 \leqslant \theta \leqslant T}$ un graphe dynamique tel que :*

- $G_\theta = (V, E_\theta)$, $1 \leqslant \theta \leqslant T$
- $\cup_{1 \leqslant \theta \leqslant T} E_\theta = E$
- *Les arêtes sont pondérées :* $w_{e_\theta} \geqslant 0 \ \forall e_\theta \in E_\theta$

Pour un ensemble donné de nœuds terminaux $S \subset V$, on cherche un ensemble fixe de nœuds $V' \subset V$, avec $S \subset V'$, et un ensemble d'arêtes $E'_\theta \subset E_\theta$ tels que :

- *Tous les nœuds de S font partie de la même composante connexe statique dans $G'_\theta = (V', E'_\theta)$ $\forall \theta \in \mathcal{T}$*
- $\sum_{\theta \in \mathcal{T}} \sum_{e'_\theta \in E'_\theta} w_{e'_\theta}$ *est minimum*

Comme c'était le cas pour la définition précédente, lorsque $T = 1$ et donc que l'on étudie un graphe statique, cette définition est équivalente à celle de l'arbre de Steiner.

Cette définition permet de connecter les terminaux par un ensemble de nœuds qui ne seraient pas toujours eux-mêmes connectés. Supposons par exemple que la composante connexe éternelle qui contient l'ensemble S ne contienne pas d'autre nœuds mais que les nœuds de S ne soient pas directement connectés entre eux. Dans ce cas, Il existe un ensemble de Steiner partiellement connecté pour connecter l'ensemble S mais il n'existe pas d'ensemble de Steiner entièrement connecté.

L'inconvénient de cette définition est qu'en l'état, puisque seuls les nœuds de l'ensemble S doivent être connectés et que nous cherchons à minimiser le poids des arêtes utilisées, alors une solution est de choisir $V' = V$, et de prendre les arêtes des arbres de Steiner statiques à chaque pas de temps. De cette façon, on obtient le poids minimum. Cette définition n'apporte donc rien par rapport à la définition 18.

4.6.1.5 Ensemble de Steiner minimum partiellement connecté

La définition précédente amène une solution qui ne présente que peu d'intérêt. Cependant elle devient plus intéressante si l'on cherche à minimiser la taille de l'ensemble V'. C'est ce que propose la définition 21.

Définition 21 (Ensemble de Steiner minimum partiellement connecté)**.** *Soit $G = (G_\theta)_{1 \leqslant \theta \leqslant T}$ un graphe dynamique tel que :*

- $G_\theta = (V, E_\theta)$, $1 \leqslant \theta \leqslant T$
- $\cup_{1 \leqslant \theta \leqslant T} E_\theta = E$

— *Les arêtes sont pondérées :* $w_{e_\theta} \geqslant 0 \ \forall e_\theta \in E_\theta$

Pour un ensemble donné de nœuds terminaux $S \subset V$*, on cherche un ensemble fixe de nœuds* $V' \subset V$*, avec* $S \subset V'$*, et un ensemble d'arêtes* $E'_\theta \subset E_\theta$ *tels que :*

— *Tous les nœuds de S font partie de la même composante connexe statique dans* $G'_\theta = (V', E'_\theta)$ $\forall \theta \in \mathcal{T}$

— $|V'|$ *est minimum*

— $\sum_{\theta \in \mathcal{T}} \sum_{e'_\theta \in E'_\theta} w_{e'_\theta}$ *est minimum parmi les solutions où* $|V'|$ *est minimum*

On notera tout de même que cette définition, bien qu'intéressante, ne correspond pas exactement au problème de Steiner statique si $T = 1$. En effet, dans le problème de Steiner d'origine, le nombre de nœuds intermédiaire n'est pas minimum. Il s'agit cependant d'une variante classique.

4.6.2 Cas particuliers

Les ensembles de Steiner minimum partiellement connectés nous semblent pertinents à étudier. Nous allons donc nous concentrer sur cette définition (définition 21). Le problème ayant l'air relativement complexe, nous nous intéressons d'abord aux différents cas particuliers, spécifiquement ceux que l'on peut trouver dans le contexte statique.

Un premier cas particulier du problème de Steiner dans un contexte statique est le cas où tous les nœuds du graphe sont des nœuds terminaux. Ce cas se résout par un algorithme polynomial. En effet, lorsque tous les nœuds du graphe sont terminaux, alors il s'agit du problème de l'arbre couvrant de poids minimum. L'algorithme de Kruskal, de complexité $O(m \cdot \log(m))$, résout ce problème.

Dans le cas dynamique, il faudrait résoudre un arbre couvrant de poids minimum sur chaque t-graphe. Le problème reste donc très simple et le contexte dynamique n'apporte pas de difficulté supplémentaire.

Un deuxième cas particulier du problème de Steiner, lui aussi polynomial dans un contexte statique, est lorsque le nombre de terminaux est exactement 2. Dans le cas statique, il s'agit simplement d'une recherche de plus court chemin, qui se résout polynomialement en $O(m + n \cdot \log(n))$ grâce à l'algorithme de Dijkstra.

Dans le cas dynamique, ce problème n'est pas aussi simple. Comme le montre l'exemple de la figure 4.18 dans lequel le graphe possède seulement deux terminaux, on peut trouver de multiples façon de connecter ces nœuds au cours du temps. Si l'on minimise d'abord le nombre de nœuds intermédiaires, on voit que les terminaux peuvent être connectés avec un nœud supplémentaire à chaque pas de temps. Les nœuds 2 et 6 permettent tous les deux de connecter les terminaux 1 et 3. Si l'on choisit le nœud 2, cela coute 12. Si l'on choisit le nœud 6, cela coute 6.

Dans le contexte statique, le cas où les poids des arêtes sont unitaires équivaut à minimiser le nombre de nœuds intermédiaires pour connecter les nœuds terminaux. On sait que ce problème est NP-difficile. Il l'est même dans le cas des graphes bipartis (Garey et Johnson, 1979) et des graphes cordaux (White et al., 1985).

Dans le contexte dynamique, il est possible de considérer la définition 21 en ignorant les poids et en ne considérant que le critère de minimisation du nombre de nœuds intermédiaires, ce qui correspond à une extension du problème de Steiner dans un graphe statique avec des poids unitaires.

Nous avons choisi de nous focaliser sur un cas particulier du problème décrit en définition 21. Il s'agit du cas où on ne considère pas de poids sur les arêtes et où $|S| = 2$. C'est l'objet de la section suivante.

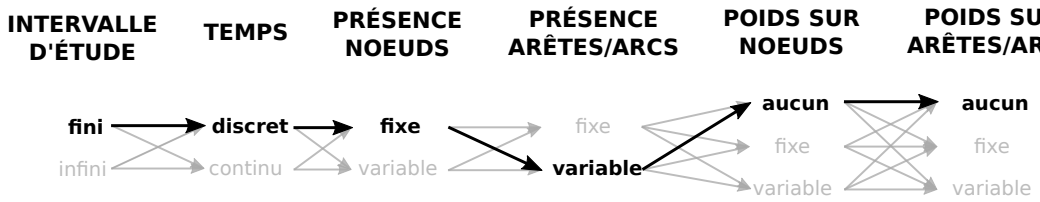


Figure 4.19 : Illustration du modèle de graphe dynamique étudié pour le problème de l'ensemble de Steiner dynamique minimum à 2 terminaux

4.6.3 Ensemble de Steiner dynamique minimum à 2 terminaux

On s'intéresse ici au problème basé sur la définition 21, sans poids et avec deux terminaux. Nous utilisons des exemples extrêmes pour montrer les différents cas possibles d'instances de ce problème. Enfin, nous sommes en mesure de montrer que ce problème est NP-complet.

4.6.3.1 Définition

Nous étudions ici des graphes dynamiques où les nœuds sont présents pendant tout l'intervalle d'étude. Les arêtes peuvent apparaitre et disparaitre au cours du temps. Comme toujours dans nos travaux, l'intervalle d'étude du graphe est un intervalle discret et fini. Nous ne considérons pas de poids sur les nœuds ou les arêtes. La figure 4.19 schématise le modèle de graphe que l'on étudie ici.

