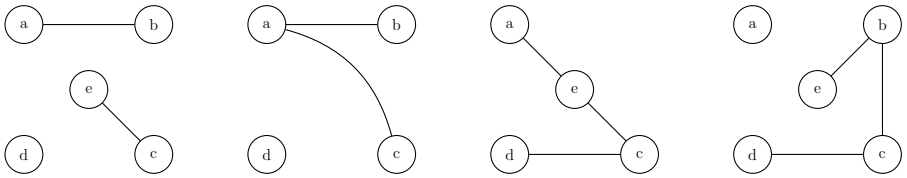
Graphes Temporels ou Dynamiques

Antonin Décimo
Encadrants : Michel Habib, Laurent Viennot

22 juin 2018

1 Introduction

Un graphe dynamique est un graphe qui évolue au cours du temps, c'est-à-dire que des nœuds ou des arêtes peuvent apparaître ou disparaître au cours du temps. On peut s'en servir pour modéliser des réseaux informatiques, comme du calcul distribué ou de la téléphonie mobile, ou bien encore des réseaux de transports routiers ou publics.



(a) Temps 0 (b) Temps 1 (c) Temps 2 (d) Temps 3

FIG. 1 : Graphe dynamique

Un graphe dynamique peut être représenté comme dans la figure 1, où le temps est continu mais les événements sont discrets. Un graphe dynamique peut être vu comme une suite de graphes statiques, chacun défini par un unique événement instantané ou un ensemble d'événements discrets.

Dans ce rapport, on rappelle quelques notions sur les graphes usuels, puis on introduit les graphes dynamiques en comparant des modèles de la littérature. On peut ensuite transposer les notions (chemins, …) et les structures (sous-graphes, …) des graphes usuels aux graphes dynamiques. On s'intéresse à un algorithme de calcul de l'arbre des plus courts chemins dans un graphe dynamique, puis à deux articles de recherche.

1.1 Rappels

Graphe Les graphes (*graphs*) sont des modèles abstraits constitués par la donnée de points, appelés nœuds (*nodes*) ou sommets (*vertices*), et de liens entre ces points, appelés arêtes (*edges*). Les arêtes comme les nœuds peuvent être étiquetées, le label représentant souvent le coût pour traverser l'élément. Un graphe est noté $G = (V, E)$, où V est l'ensemble des sommets et E l'ensemble des arêtes.

Non-orienté, orienté Un graphe $G = (V, E)$ est non-orienté si E est un ensemble de paires de sommets, orienté si E est un ensemble de couples de sommets.

Chaîne, chemin Une chaîne est une suite finie d'arêtes consécutives dans un graphe non-orienté. Un chemin est l'équivalent dans un graphe orienté.

Degré Le degré d'un sommet d'un graphe est le nombre de liens reliant ce sommet, noté $\deg : V \to \mathbb{N}$. Dans le cas d'un graphe orienté, on parle de degré entrant, noté $\deg^-$, et de degré sortant, noté $\deg^+$.

Connexité (*connectivity*) Un graphe non-orienté est connexe si pour tous $u, v \in V$ il existe une chaîne de u à v. Dans un graphe orienté, on parle de connexité forte s'il existe un chemin pour tous $u, v \in V$.

Graphe k-régulier (k-*regular*) Un graphe est k-régulier si tous ses sommets sont de degré k.

k-sommet-connexe (k-*vertex-connected*) Un graphe est k-sommet-connexe s'il possède plus de k sommets et s'il reste connexe après en avoir ôté strictement moins de k.

Graphe expanseur (*expander graph*) Le taux d'expansion d'un graphe est une mesure de sa connexité. Soit $G = (V, E)$ un graphe, et W un sous-graphe de G. La **frontière extérieure des sommets** (*outer vertex boundary*) est l'ensemble des sommets de $V \setminus W$ ayant au moins un voisin dans W.

$$\partial(W) = \{v \in V \setminus W \mid \exists w \in W, (w, v) \in E\}$$

On définit alors le **taux d'expansion des sommets**.

$$h(G) = \min_{0<|W|\leq n/2} \frac{|\partial(W)|}{|W|}$$

G est un graphe c-expanseur si pour tout $W \subset V$ tel que $|W| \leq n/2$, alors $|\partial(W)| \geq c|W|$.