On cherche l'ensemble minimum de nœuds intermédiaires nécessaire afin de connecter les deux terminaux.

Définition 22 (Ensemble de Steiner dynamique minimum à 2 terminaux (DMSS2))**.** *Soit* $G = (G_\theta)_{1 \leqslant \theta \leqslant T}$ *un graphe dynamique tel que :*

- $G_\theta = (V, E_\theta)$, $1 \leqslant \theta \leqslant T$
- $\cup_{1 \leqslant \theta \leqslant T} E_\theta = E$

Pour un ensemble donné de nœuds terminaux $S \subset V$ *tel que* $|S| = 2$*, on cherche un ensemble fixe de nœuds* $V' \subset V$*, avec* $S \subset V'$*, et un ensemble d'arêtes* $E'_\theta \subset E_\theta$ *tels que :*

- *Tous les nœuds de S font partie de la même composante connexe statique dans* $G'_\theta = (V', E'_\theta)$ $\forall \theta \in \mathcal{T}$
- $|V'|$ *est minimum*

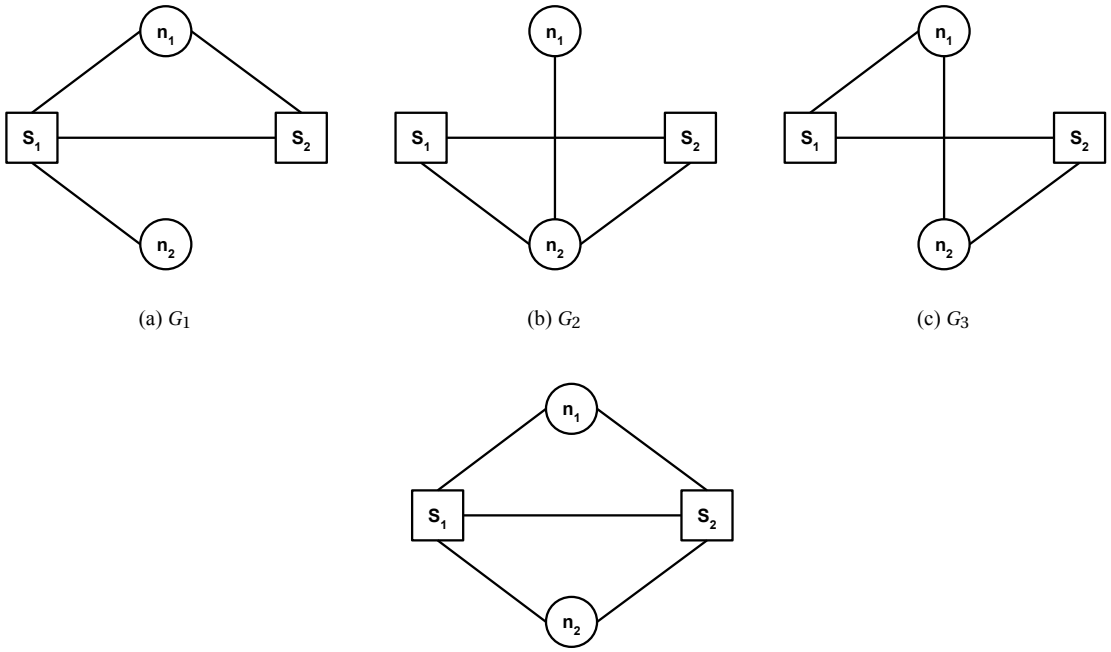
Dans la suite, nous désignons l'ensemble de Steiner dynamique minimum à 2 terminaux par l'acronyme DMSS2 (pour son nom en anglais *Dynamic Minimum Steiner Set, 2 Terminals*).

4.6.3.2 Exemples

À travers plusieurs exemples extrêmes, nous allons observer les différents cas que l'on peut rencontrer dans des instances du problème DMSS2.

La figure 4.20 montre un graphe dynamique sur 4 pas de temps. Les nœuds s_1 et s_2 sont les terminaux. On peut observer que dans chacun des t-graphes, il existe une arête entre s_1 et s_2. De ce fait, dans cet exemple, le nombre de nœuds intermédiaires nécessaires est 0.

Observons à présent l'exemple de la figure 4.21. Il s'agit d'un graphe dynamique sur n pas de temps. Les nœuds s_1 et s_2 sont les terminaux. Ce graphe a en plus n autre nœuds $n_1, \ldots, n_n$. Dans chaque t-graphe G_θ, il y a exactement deux arêtes : une première entre s_1 et n_θ et une seconde entre n_θ et s_2.



(a) G_1 (b) G_2 (c) G_3

(d) G_4

Figure 4.20 : Exemple d'instance du problème DMSS2. Les nœuds carrés sont les terminaux. Aucun nœud intermédiaire n'est nécessaire.

Dans un tel graphe, les terminaux sont toujours connectés par un chemin de longueur 2. Cependant le nœud assurant la connexion entre s_1 et s_2 est différent à chaque pas de temps. Dans ce cas, tous les nœuds du graphes doivent être sélectionnés afin de connecter les terminaux à chaque pas de temps.

Le graphe de la figure 4.22 montre un graphe sur 4 pas de temps. Les nœuds s_1 et s_2 sont les terminaux. Aux trois premiers pas de temps, il existe un chemin entre s_1 et s_2 passant par un seul nœud : n_3 dans G_1, n_4 dans G_2, n_5 dans G_3. Au dernier pas de temps, il existe une arête entre s_1 et s_2.

On ne peut pas déduire de ces plus courts chemins à chaque pas de temps la solution optimale du problème. En effet, en utilisant les nœuds présents sur tous les plus courts chemins à chaque pas de temps, on utiliserait 3 nœuds intermédiaires. Ici la solution optimale consiste à n'utiliser que deux nœuds intermédiaires (n_1 et n_2).

Grâce à ces exemples, cela permet de réaliser que le problème semble loin d'être trivial. C'est ce que nous montrons précisément dans la section suivante.

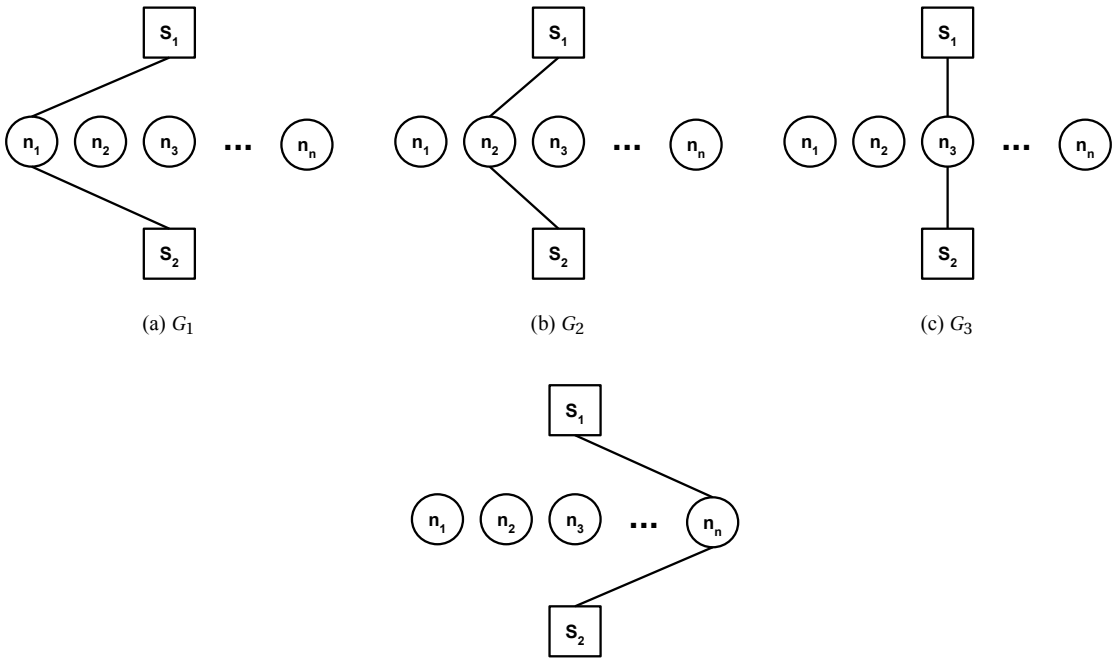
4.6.3.3 NP-complétude

Afin de prouver la complexité du problème DMSS2, nous devons nous pencher sur le problème de décision associé. Nous décrivons ce problème de décision dans la définition 23.

Nous prouvons que le problème DMSS2 est NP-complet grâce à une réduction polynomiale depuis le problème bien connu du *Vertex Cover* (VC), aussi appelé problème de couverture minimum par sommet en français. Nous rappelons le problème de décision du Vertex Cover dans la définition 24.

Définition 23 (Problème de décision du DMSS2)**.** *Soit* $G = (G_\theta)_{1 \leqslant \theta \leqslant T}$ *un graphe dynamique tel que :*

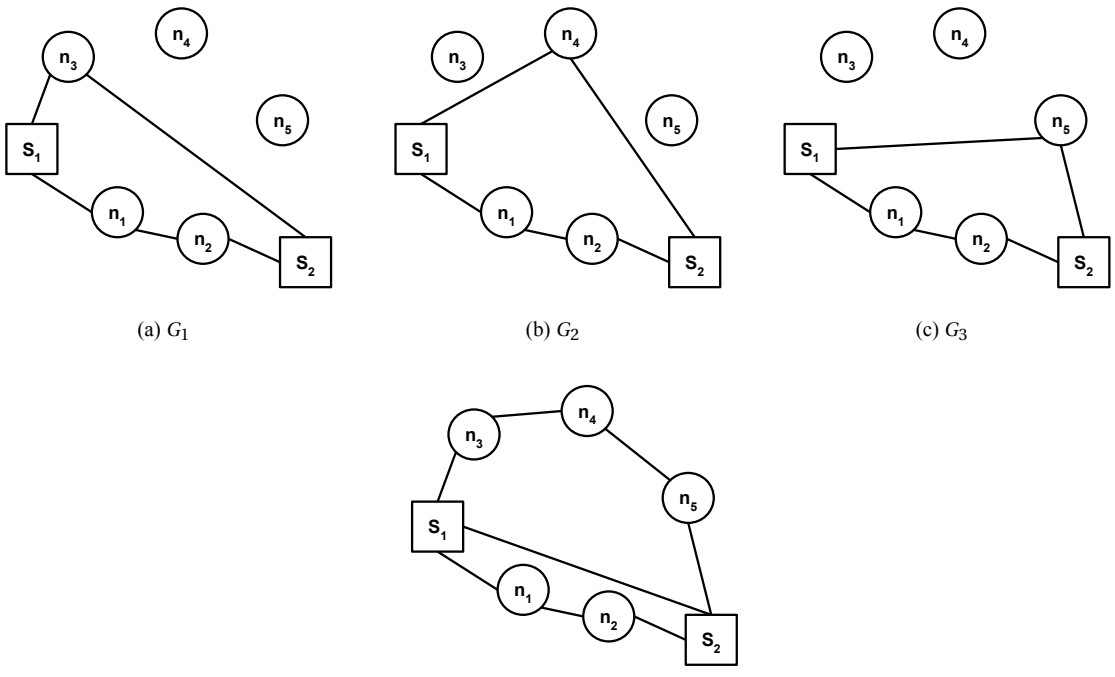
— $G_\theta = (V, E_\theta)$, $1 \leqslant \theta \leqslant T$



(a) G_1 (b) G_2 (c) G_3

(d) G_n

Figure 4.21 : Exemple d'instance du problème DMSS2. Les nœuds carrés sont les terminaux. Tous les nœuds du graphe sont nécessaires comme nœuds intermédiaires.



(a) G_1 (b) G_2 (c) G_3

(d) G_4

Figure 4.22 : Exemple d'instance du problème DMSS2. Les nœuds carrés sont les terminaux. Les plus courts chemins entre les terminaux ne donnent pas la solution optimale.

— $\cup_{1 \leqslant \theta \leqslant T} E_\theta = E$

Soient k un entier positif et $S \subset V$ un sous-ensemble de nœuds tel que $|S| = 2$. On définit un sous-ensemble de nœuds V_s tel que $S \subset V_s \subset V$ et S est inclus dans une composante connexe statique à chaque pas de temps $\theta \in \mathcal{T}$ dans le sous-graphe de G_θ induit par V_s.
***Question :** Existe-t-il un ensemble de nœuds V_s de taille k ?*

Définition 24 (Rappel du problème de décision du Vertex Cover)**.** *Soient $G = (V, E)$ un graphe (statique) et k un enter positif. On définit un sous-ensemble de nœuds $V_c \subset V$ tel que $\forall\ (i, j) \in E$, alors $i \in V_c$ ou $j \in V_c$.*
***Question :** Existe-t-il un ensemble de nœuds V_c de taille k ?*

La définition 24 pose le problème de décision du Vertex Cover dans un graphe statique. Il est possible de réduire polynomialement ce problème au problème de décision de DMSS2. Puisque VC est NP-complet, cela prouve que DMSS2 est aussi NP-complet. Nous le montrons dans la suite.

Lemme 21 (Transformation vers DMSS2)**.** *Il existe une transformation polynomiale permettant de transformer une instance de VC en une instance de DMSS2.*

Démonstration. À partir d'un graphe statique $G = (V, E)$ et d'un ensemble de Vertex Cover V_c, on construit un graphe dynamique G^{DYN} ayant un sous ensemble de nœuds V_s de la façon suivante :

— À chaque nœud i du graphe G correspond un nœud i dans le graphe dynamique G^{DYN}.
— Le graphe dynamique G^{DYN} possède deux nœuds supplémentaire a et b qui sont les deux nœuds terminaux.
— À chaque arête $e = (i; j)$ dans le graphe G correspond un pas de temps θ_e dans le graphe dynamique G^{DYN} et $G^{DYN}_{\theta_e}$ possède exactement 4 arêtes : $(a; i)$, $(i; b)$, $(a; j)$, $(j; b)$.
— Les nœuds de l'ensemble V_s sont exactement les nœuds de l'ensemble V_c.

La figure 4.23 montre un exemple d'une telle transformation. L'instance de Vertex Cover est représentée dans la figure 4.23a. Les nœuds de l'ensemble V_c sont indiqués en bleu. Les figures 4.23b à 4.23f montrent l'instance de DMSS2 correspondante. Chaque pas de temps du graphe G^{DYN} est représenté. Les nœuds a et b sont les terminaux et les nœuds de l'ensemble V_s sont indiqués en bleu.

Si le graphe G possède n nœuds et m arêtes, alors le graphe dynamique G^{DYN} évolue sur m pas de temps et possède $n+2$ nœuds. Chaque t-graphe G^{DYN}_θ a exactement 4 arêtes. Il y a une correspondance directe entre les nœuds de V_c et ceux de V_s.