2 Modèles de graphes dynamiques

Les graphes dynamiques (*dynamic graphs*) sont aussi appelés graphes temporels (*temporal graphs*), graphes de flux (*stream graphs*), réseaux dynamiques (*dynamic networks*), graphes évolutifs (*evolving graphs*), graphes variants avec le temps (*time-varying graphs*).

2.1 Modèles sans latence

Graphe de flux [6] (*stream graph*) Un graphe de flux $S = (T, V, W, E)$ est un ensemble fini de nœuds V, un ensemble d'instants T, un ensemble de nœuds temporels $W \subseteq T \times V$ (un ensemble de nœuds indexés par un temps), et un ensemble d'arêtes $E \subseteq T \times (V \otimes V)$ tel que si $(t, uv) \in E$ alors $(t, u) \in W$ et $(t, v) \in W$ (si une arête existe à un temps t, alors les deux nœuds de l'arête existent aussi à ce temps).

Flux de liens [6] (*link stream*) Si tous les nœuds du graphe de flux sont présents à chaque instant, on parle de flux de liens.

Graphe évoluant [5] (*evolving graph*) Il s'agit d'un graphe dynamique $G = (V, E)$, où V est un ensemble de nœuds, et $E : \mathbb{N}^+ \to \mathcal{P}(V \times V)$ est une fonction attribuant à chaque tour $r \in \mathbb{N}^+$ un ensemble d'arêtes $E(r)$ pour ce tour. Ici l'ensemble des nœuds ne varie pas.

Il n'y a pas de différence fondamentale entre les *link streams* et les *evolving graphs*, mais notons que les *stream graphs* et les *link streams* se veulent plus généraux en acceptant une définition plus large de la temporalité (continue comme discrète), alors que les *evolving graphs* sont discrets.

2.2 Modèles avec latence

Time-Varying Graphs [2] (TVG) La présence des entités est établie par intervalle de temps (discret ou continu). On note $\mathbb{T}$ le domaine temporel. Les auteurs proposent de rajouter un label L sur chaque arête, et préfèrent indiquer la présence d'arêtes et de nœuds par des fonctions. On a donc $G = (V, E \subseteq L \times V \otimes V, T \subseteq \mathbb{T}, \rho, \zeta)$, où :

- $\rho : E \times T \to \{\top, \bot\}$ est appelée fonction de présence et indique si une arête donnée est disponible à un temps donné.
- $\zeta : E \times T \to \mathbb{T}$ est appelée fonction de latence et indique le temps de parcours d'une arête donnée pour un départ à un temps donné (la latence d'une arête pouvant changer au cours du temps).

On peut à l'instar des arêtes étendre ce modèle aux nœuds du graphe, avec ψ la fonction de présence et φ la fonction de latence. En choisissant une fonction de latence nulle, on se ramène à un modèle sans latence.

3 Notions et Propriétés

L'article [6] a pour but d'étendre les notions des graphes usuels et des séries temporelles en les combinant pour donner forme aux graphes de flux (*stream graphs*), $S = (T, V, W, E)$. On retiendra quelques-unes de ces notions.

Couverture (*coverage*) La couverture de S est la proportion de nœuds présents sur toutes les possibilités de présence de nœuds.

$$cov(S) = \frac{|W|}{|T \times V|}$$

Nombre de nœuds, de liens Chaque nœud (reps. lien) contribue proportionnellement à $n = \frac{|W|}{|T|}$ (resp. $m = \frac{|E|}{|T|}$) par son temps de présence dans S : $v \in V$ vaut 1 s'il est toujours présent dans S, $(u, v) \in V \times V$ vaut 1 si elle est toujours présente dans S.

Voisinage et degré Dans un graphe usuel $G = (V, E)$, le voisinage $N(v)$ de $v \in V$ est $N(v) = \{u \,|\, (u, v) \in E\}$, et $\deg(v) = |N(v)|$. Dans un graphe dynamique, le voisinage d'un nœud est défini par $N(v) = \{(t, u) \,|\, (t, uv) \in E\}$. La définition du degré est identique. On peut restreindre la définition à un intervalle de temps $N_{[t_1,t_2]}(v) = \{u \,|\, (t, uv) \in E, t \in [t_1, t_2]\}$.