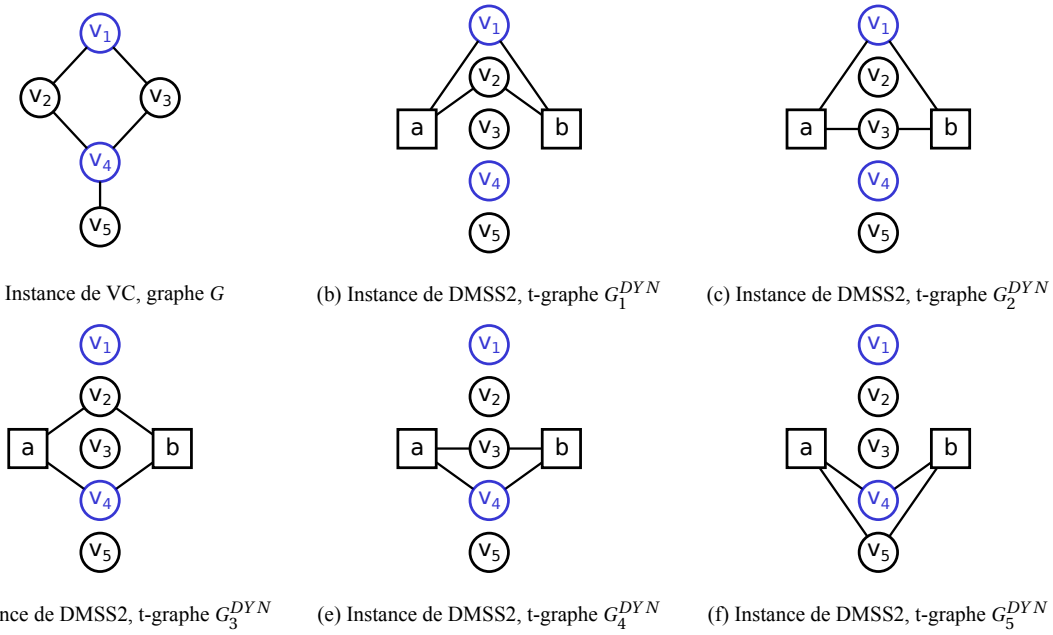
L'instance de DMSS2 est donc construite en temps polynomial $O(n + m)$ à partir d'une instance de VC. □

Lemme 22 (Obtention d'une solution de DMSS2)**.** *Dans le cadre de la transformation polynomiale du lemme 21, une instance valide de VC donne une instance valide de DMSS2.*

Démonstration. Considérons une instance valide de Vertex Cover du graphe G et l'ensemble de nœuds V_c correspondant. Soit V_s un sous-ensemble de nœuds de G^{DYN} avec les mêmes nœuds que V_c. L'ensemble V_s est donc aussi de taille k.

Considérons un pas de temps θ donné. Le t-graphe correspondant G^{DYN}_θ contient exactement deux chemins distincts (en termes d'arêtes et de nœuds) de longueur 2 entre les nœuds a et b. Le premier chemin passe par un nœud i et le second passe par un nœud j. Par construction de G^{DYN}, on sait que $(i; j)$ est une arête du graphe G. Donc soit i est dans l'ensemble V_c, soit j est dans l'ensemble V_c (ou les deux y sont). Puisque les ensembles V_c et V_s contiennent les mêmes nœuds, alors dans G^{DYN} on a $i \in V_s$ ou $j \in V_s$ (ou $i, j \in V_s$). Puisque l'un de ces nœuds au moins est dans V_s, alors il existe dans G^{DYN}_θ un chemin (au moins) entre a et b ne passant que par des nœuds de V_s.

Cela reste vrai quel que soit le pas de temps $\theta \in \mathcal{T}$. Donc V_s est bien un ensemble valide pour le problème DMSS2. □



(a) Instance de VC, graphe G

(b) Instance de DMSS2, t-graphe G_1^{DYN}

(c) Instance de DMSS2, t-graphe G_2^{DYN}

(d) Instance de DMSS2, t-graphe G_3^{DYN}

(e) Instance de DMSS2, t-graphe G_4^{DYN}

(f) Instance de DMSS2, t-graphe G_5^{DYN}

Figure 4.23 : Exemple de transformation polynomiale d'une instance de VC ayant 5 arêtes vers une instance de DMSS2 avec 5 pas de temps. Les ensembles $V_c = V_s = \{v_1; v_4\}$ sont indiqués en bleu.

Lemme 23 (Obtention d'une solution de VC)**.** *Dans le cadre de la transformation polynomiale du lemme 21, une instance valide de DMSS2 donne une instance valide de VC.*

Démonstration. Considérons une instance valide du problème DMSS2 dans le graphe G^{DYN} et l'ensemble V_s correspondant de taille k. Soit V_c un sous-ensemble de nœuds du graphe G contenant les mêmes nœuds que V_s. Il est donc évident que l'ensemble V_c est aussi de taille k.

Soit $e = (i; j)$ une arête de G. Au pas de temps correspondant θ_e, le t-graphe $G_{\theta_e}^{DYN}$ a exactement deux chemins distincts de longueur 2 entre a et b, un premier passant par le nœud i et un second passant par le nœud j. Puisque V_s est un ensemble valide pour le problème DMSS2, alors cela signifie que soit i, soit j est dans l'ensemble V_s (soit i et j sont tous les deux dans l'ensemble V_s). Puisque l'ensemble V_c contient les mêmes nœuds que V_s, alors dans le graphe G, on a $i \in V_c$ ou $j \in V_c$ ou $i, j \in V_c$. Et donc l'arête e a une de ses extrémités au moins dans l'ensemble V_c.

Cela reste vrai pour toutes les arêtes de G. Donc V_c est bien un ensemble valide pour le problème du Vertex Cover. □

Théorème 24 (Complexité du problème DMSS2)**.** *Le problème de décision du DMSS2 est NP-complet.*

Démonstration. Immédiat grâce aux lemmes 21, 22 et 23. □

Si $T = 1$, le problème DMSS2 s'interroge simplement sur la longueur du plus court chemin en nombre de sauts entre les deux nœuds terminaux. Il s'agit d'un problème polynomial. Dans sa version en contexte dynamique, ce problème est NP-complet.

4.6.4 Pistes algorithmiques

Nous venons de prouver que même la variante la plus simple du problème décrit par la définition 21 est NP-complète. Même sans considérer les poids, et lorsque nous avons seulement deux terminaux, le problème est NP-complet.

Cela veut dire qu'il n'est pas envisageable de le résoudre à l'optimum dans le cas général. Cependant, comme pour le problème de l'arbre de Steiner dans un graphe statique, lui aussi NP-complet, des algorithmes pseudo-polynomiaux peuvent résoudre le problème de façon optimale et être efficace dans certains cas particuliers.

Si l'on fixe la taille de l'ensemble de Steiner que l'on souhaite, alors il est possible de trouver des algorithmes dont la complexité dépend de ce paramètre.

En effet, on peut chercher spécifiquement un ensemble de Steiner de taille k satisfaisant la définition 19 ou la définition 20 pour connecter les terminaux S. On doit donc sélectionner, parmi les nœuds du graphe V, tous les ensembles V' de taille k qui contiennent S. Pour cela on cherche tous les ensembles de taille $k-|S|$ dans l'ensemble $V \setminus S$. Il y en a $\binom{n-|S|}{k-|S|}$.

Une fois les potentiels ensembles V' trouvés, il faut vérifier que les nœuds de S appartiennent à la même composante connexe dans le sous-graphe induit par l'ensemble V' de chaque t-graphe, autrement dit, que les nœuds de S font partie de la même composante connexe éternelle dans le sous-graphe induit par V'. On peut ensuite éliminer les ensembles ne satisfaisant pas cette condition.

Pour les ensembles restants V', si on s'intéresse à la définition 19, on calcule pour chacun le poids total des arbres couvrants de poids minimum à chaque pas de temps. Si on s'intéresse à la définition 20, on calcule pour chacun le poids total de l'arbre de Steiner statique connectant les nœuds de S dans le sous-graphe induit par V' à chaque pas de temps (puisqu'il n'est pas obligatoire de connecter tous les nœuds de V'). On peut ensuite choisir l'ensemble qui a le poids le plus faible.

Si l'on souhaite utiliser cette méthode en pratique, il ne faut pas que le graphe ait trop de nœuds, ni que l'ensemble de Steiner que l'on recherche soit trop grand. Et pour la définition 20 en particulier, puisqu'il est nécessaire d'utiliser la résolution d'un arbre de Steiner, il faut aussi pouvoir faire cela efficacement.

Ces questions ont fait l'objet d'un stage de recherche de master dans l'équipe (Moal, 2020).

En résumé, le paramètre k indiquant la taille de l'ensemble que l'on souhaite va énormément influer sur le temps de calcul. Ce n'est pas utilisable à large échelle.

Si l'on se donne comme objectif de trouver un ensemble de Steiner en temps polynomial, dans ce cas il ne faut pas chercher à résoudre le problème à l'optimum, et il faut s'intéresser à des algorithmes d'approximation.

4.7 Conclusion

L'ensemble des composantes connexes d'un graphe peut être une information importante à elle seule. Mais cela peut aussi servir de condition de départ pour d'autres problèmes.

D'autre part, la connexité ne concerne pas uniquement le calcul de composantes connexes. Il existe d'autres problèmes liés aux notions de connexité qui ont été étudiés dans les graphes.

Pour toutes ces raisons, nous avons travaillé sur la connexité en contexte dynamique. Dans un premier temps, nous nous sommes intéressés aux composantes connexes et nous avons étudié le problème de Steiner dans un second temps.

Nous nous sommes questionnés sur ce que pouvait signifier, pour un ensemble de nœuds, le fait d'être une composante connexe dans un graphe dynamique, en étant le plus proche possible de la définition statique. Nous avons apporté, comme réponse à cette question, la définition des composantes connexes persistantes, qui traduisent la connexité d'un ensemble de nœuds au cours du temps. Nous avons également défini une relation de dominance entre les composantes.

En nous appuyant sur la définition de composantes connexes persistantes, nous avons été en mesure de définir d'autres types de composantes dans les graphes dynamiques, ayant des propriétés particulières. Nous avons proposé trois extensions possibles. Les composantes fortement connexes persistantes ne présentent pas de difficultés supplémentaires par rapport aux composantes connexes persistantes. Les composantes éternelles sont un cas simple et il est possible de trouver toutes les composantes de ce type en temps linéaire en m et linéaire en T. À l'inverse, les composantes connexes persistantes discontinues ne peuvent pas être trouvées facilement du fait qu'un graphe dynamique en possède potentiellement un nombre exponentiel.

Nous avons créé un algorithme polynomial, quadratique en n et linéaire en T, permettant d'extraire d'un graphe dynamique donné toutes les composantes connexes persistantes non-dominées. Nous avons mené une étude expérimentale sur cet algorithme qui nous a permis de constater que son comportement pratique correspond au comportement théorique que nous attendions, déterminé par la complexité de l'algorithme. L'expérience a aussi montré que l'algorithme permet de résoudre, en pratique, le problème de recherche de composantes connexes persistantes non dominées dans des graphes de grande taille avec des horizons de temps importants. L'étude expérimentale que nous avons menée, qui a porté sur 4 modèles de graphes différents, nous a de plus permis d'observer que la taille et la durée des composantes connexes dépend grandement de la structure du graphe étudié.

Nous nous sommes ensuite intéressés au problème de Steiner en commençant par nous interroger sur la définition que pouvait avoir ce problème dans un graphe dynamique. Le problème d'origine a pour but de connecter un ensemble de nœuds. Nous avons donc réfléchi à une version du problème de Steiner dans un graphe ayant le même objectif. Nous avons donc envisagé plusieurs variantes, dont le but était de connecter un ensemble de nœuds dans le temps, qui présentaient toutes des inconvénients, avant d'arriver à la définition du problème de l'ensemble de Steiner dynamique minimum partiellement connecté.

Nous avons étudié un cas particulier de ce problème, dans lequel il n'y a pas de poids sur les arêtes et où seuls deux nœuds du graphe doivent être connectés dans le temps. La version statique de ce problème est très simple et se résout polynomialement par la recherche d'un plus court chemin. Nous avons pu prouver, à l'aide d'une réduction polynomiale depuis le problème de couverture par sommets, que le problème de décision associé, dans le cas dynamique, est NP-complet.

Notre travail sur les composantes connexes persistantes nous a permis de trouver un algorithme de résolution efficace. On rappelle que contrairement à la plupart des travaux de la littérature s'intéressant aux questions de connexité, l'algorithme que l'on propose pour résoudre notre problème est polynomial.

Mais nous pouvons cependant nous interroger sur la possibilité de trouver un meilleur algorithme. Il semble néanmoins peu concevable que l'on puisse faire l'économie d'au moins un parcours de graphe à chaque pas de temps de l'intervalle d'étude.

Nous pouvons aussi poursuivre nos efforts sur cette notion de connexité. On peut envisager d'utiliser cette définition de composantes connexes comme base pour d'autres travaux sur les graphes dynamiques qui nécessiteraient la vérification d'une condition de connexité.

La résolution du problème de l'ensemble de Steiner dynamique minimum partiellement connecté n'est pas envisageable par un algorithme exact pour des ensembles de Steiner de taille quelconque. La version sans poids et avec deux terminaux étant déjà NP-complète, on ne pourra pas développer un algorithme polynomial. Si l'on souhaite malgré tout résoudre ce problème, deux choix s'offrent à nous.

On peut envisager la recherche d'un algorithme exact pseudo-polynomial. Il faudra ensuite, pour pouvoir utiliser un tel algorithme en pratique, identifier les cas particuliers dans lesquels l'algorithme est efficace.

L'autre possibilité pour résoudre un tel problème est de ne pas tenter la résolution exacte. On peut imaginer des algorithmes d'approximation, dont la complexité serait polynomiale, et qui présenteraient des résultats proches de l'optimum, voire l'optimum dans certains cas.

Chapitre 5

Conclusion générale

Ce travail était consacré aux problèmes de graphes en contexte dynamique. La question sous-jacente était de s'interroger sur les problèmes de graphes et comment ils pouvaient être étendus lorsque le graphe est dynamique. Pour quelques problèmes de graphes, notre objectif était de trouver le modèle de graphe dynamique approprié, de définir clairement le problème dans le contexte dynamique et de proposer un algorithme pour le résoudre.

Selon cet objectif, nous avons travaillé sur des problèmes de flots, flot maximum et flot de cout minimum. Nous avons travaillé sur la connexité en nous intéressant aux composantes connexes dans un graphe dynamique puis au problème de Steiner.

Nous avons considéré le problème du flot maximum dans un graphe dynamique sans temps de traversée et avec stockage illimité de flot sur les nœuds du graphe. Nous avons exploré la piste des capacités cumulées qui exploite toute la connaissance que l'on a du graphe en prenant en compte l'évolution des capacités sur tout l'intervalle d'étude, sur tous les arcs du graphe. Cette méthode permet d'obtenir une borne supérieure sur le flot maximum. Nous avons pu montrer par l'expérimentation que la borne supérieure n'est pas toujours atteinte, mais qu'elle est de bonne qualité.

Nous avons travaillé sur le flot de cout minimum sur des graphes dynamiques sans temps de traversée et sans stockage. Nous avons développé une méthode, inspirée de l'algorithme *Successive Shortest Path* pour le flot de cout minimum dans un graphe statique, qui calcule le flot optimal avec une complexité théorique $O((Q+T)\cdot(m+n\cdot\log(n)+\log(T)))$, ce qui est polynomial en T, en n et en m. Cette complexité est nettement meilleure que celle de la méthode sur le graphe développé. Nous avons réalisé une étude expérimentale qui a montré que le comportement pratique de l'algorithme est cohérent avec le comportement théorique défini par sa complexité. Cette étude a aussi montré que le temps d'exécution, raisonnable, de l'algorithme permet son utilisation en pratique. Ce travail a abouti à la publication d'un article (Vernet et al., 2020).