Uniformité Notons T_u l'ensemble des instants où u est présent. Si pour deux nœuds u et v, $|T_u| = |T_v|$, alors on peut avoir $T_u = T_v$, ou $T_u \cap T_v = \emptyset$, ou n'importe quelle autre situation. Pour capturer l'existence possible de liens entre u et v, on définit l'uniformité du graphe :

$$\mathbb{U}(S) = \frac{\sum_{uv \in V \times V} |T_u \cap T_v|}{\sum_{uv \in V \times V} |T_u \cup T_v|}$$

Si $\mathbb{U}(S) = 1$, on dit que S est uniforme. Tous les nœuds sont présents aux mêmes instants.

Compacité Soit $S = (T, V, W, E)$ un graphe de flux, définissons $S' = (T', V', W, E)$ tel que $T' = [\min\{t \mid \exists (t, v) \in W\}, \max\{t \mid \exists (t, v) \in W\}]$ (on restreint le graphe de la première apparition à la dernière apparition de nœud), et $V' = \{v \mid \exists (t, v) \in W\}$ (on restreint le graphe aux nœuds qui apparaissent au moins une fois). La compacité est définie comme suit :

$$c(S) = \frac{|W|}{|T' \times V'|} = cov(S')$$

Si $c(S) = 1$, on dit que S est compact : les temps de présence de tous les nœuds sont le même intervalle de T.

L'uniformité et la compacité d'un graphe de flux sont nécessairement 1.

L'article [4] présente deux notions intéressantes.

Connexité T-intervalle Un graphe dynamique $G = (V, E)$ est T-intervalle connexe (T-*interval connected*) pour $T \geq 1$ si pour tout $r \in \mathbb{N}$, le graphe statique $G_{r,T} := \left(V, \bigcap_{i=r}^{r+T-1} E(r)\right)$ est connexe. Le graphe est ∞-intervalle connexe s'il existe un graphe statique connexe $G' = (V, E')$ tel que pour tous $r \in \mathbb{N}, E' \subseteq E(r)$

Causalité de Lamport Soit un graphe dynamique $G = (V, E, T)$, on peut définir un ordre $\rightarrow$ sur (V, T), tel que $(u, r) \rightarrow (v, r')$ ssi $(r' = r + 1)$ et $\{u, v\} \in E(r)$. L'**ordre causal** $\rightsquigarrow$ est défini comme clôture réflexive et transitive de $\rightarrow$.

3.1 Voyages et distances

Les articles [1], [2], et [6] définissent les concepts de chemins, distances, et voyages dans les graphes dynamiques. Nous allons considérer le modèle de [2] ($G = (V, E, T, \rho, \zeta)$), plus complet.

Voyages, chemins (*journeys, paths*) Un voyage est une séquence $\mathcal{J} = \{(e_1, t_1), \ldots, (e_k, t_k)\}$ telle que $e_1, \ldots, e_k$ est une marche dans le graphe, $\rho(e_i, t_i) = \top$, et $t_{i+1} \geq t_i + \zeta(e_i, t_i)$ pour tout $i < k$. Selon l'application, on pourra supposer $\rho_{[t_i, t_i + \zeta(e_i, t_i)]}(e_i) = \top$, c'est-à-dire que l'arête reste présente tandis qu'elle est traversée.

On note $arrival(\mathcal{J})$ et $departure(\mathcal{J})$ les temps de départ et d'arrivée du chemin $\mathcal{J}$. $\mathcal{J}_{(u,v)}$ est un chemin de u à v, $\mathcal{J}^*_{(u,v)}$ est l'ensemble des chemins de u à v.

Les auteurs ne proposent pas de définition tenant compte de la latence des nœuds. Rappelons que si une arête est présente, alors chacun des deux nœuds la constituant est aussi présent.

Dans les graphes dynamiques, les chemins sont asymétriques. Il est impossible de revenir en arrière dans le temps. Il est parfois possible de revenir sur un nœud à une date ultérieure.