Nous avons ensuite travaillé sur la notion de composantes connexes dans un graphe dynamique. Nous avons proposé la définition des composantes connexes persistantes qui traduisent le fait, pour un ensemble de nœuds, de rester connectés pendant un certain temps. Nous avons défini une notion de dominance entre les composantes basée sur leur taille (en nombre de nœuds) et leur durée (en nombre de

pas de temps de présence). Nous avons proposé un algorithme qui permet d'identifier les composantes connexes persistantes non-dominées en temps $O(n^2 \cdot T)$. Nous avons mené une étude expérimentale qui a montré que le comportement pratique de l'algorithme correspond au comportement théorique attendu déterminé par sa complexité. Le temps de calcul raisonnable de l'algorithme permet d'envisager son utilisation en pratique. De plus, la structure de cet algorithme permet de l'utiliser sur des modèles de graphes en ligne. L'algorithme a été testé sur différents types de graphes. Cela a permis de montrer que la structure du graphe sous-jacent influe sur la taille et la durée des composantes. De même, la taille et la durée des composantes influent sur le temps de calcul pour les identifier. Nous avons aussi proposé des extensions possibles de la définition des composantes connexes persistantes. Nous avons ainsi défini les composantes fortement connexes persistantes constituées d'un ensemble de nœuds restant fortement connectés pendant un certain temps dans un graphe dynamique orienté. Notre algorithme défini pour les graphes non orientés permet d'identifier ces composantes dans un graphe dynamique orienté, après une modification mineure. Nous avons aussi défini les composantes connexes éternelles qui correspondent à un ensemble de nœuds qui restent connectés pendant tout l'intervalle d'étude. Nous avons proposé un algorithme de complexité $O(m \cdot T)$ afin d'identifier toutes les composantes connexes éternelles. Nous avons aussi défini les composantes connexes persistantes discontinues qui correspondent à un ensemble de nœuds connectés pendant plusieurs pas de temps non nécessairement consécutifs. Nous avons prouvé que le nombre de telles composantes est exponentiel et qu'il est donc inenvisageable de les identifier par un algorithme polynomial.

Nous nous sommes intéressés au problème de Steiner dans un graphe dynamique. Nous avons proposé plusieurs définitions possibles. Nous nous sommes concentrés sur une variante avec deux nœuds terminaux, et aucun poids. Dans un graphe statique, cette variante du problème est polynomiale puisqu'il s'agit de chercher un plus court chemin en nombre d'arêtes entre les deux terminaux. Nous avons prouvé que le problème est NP-complet dans un graphe dynamique.

En résumé, nous avons étudié différents problèmes. Pour chacun, nous avons défini le problème clairement dans le cadre des graphes dynamiques et proposé des algorithmes polynomiaux lorsqu'il était possible de le faire. Nous pouvons dire que ce travail a apporté des éléments nouveaux à la connaissance que l'on a des problèmes de graphes en contexte dynamique.

Les problèmes de flots peuvent se résoudre simplement grâce au graphe développé. Mais cette méthode est inefficace car très couteuse. C'est pourquoi il est pertinent de chercher à concevoir un algorithme n'utilisant pas le graphe développé. Ce que nous avons proposé exploite les spécificités du modèle de graphe dynamique afin de d'atteindre une complexité théorique bien meilleure que celle des méthodes utilisant le graphe développé.

La connexité dans un contexte dynamique a été beaucoup moins étudiée que les problèmes de flot. Nous avons apporté une nouvelle définition, qui introduit un concept différent de ce qui peut être trouvé dans la littérature. Nous avons proposé un algorithme spécifique car le problème ne se ramène pas à un problème sur un graphe statique.

Notre travail préliminaire sur le problème de Steiner a permis d'identifier un cas particulier intéressant où la variante dynamique du problème, qui est polynomial dans un graphe statique, est NP-complète.

On peut maintenant s'interroger sur la manière dont ce travail peut être poursuivi. Les méthodes que nous avons proposées pour les problèmes de flots ne peuvent pas être étendues directement. En effet, la méthode par capacités cumulées nous donne uniquement une borne supérieure. On peut tout de même utiliser cette borne pour obtenir des garanties de performance sur des algorithmes approchés de faible complexité. Notons également qu'une borne inférieure évidente s'obtient en résolvant le problème sans stockage. Quant à la méthode pour le flot de cout minimum, elle repose intégralement sur le modèle de graphe sans temps de traversée et sans stockage. Il n'est donc pas possible d'étendre cette méthode à un

modèle avec temps de traversée ou stockage. Une direction possible vers laquelle ce travail pourrait se poursuivre serait d'étudier des modèles de graphes avec incertitudes. En effet, nous n'avons considéré que des modèles pour lesquels on connait le graphe dans son ensemble et où les évolutions du graphes étaient aussi connues à l'avance. Il serait donc intéressant de s'interroger sur les problèmes de flots dans des modèles où l'évolution du graphe suivrait une loi de probabilité ou pour lesquels la connaissance du graphe ne serait que locale.

Dans notre travail sur les composantes connexes persistantes, nous avons déjà considéré un certain nombre d'extensions possibles. Il serait maintenant intéressant de proposer des cas d'utilisation de ces définitions. Il est possible d'envisager soit des cas d'application directs des composantes connexes persistantes, soit des problèmes pour lesquels un prérequis serait d'identifier ce type de composantes.

Nous avons prouvé qu'avec deux terminaux déjà, le problème de Steiner dans un graphe dynamique est NP-complet. Il n'est donc pas envisageable de concevoir un algorithme polynomial exact. Un algorithme exact pseudo-polynomial peut être une piste pour résoudre ce problème. Il sera alors nécessaire d'identifier les cas particuliers pour lesquels un tel algorithme est efficace. Une autre possibilité pour résoudre ce problème serait de passer par des algorithmes approchés, comme c'est le cas de beaucoup de travaux traitant du problème de Steiner dans un graphe statique.

Dans ce travail, nous avons étudié les flots et les problèmes de connexité dans des graphes dynamiques. Nous avons redéfini ces problèmes pour les adapter au contexte dynamique et aux modèles de graphes que l'on étudiait. Nous avons proposé des algorithmes exacts et polynomiaux afin de résoudre ces problèmes.

Nous avons apporté une contribution aux problèmes de graphes dynamiques. Il est possible de poursuivre ce travail, en étudiant d'autres problèmes de graphes, ou en approfondissant ce que nous avons réalisé.

Bibliographie et index

Bibliographie

Agrawal, A., P. Klein, et R. Ravi (1995). When trees collide : An approximation algorithm for the generalized steiner problem on networks. *SIAM Journal on Computing 24*(3), 440–456.

Ahuja, R. K., T. L. Magnanti, et J. B. Orlin (1993). Network flows : theory, algorithms, and applications.

Akrida, E. C., J. Czyzowicz, L. Gąsieniec, Łukasz Kuszner, et P. G. Spirakis (2019). Temporal flows in temporal networks. *Journal of Computer and System Sciences 103*, 46 – 60.

Akrida, E. C. et P. G. Spirakis (2019). On verifying and maintaining connectivity of interval temporal networks. *Parallel Processing Letters 29*(02).

Albert, R. et A.-L. Barabási (2002). Statistical mechanics of complex networks. *Reviews of modern physics 74*(1), 47–97.

Anderson, E. J., P. Nash, et A. B. Philpott (1982). A class of continuous network flow problems. *Mathematics of Operations Research 7*(4), 501–514.

Bampis, E., B. Escoffier, M. Lampis, et V. T. Paschos (2018). Multistage matchings.

Bassett, D. S., N. F. Wymbs, M. P. Rombach, M. A. Porter, P. J. Mucha, et S. T. Grafton (2013). Task-based core-periphery organization of human brain dynamics. *PLoS Comput Biol 9*(9), e1003171.

Baste, J., B.-M. Bui-Xuan, et A. Roux (2020). Temporal matching. *Theoretical Computer Science 806*, 184 – 196.

Baumann, N. et E. Köhler (2007). Approximating earliest arrival flows with flow-dependent transit times. *Discrete Applied Mathematics 155*(2), 161–171.

Bertsekas, D. P., R. G. Gallager, et P. Humblet (1992). *Data networks*, Volume 2. Prentice-Hall International New Jersey.

Bhadra, S. et A. Ferreira (2003). Complexity of connected components in evolving graphs and the computation of multicast trees in dynamic networks. In *International Conference on Ad-Hoc Networks and Wireless*, pp. 259–270. Springer.

Bui-Xuan, B.-M., A. Ferreira, et A. Jarry (2003). Computing shortest, fastest, and foremost journeys in dynamic networks. *International Journal of Foundations of Computer Science 14*(02), 267–285.

Cai, X., D. Sha, et C. Wong (2001). Time-varying minimum cost flow problems. *European Journal of Operational Research 131*(2), 352 – 374. Artificial Intelligence on Transportation Systems and Science.