Distance topologique La distance topologique entre deux nœuds dans un graphe est la longueur minimum (en nombre d'arêtes) des chemins entre ces nœuds. On parle de **chemin le plus court** (*shortest path*).

Distance temporelle On dit qu'un chemin est **le plus rapide** (*fastest path*) s'il est de durée (la somme des latences des nœuds et des arêtes rencontrés) minimum.
Le temps pour atteindre (t, v) depuis u au temps α est $\mathcal{T}_\alpha(u, (t, v)) = \omega - \alpha$, où ω est la plus petite valeur pour laquelle il existe un chemin de (α, u) à (ω, v). Un tel chemin est appelé **premier chemin** (*foremost path*).

3.2 Structures

En généralisant le langage habituel des graphes, on peut définir les **sous-graphes dynamiques**, soit en considérant les sous-graphes induits par des sous-ensembles de nœuds ou d'arêtes, soit en se restreignant à un intervalle temporel.

Pour un graphe dynamique $\mathcal{G} = (V, E, T, \rho, \zeta)$, on appelle **graphe sous-jacent** $G = (V, E)$ le graphe statique dans lequel on oublie la dimension temporelle, et qui indique seulement les nœuds en relation à un certain temps dans T.

3.3 Graphes dynamiques aléatoires

Dans [2] et [5], les auteurs construisent des graphes dynamiques aléatoires en posant que la fonction de présence d'une arête $\rho : E \times T \to 0, 1$ suit une loi de probabilité. On peut imaginer des graphes dynamiques suivant des chaînes de Markov, où une arête apparaît au tour suivant avec une probabilité p et disparaît avec une probabilité q, indépendamment des autres arêtes. Si $q = 1 - p$, chaque graphe statique est un graphe d'Erdős-Rényi.

4 Algorithmes

4.1 Flot maximum

Flot On considère un graphe orienté $G = (V, E)$ et deux sommets $s, t \in V$. Les arcs sont munis de capacités $c : E \to \mathbb{R}^+$. Un flot sur G est un vecteur ϕ indexé par E qui vérifie les lois de Kirchhoff :

- $\forall (u, v) \in E, 0 \leq \phi(u, v) \leq c(u, v)$
- $\forall v \in V, \sum_{(u,v)\in E} \phi(u, v) = \sum_{(v,w)\in E} \phi(v, w)$

On cherche un flot qui maximise $\phi(t, s)$.

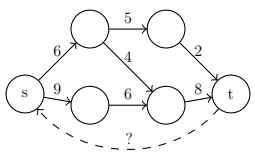


FIG. 2 : Flot

Les algorithmes partent du flot nul, et utilisent une suite de chemins augmentant. Entre deux étapes, le graphe change. On peut donc voir les algorithmes de flot maximum comme les algorithmes de graphes dynamiques.

4.2 Diamètre dynamique (*dynamic diameter*, *flooding time*)

Soit $G = (V, E)$ un graphe dynamique et $u \in V$. Si $\forall v \in V, (u, t) \rightsquigarrow (v, t' + D)$, alors D est le diamètre dynamique du graphe. Il faut comprendre que si u envoie un message au temps t, alors le message mettra au maximum D tours pour atteindre tous les nœuds du graphe. Le problème est de déterminer le diamètre dynamique du graphe. On peut établir une notion de diamètre dynamique pour chacune des trois distances sur les graphes dynamiques.

4.3 Autres

Plusieurs autres problèmes sont décrits dans [4]. Dans cette situation, les algorithmes sont distribués : chaque nœud exécute l'algorithme. Les arêtes servent à communiquer entre les nœuds.

Comptage Quand l'algorithme est exécuté sur un graphe à n nœuds, chaque nœud termine l'exécution en produisant n.

k-vérification Chaque nœud débute avec un entier k et doit déterminer si $n \leq k$. Les nœuds répondent "oui" ssi il y a au plus k nœuds dans le graphe.

Dissémination k-jeton k jetons d'information sont disséminés dans le réseau. Chaque nœud doit finir par connaître les k jetons.

4.4 Plus court chemin (*shortest path*)

Modèle On considère le problème de calculer les chemins les plus courts en nombre de sauts (*hop-count*) d'un nœud u pour chacun des autres nœuds du graphe. Dans un graphe usuel, cela revient à exécuter le parcours en largeur (*BFS*), ou l'algorithme de Dijkstra si les arêtes sont valuées à 1.

L'algorithme 1, présenté dans [1], calcule les plus courts chemins dans un graphe dynamique. Les auteurs font plusieurs suppositions :

- On ne peut pas s'engager dans une arête si elle disparaît avant de l'avoir traversée. Si l'arête e existe sur $[t_1, t_2]$, et qu'on s'engage dans l'arête au temps $t \in [t_1, t_2]$, alors $t + traversal(e) \leq t_2$.
- Si l'arête $e = (u, v)$ existe sur $[t_1, t_2]$, alors u et v existent sur $[t_1, t_2]$.
- Il n'existe qu'une seule arête entre deux nœuds (pas de multigraphe).
- La latence d'une arête ne varie pas au cours du temps.

Préfixe L'algorithme usuel est basé sur l'observation qu'un préfixe d'un plus court chemin est un plus court chemin lui-même. Malheureusement, dans les graphes dynamiques cette propriété n'est plus vraie. Dans le graphe de la figure 3, le plus court chemin de s à t est bien $\mathcal{J} = s \to a \to b \to t$, mais le préfixe $\mathcal{J}''' = s \to a \to b$ de $\mathcal{J}$ n'est pas le plus court chemin de s à b, c'est $s \to b$, disponible au temps 1.

(a) Temps 0 (b) Temps 1

FIG. 3 : Le préfixe d'un plus court chemin n'est pas le plus court chemin

Les auteurs de [1] proposent une autre propriété de préfixe.

Soit un graphe dynamique $G = (V, E)$ et un plus court chemin $\mathcal{J}_{(u,w)}$ dans G. Si la dernière arête (v, w) de $\mathcal{J}_{(u,w)}$ débute au temps t, alors le chemin préfixe $\mathcal{J}'_{(u,v)}$ de $\mathcal{J}_{(u,w)}$ est plus court que tous les autres chemins de u à v finissant avant t.

Condition d'arrêt L'algorithme présentait comme condition de boucle principale :

`tant qu'il existe` $w \in V_G$ `tel que` $location(w)$ `n'est pas définie`.

Cette condition n'est pas très pertinente ; en effet si un nœud n'est pas atteignable depuis $(s, 0)$, alors ce nœud ne sera jamais marqué par *location* et l'algorithme ne termine pas. Il existe une meilleure condition d'arrêt : si aucun nœud n'a été inséré à la profondeur d, alors il n'existe pas de plus court chemin de longueur au moins d et l'algorithme se termine.

Neighbours Pour un nœud u présent au temps t, la fonction $neighbours : V \times \mathbb{T} \to \mathcal{P}(V \times \mathbb{T})$ renvoie (v, t') si v est un voisin accessible de u au temps le plus tôt $t' \geq t$.

Input : Un graphe dynamique G, un nœud $s \in V_G$
Data : L'arbre T des paires $(u, t) \in V_G \times \mathbb{R}_+^*$; un entier d ; un tableau $earliest : V_G \to \mathbb{R}_+^*$
Result : L'arbre des plus courts chemins T, un tableau $location : V_G \to T$

$T \leftarrow \{(s, 0)\}$
$earliest(s) \leftarrow 0$
$\forall u \neq s, earliest(u) \leftarrow \infty$
$d \leftarrow 1$
$location(s) \leftarrow (s, 0)$
$run \leftarrow \top$
while $run = \top$ **do**
  $run \leftarrow \bot$
  foreach $(u, t) \in T$ *à la profondeur* d **do**
    foreach $(v, t') \in neighbours(u, t)$ **do**
      if $location(v)$ *n'est pas définie* **then**
        $location(v) \leftarrow (v, t')$
      end
      if $t' < earliest(v)$ **then**
        $earliest(v) \leftarrow t'$
        $parent((v, t')) \leftarrow (u, t)$ dans T
        $run \leftarrow \top$
      end
    end
  end
  $d \leftarrow d + 1$
end

Algorithm 1 : Arbre des plus courts chemins

Une fois que l'arbre a été calculé, retrouver le chemin de s à n'importe quel sommet u est aisé : $location(u)$ donne la paire (u, t) correspondante dans l'arbre. On retrouve le chemin en remontant de parent en parent dans l'arbre T, depuis (u, t) jusqu'à $(s, 0)$.