Casteigts, A., P. Flocchini, W. Quattrociocchi, et N. Santoro (2012). Time-varying graphs and dynamic networks. *International Journal of Parallel, Emergent and Distributed Systems 27*(5), 387–408.

Casteigts, A., R. Klasing, Y. M. Neggaz, et J. G. Peters (2015). Efficiently testing t-interval connectivity in dynamic graphs. In *International Conference on Algorithms and Complexity*, pp. 89–100. Springer.

Casteigts, A., J. G. Peters, et J. Schoeters (2019). Temporal Cliques Admit Sparse Spanners. In C. Baier, I. Chatzigiannakis, P. Flocchini, et S. Leonardi (Eds.), *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, Volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Dagstuhl, Germany, pp. 134 :1–134 :14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

Cattuto, C., M. Quaggiotto, A. Panisson, et A. Averbuch (2013). Time-varying social networks in a graph database : a neo4j use case. In *First international workshop on graph data management experiences and systems*, pp. 1–6.

Clementi, A. E., C. Macci, A. Monti, F. Pasquale, et R. Silvestri (2008). Flooding time in edge-markovian dynamic graphs. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pp. 213–222.

Démare, T., C. Bertelle, A. Dutot, et L. Lévêque (2017). Modeling logistic systems with an agent-based model and dynamic graphs. *Journal of Transport Geography 62*, 51 – 65.

Dutot, A., F. Guinand, D. Olivier, et Y. Pigné (2007). Graphstream : A tool for bridging the gap between complex systems and dynamic graphs. In *Emergent Properties in Natural and Artificial Complex Systems. Satellite Conference within the 4th European Conference on Complex Systems (ECCS'2007)*.

Erdős, P. et A. Rényi (1960). On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci 5*(1), 17–60.

Fleischer, L. et M. Skutella (2003). Minimum cost flows over time without intermediate storage. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 66–75. Society for Industrial and Applied Mathematics.

Fleischer, L. et É. Tardos (1998). Efficient continuous-time dynamic network flow algorithms. *Operations Research Letters 23*(3), 71–80.

Ford, L. R. et D. R. Fulkerson (1956). Maximal flow through a network. *Canadian journal of Mathematics 8*(3), 399–404.

Ford, L. R. et D. R. Fulkerson (1958). Constructing maximal dynamic flows from static flows. *Operations Research 6*(3), 419–433.

Ford, L. R. et D. R. Fulkerson (1962). *Flows in networks*. Princeton University Press.

Garey, M. R. et D. S. Johnson (1979). *Computers and intractability*, Volume 174. Freeman San Francisco.

Gómez-Calzado, C., A. Casteigts, A. Lafuente, et M. Larrea (2015). A connectivity model for agreement in dynamic systems. In *European Conference on Parallel Processing*, pp. 333–345. Springer.

Grande, E., G. Nicosia, A. Pacifici, et V. Roselli (2018). An exact algorithm for a multicommodity min-cost flow over time problem. *Electronic Notes in Discrete Mathematics 64*, 125–134.

Guinand, F. et Y. Pigné (2015). Time considerations for the study of complex maritime networks. In *Maritime Networks : Spatial structures and time dynamics*, Number 1, pp. 163–189. Routledge.

Gupta, A., K. Talwar, et U. Wieder (2014). Changing bases : Multistage optimization for matroids and matchings. In *International Colloquium on Automata, Languages, and Programming*, pp. 563–575. Springer.

Halpern, J. (1979). A generalized dynamic flows problem. *Networks 9*(2), 133–167.

Hashemi, S. M. et E. Nasrabadi (2012). On solving continuous-time dynamic network flows. *Journal of Global Optimization*, 1–28.

Holme, P. (2015). Modern temporal network theory : a colloquium. *The European Physical Journal B 88*(9), 234.

Hoppe, B. et É. Tardos (1994). Polynomial time algorithms for some evacuation problems. In *SODA*, Volume 94, pp. 433–441.

Hoppe, B. et É. Tardos (2000). The quickest transshipment problem. *Mathematics of Operations Research 25*(1), 36–62.

Huang, S., A. W.-C. Fu, et R. Liu (2015). Minimum spanning trees in temporal graphs. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pp. 419–430. ACM.

Huyghues-Despointes, C., B.-M. Bui-Xuan, et C. Magnien (2016). Forte Δ-connexité dans les flots de liens. In *ALGOTEL 2016-18èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications*. Abstract in English : Strong Δ-connectivity in link streams.

Imase, M. et B. M. Waxman (1991). Dynamic steiner tree problem. *SIAM Journal on Discrete Mathematics 4*(3), 369–384.

Jarry, A. et Z. Lotker (2004). Connectivity in evolving graph with geometric properties. In *Proceedings of the 2004 joint workshop on Foundations of mobile computing*, pp. 24–30. ACM.

Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, pp. 85–103. Boston, MA : Springer US.

Kempe, D., J. Kleinberg, et A. Kumar (2002). Connectivity and inference problems for temporal networks. *Journal of Computer and System Sciences 64*(4), 820–842.

Khodaverdian, A., B. Weitz, J. Wu, et N. Yosef (2016). Steiner network problems on temporal graphs. *arXiv preprint arXiv :1609.04918*.

Khodayifar, S., M. Raayatpanah, et P. Pardalos (2017). A polynomial time algorithm for the minimum flow problem in time-varying networks. *Annals of Operations Research*, 1–11.

Klinz, B. et G. J. Woeginger (2004). Minimum-cost dynamic flows : The series-parallel case. *Networks 43*(3), 153–162.

Kondor, D., M. Pósfai, I. Csabai, et G. Vattay (2014). Do the rich get richer ? an empirical analysis of the bitcoin transaction network. *PloS one 9*(2), e86197.

Koster, A. et X. Muñoz (2009). *Graphs and algorithms in communication networks : studies in broadband, optical, wireless and ad hoc networks*. Springer Science & Business Media.

Kovanen, L., M. Karsai, K. Kaski, J. Kertész, et J. Saramäki (2011). Temporal motifs in time-dependent networks. *Journal of Statistical Mechanics : Theory and Experiment 2011*(11), P11005.

Kuhn, F., N. Lynch, et R. Oshman (2010). Distributed computation in dynamic networks. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pp. 513–522.

Lamprou, I., R. Martin, et P. Spirakis (2018). Cover time in edge-uniform stochastically-evolving graphs. *Algorithms 11*(10), 149.

Latapy, M., T. Viard, et C. Magnien (2017). Stream graphs and link streams for the modeling of interactions over time.

Li, M.-X., V. Palchykov, Z.-Q. Jiang, K. Kaski, J. Kertész, S. Miccichè, M. Tumminello, W.-X. Zhou, et R. N. Mantegna (2014). Statistically validated mobile communication networks : the evolution of motifs in european and chinese data. *New Journal of Physics 16*(8), 083038.

Lin, M. et P. Jaillet (2015). On the quickest flow problem in dynamic networks : a parametric min-cost flow approach. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1343–1356. Society for Industrial and Applied Mathematics.

Luo, J. et L. Kuang (2014). A new method for predicting essential proteins based on dynamic network topology and complex information. *Computational biology and chemistry 52*, 34–42.

Michail, O. (2016). An introduction to temporal graphs : An algorithmic perspective. *Internet Mathematics 12*(4), 239–280.

Michail, O. et P. G. Spirakis (2016). Traveling salesman problems in temporal graphs. *Theoretical Computer Science 634*, 1–23.

Miller-Hooks, E. et S. S. Patterson (2004). On solving quickest time problems in time-dependent, dynamic networks. *Journal of Mathematical Modelling and Algorithms 3*(1), 39–71.

Minieka, E. (1973). Maximal, lexicographic, and dynamic network flows. *Operations Research 21*(2), 517–527.

Minieka, E. (1974). Dynamic network flows with arc changes. *Networks 4*(3), 255–265.

Moal, F. (2020, Septembre). Étude des ensembles de steiner de cardinalité k. Stage de master, Université Le Havre Normandie. Encadré par Éric Sanlaville.