Preuve pour les *Link streams* Supposons que le graphe est un *link stream*, c'est-à-dire que l'ensemble des nœuds est figé. Tous les nœuds sont présents pendant toute l'existence du graphe, et seules les arêtes changent.

Soit $G = (V, E)$ un tel graphe. Cherchons les plus courts chemins depuis $(s, 0), s \in V$.

Initialisation T est initialisé à $(s, 0)$. Si le graphe est réduit à s, T est bien un arbres de plus courts chemins. $location(v)$ pointe vers le plus court chemin $s \xrightarrow{0} s$.

Récurrence Soit (u, t_u) un sommet accessible depuis $(s, 0)$ en d sauts. (u, t_u) est dans l'arbre T à la profondeur d. Soit (v, t_v) un voisin de (u, t_u) accessible au plus tôt en t_v. $(s, 0) \xrightarrow{d} (u, t_u)$ est le plus court de tous les chemins de s à u finissant avant t_v.

Supposons que $(s, 0) \xrightarrow{d} (u, t_u) \to (v, t_v)$ soit un plus court chemin. Il y a alors deux possibilités.

- C'est la première fois que l'on rencontre v à la profondeur $d + 1$. $location(v)$ n'est pas définie, car sinon $(v, \_)$ serait déjà dans l'arbre à une profondeur $< d$. On définit donc $location(v) := (v, t_v)$. De même, $earliest(v) = \infty$. Alors $earliest(v) := t_v$, et (v, t_v) est ajouté dans l'arbre des plus courts chemins avec comme parent (u, t_u).
- On a déjà rencontré v à une profondeur $\leq d + 1$. $location(v)$ est donc définie. Rappelons que les nœuds sont toujours présents, donc $\forall u \in V, \forall t' > t \in T, neighbours(u, t') \subseteq neighbours(u, t)$. Si $t_v < earliest(v)$, alors (v, t_v) capture plus de voisins que $(v, earliest(v))$ et doit être ajouté à l'arbre des plus courts chemins avec comme parent (u, t_u). On met à jour $earliest(v) := t_v$. Il n'y a pas besoin de mettre à jour $location(v)$ qui pointe déjà vers un plus court chemin $s \xrightarrow{d+1} v$. Si $t_v \geq earliest(v)$ alors $(v, earliest(v))$ capture au moins autant de voisins que (v, t_v), et (v, t_v) n'est pas ajouté à T.

À la fin de cette boucle, $(v, earliest(v))$ est présent dans l'arbre des plus courts chemins à la profondeur $d + 1$ avec comme parent (u, t_u), $location(v)$ pointe vers un plus court chemin $s \xrightarrow{d+1} v$, et $earliest(v)$ est le temps le plus tôt auquel on puisse accéder à v. L'invariant est conservé.

Supposons que $\mathcal{J} = (s, 0) \xrightarrow{d} (u, t_u) \to (v, t_v)$ ne soit pas un plus court chemin $s \xrightarrow{d+1} v$. Par conséquent, il existe un plus court chemin $\mathcal{J}' = s \xrightarrow{d'} v$ avec $d' \leq d$. Par hypothèse, $\mathcal{J}'$ est déjà dans T. On veut vérifier que $\mathcal{J}$ n'est pas ajouté comme plus court chemin de s à v dans l'arbre. Deux cas se présentent :

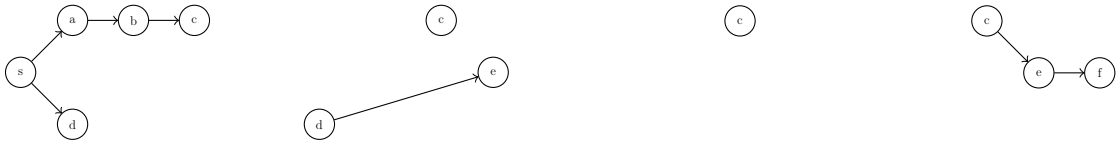
- Si $t_v < earliest(v)$, alors $neighbours(v, earliest(v)) \subseteq neighbours(v, t_v)$. Il faut donc ajouter (v, t_v) à T avec comme parent (u, t_u), mais sans mettre à jour $location(v)$. Ainsi, $location(v)$ continuera de pointer vers un plus court chemin $s \xrightarrow{d'} v$. S'il existe un voisin (w, t) de (v, t_v) accessible seulement sur $[t_v, earliest(v)[$, alors l'invariant est vérifié pour ce voisin. En effet le chemin $(s, 0) \xrightarrow{d+1} (v, t_v)$ est le plus court des chemins de s à v se terminant avant t. On met à jour $earliest(v) := t_v$.
- Si $t_v \geq earliest(v)$, alors $neighbours(v, t_v) \subseteq neighbours(v, earliest(v))$. Il n'y a donc rien à faire de (v, t_v).

À la fin de cette boucle, l'invariant est correct pour tous les nœuds de l'arbre.

La terminaison de l'algorithme est évidente, l'algorithme est donc correct sur les *link streams*.

Preuve pour les *stream graphs* Dans le cas d'un *stream graph* (on attribue une fonction de présence aux nœuds), l'algorithme n'est plus correct. Observons que *neighbours* doit vérifier la propriété suivante : si u est présent sur $[t_1, t_2]$ puis $[t_3, t_4]$, et soit $t \in [t_1, t_2]$, alors $neighbours(u, t)$ ne renvoie que les voisins de u accessibles en $[t_1, t_2]$ et pas $[t_3, t_4]$. Si *neighbours* ne vérifie pas cette propriété, alors on pourrait stationner sur u à partir de t_1 pour accéder aux voisins disponibles en $[t_3, t_4]$, bien que u disparaisse entre t_2 et t_3.

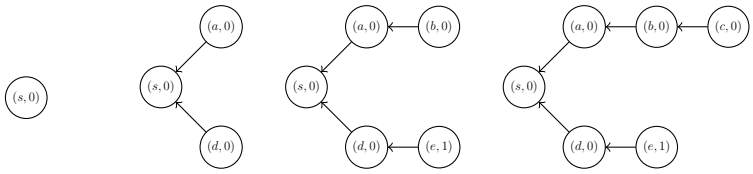
Considérons le graphe de la figure 4, où la latence des arêtes est 0. Il y a 4 étapes. L'algorithme ne pourra pas découvrir le nœud f, pourtant accessible au temps 3 par $s \to a \to b \to c \to e \to f$, en stationnant sur c aux temps 1 et 2.



(a) G_0 (b) G_1 (c) G_2 (d) G_3

FIG. 4 : Graphe dynamique $G = (G_0, G_1, G_2, G_3)$

Le problème se pose lors de la construction de l'arbre de la figure 5 des plus courts chemins du graphe de la figure 4. Dans T_2, on découvre pour la première fois e que l'on marque $location(e) = (e, 1)$ et $earliest(e) = 1$. Dans T_3, en examinant $(c, 0)$ on va découvrir le nœud $(e, 3)$. Or $earliest(e) = 1 < 3$, donc on n'entre pas dans le deuxième `if` et $(e, 3)$ n'est pas introduit dans l'arbre. Par conséquent, f n'est jamais découvert.



(a) T_0 (b) T_1 (c) T_2 (d) T_3

FIG. 5 : Arbre des plus courts chemins de G

L'exemple donné vérifie bien les conditions de la propriété du préfixe ; $\mathcal{J}_1 = s \to a \to b \to c \to e \to f$ est bien le plus court chemin de s à f, mais le chemin préfixe $\mathcal{J}_1' = s \to a \to b \to c \to e$ n'est pas le plus court chemin de s à e, c'est $\mathcal{J}_2 = s \to d \to e$. Notons que $e \to f$ démarre au temps $t = 3$, et $\mathcal{J}_2$ finit en 1, donc avant t.

Implémentation Une implémentation de cet algorithme réalisée en OCaml est disponible à l'adresse `https://github.com/MisterDA/tre-dynamic-graphs`. Les contraintes de l'algorithme ont été conservées.