Nannicini, G., P. Baptiste, G. Barbier, D. Krob, et L. Liberti (2010). Fast paths in large-scale dynamic road networks. *Computational Optimization and Applications 45*(1), 143–158.

Nasrabadi, E. et S. M. Hashemi (2010). Minimum cost time-varying network flow problems. *Optimization Methods & Software 25*(3), 429–447.

Neggaz, M. Y. (2016, October). *Automatic classification of dynamic graphs*. Theses, Université de Bordeaux.

Newman, M. E. (2003). The structure and function of complex networks. *SIAM review 45*(2), 167–256.

Nguyen, N. P., T. N. Dinh, Y. Xuan, et M. T. Thai (2011). Adaptive algorithms for detecting community structure in dynamic social networks. In *2011 Proceedings IEEE INFOCOM*, pp. 2282–2290.

Nicosia, V., J. Tang, M. Musolesi, G. Russo, C. Mascolo, et V. Latora (2012). Components in time-varying graphs. *Chaos : An interdisciplinary journal of nonlinear science 22*(2).

Oberoi, K. S. (2019, November). *Spatio-temporal modeling of urban road traffic*. Theses, Normandie Université.

Orda, A. et R. Rom (1995). On continuous network flows. *Operations Research Letters 17*(1), 27–36.

Orlin, J. B. (1984). Minimum convex cost dynamic network flows. *Mathematics of Operations Research 9*(2), 190–207.

Parpalea, M. et E. Ciurea (2011). The quickest maximum dynamic flow of minimum cost. *International Journal of Applied Mathematics and Informatics 3*(5), 266–274.

Rostami, R. et A. Ebrahimnejad (2014). An approximation algorithm for discrete minimum cost flows over time problem. *International Journal of Operational Research 20*(2), 226–239.

Rosvall, M., A. V. Esquivel, A. Lancichinetti, J. D. West, et R. Lambiotte (2014). Memory in network flows and its effects on spreading dynamics and community detection. *Nature communications 5*(1), 1–13.

Rozenshtein, P., A. Gionis, B. A. Prakash, et J. Vreeken (2016). Reconstructing an epidemic over time. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1835–1844.

Scholtes, I., N. Wider, R. Pfitzner, A. Garas, C. J. Tessone, et F. Schweitzer (2014). Causality-driven slow-down and speed-up of diffusion in non-markovian temporal networks. *Nature communications 5*(1), 1–9.

Skutella, M. (2009). An introduction to network flows over time. In *Research Trends in Combinatorial Optimization*, pp. 451–482. Springer.

Taylor, I. W., R. Linding, D. Warde-Farley, Y. Liu, C. Pesquita, D. Faria, S. Bull, T. Pawson, Q. Morris, et J. L. Wrana (2009). Dynamic modularity in protein interaction networks predicts breast cancer outcome. *Nature biotechnology 27*(2), 199–204.

Tirico, M., S. Balev, A. Dutot, et D. Olivier (2018). Morphogenesis of complex networks : A reaction diffusion framework for spatial graphs. In *International Conference on Complex Networks and their Applications*, pp. 769–781. Springer.

Watts, D. J. et S. H. Strogatz (1998). Collective dynamics of 'small-world'networks. *Nature 393*(6684), 440–442.

Weron, R. (2014). Electricity price forecasting : A review of the state-of-the-art with a look into the future. *International Journal of Forecasting 30*(4), 1030–1081.

White, K., M. Farber, et W. Pulleyblank (1985). Steiner trees, connected domination and strongly chordal graphs. *Networks 15*(1), 109–124.

Wilkinson, W. L. (1971). An algorithm for universal maximal dynamic flows in a network. *Operations Research 19*(7), 1602–1612.

Zhao, B., W. Wang, G. Xue, N. Yuan, et Q. Tian (2015). An empirical analysis on temporal pattern of credit card trade. In *International Conference in Swarm Intelligence*, pp. 63–70. Springer.

Publications

Balev, S., Y. Pigné, E. Sanlaville, et M. Vernet (2020). On the complexity of the Dynamic Steiner Tree Problem. Rapport interne.

Vernet, M., M. Drozdowski, Y. Pigné, et E. Sanlaville (2020). A theoretical and experimental study of a new algorithm for minimum cost flow in dynamic graphs. *Discrete Applied Mathematics*.

Vernet, M., Y. Pigné, et E. Sanlaville (2020, February). A Study of Connectivity on Dynamic Graphs : Computing Persistent Connected Components. Article soumis.

Résumé Les problèmes de graphes ont été largement étudiés dans le cas des graphes statiques. Cependant, ces graphes ne permettent pas de prendre en compte la dimension temporelle, qui est souvent une donnée importante pour les situations à modéliser. Les graphes dynamiques viennent combler ces lacunes en permettant de modéliser des évolutions dans le temps. On peut alors s'interroger sur ces mêmes problèmes de graphes dans un contexte dynamique. Cela passe d'abord par la définition du modèle de graphes dynamiques le plus approprié et la modélisation précise du problème sur ces graphes. Lorsque le problème ne peut pas être résolu efficacement en appliquant directement des méthodes connues sur les graphes statiques, il faut alors concevoir un algorithme de résolution spécifique aux graphes dynamiques et l'analyser théoriquement et expérimentalement.

En suivant cette démarche, l'objectif de cette thèse est de s'interroger sur l'extension aux graphes dynamiques des problèmes bien connus sur les graphes statiques. Ce travail s'intéresse à plusieurs problèmes de graphes en contexte dynamique en se focalisant sur les aspects algorithmiques et en s'abstrayant des domaines d'applications.

Nous nous intéressons d'abord aux problèmes de flot dans les graphes dynamiques et proposons en particulier pour le problème du flot de cout minimum un algorithme polynomial permettant de résoudre le problème de façon optimale pour un modèle de graphe dynamique spécifique. Des problèmes liés à la connexité des graphes dynamiques sont aussi étudiés. Les composantes connexes persistantes, extension des composantes connexes aux graphes dynamiques, traduisent la connexité d'un ensemble de nœuds pendant un certain nombre de pas de temps consécutifs. De façon analogue à la notion de maximalité des composantes connexes dans un graphe statique, une notion de dominance entre composantes connexes persistantes est définie. Un algorithme polynomial permettant d'identifier toutes les composantes connexes persistantes non dominées est proposé. Plusieurs extensions à la définition de composantes connexes persistantes sont étudiées. Nous proposons enfin des extensions possibles du problème de Steiner aux graphes dynamiques. Nous nous concentrons sur un cas particulier et montrons la NP-complétude de ce problème.

Mots clé graphe, algorithmique, graphe dynamique, algorithme exact, modélisation, problème de flot, connexité, problème de Steiner, complexité

Abstract Graph problems have been widely studied in the case of static graphs. However, these graphs do not allow a time dimension to be considered, even though time is an important variable for the situations to model. Dynamic graphs make it possible to model evolution over time. This is a reason to wonder about graph problems in a dynamic context. First, it is necessary to define the most appropriate dynamic graphs model and the precise problem on those graphs. When the problem cannot be efficiently solved directly using known static graph methods, an algorithm specific to dynamic graphs must be designed and analyzed theoretically and practically.

With that approach, this thesis' objective is to study graph problems' extensions to dynamic graphs. This works deals with several graph problems in a dynamic context by focusing on algorithmic aspects and without considering application domains.

Flow problems in dynamic graphs are first studied and we propose a polynomial algorithm to optimally solve the minimum cost flow problem for one dynamic graph model. Connectivity problems are then studied. Persistent connected components, an extension of connected components to dynamic graphs, are a set of vertices remaining connected for consecutive time steps. Analogously to the maximality concept of connected components in static graphs, a dominance concept between persistent connected components is defined. A polynomial algorithm to identify all non dominated connected components in a dynamic graph is designed. Several extensions to the definition of persistent connected components are studied. Finally, we propose several possible extensions of the Steiner problem to dynamic graphs. We focus on a special case and show the NP-completeness of the problem.

Key words graph, algorithmic, dynamic graph, exact algorithm, model, flow problem, connectivity, Steiner problem, complexity