Le graphe est présenté sous forme de liste d'adjacence (listing 1). Chaque nœud contient ses temps de présence, la liste de ses voisins et les temps où chacun d'entre eux est accessible, ainsi que la latence de l'arc. Les nœuds sont identifiés de manière unique par leur nom. On suppose que les données sont valides (les intervalles de temps ne se recoupent pas, …) et triées.

Pour le graphe de la figure 1, $shortest\_path(a, d)$ donne :

```
a 0.000000 -> c 1.000000 -> d 2.000000
```

et pour le graphe de la figure 4, $shortest\_path(s, f)$ donne (montrant que f n'est pas découvert) :

```
Fatal error : exception Failure("location(f) was not set")
```

```
type time = float
type time_interval = time * time (* [start, finish] *)

type node = {
    name   : string;
    node_schedule   : time_interval list;
    neighbours   : neighbour list;
  }
and neighbour = {
    node   : node;
    arc_schedule   : time_interval list;
    traversal   : time;
  }
type graph = node list
```

Listing 1 : Structure de données de graphe dynamique

Complexité La profondeur maximum de l'arbre est d, le diamètre en sauts (*hop diameter*) du graphe. L'arbre contient au maximum nd paires (n à chaque profondeur). Vérifier les voisins d'un sommet u prend un temps proportionnel au degré de u, donc vérifier les voisins de tous les sommets d'une certaine profondeur prend au maximum $2m$ étapes. L'algorithme a donc une complexité en $O(md)$.

Optimisation de l'algorithme On a vu que dans le cas des *link streams*, $\forall u \in V, \forall t' > t \in T, neighbours(u,t') \subseteq neighbours(u,t)$. Si on observe d'abord les voisins de (u,t'), puis qu'on découvre (u,t), il est inutile d'observer deux fois $neighbours(u,t) \setminus neighbours(u,t')$.

Révision de l'algorithme Une nouvelle propriété de préfixe doit être définie. Je conjecture que la suivante aura le bon comportement.

Soit un graphe dynamique $G = (V,E)$ et un plus court chemin $\mathcal{J}_{(u,w)}$ dans G. Si la dernière arête (v,w) de $\mathcal{J}_{(u,w)}$ débute au temps t, et soit $\mathcal{I}^*_{(u,v)}$ l'ensemble des chemins finissant avant t et tels que $\forall \mathcal{I}_{(u,v)} \in \mathcal{I}^*_{(u,v)}, w \in neighbours(v, arrival(\mathcal{I}_{(u,v)}))$, alors le chemin préfixe $\mathcal{J}'_{(u,v)}$ de $\mathcal{J}_{(u,w)}$ est plus court que tous les chemins de $\mathcal{I}^*_{(u,v)}$.

Une autre possibilité serait de modifier le graphe en entrée. Si un nœud u apparaît sur i intervalles et disparaît entre, il peut être remplacé par autant de nœuds $u_j, 1 \leq j \leq i$ existant sur l'intervalle j, avec le voisinage de u sur l'intervalle j.

5 Notes de Lecture

5.1 Time-varying graphs and dynamic networks [2]

Les auteurs identifient trois champs d'applications possibles des graphes dynamiques.

Delay-Tolerant Networks [3] La caractéristique principale de ces réseaux est la possibilité de ne pas être connexe à un moment donné. Les chemins généralement disponibles au cours du temps, et la diffusion ou le routage sont réalisables grâce au mécanisme de *store-carry-forward* (un exemple est celui du courrier électronique).

Opportunistic-Mobility Networks Ce sont aussi des DTN, avec la différence que les nœuds sont mobiles. Il existe des réseaux de taxis et de bus équipés de routeurs.

Real-World Complex Networks Dans cette catégorie tombent tous les autres types de réseaux (réseaux sociaux, transports, …), unifiés par l'utilisation d'un même formalisme et de concepts comme les chemins, la distance, la connexité, en fonction de l'évolution du réseau.

Les auteurs définissent ensuite les *time-varying graphs*, un modèle de graphe dynamique avec latence. Ils se contente dans le reste de l'article d'évoquer la latence sur les arêtes. La latence sur un nœud peut être imaginée comme un temps de calcul, ou de transfert.

Les auteurs décrivent le graphe sous-jacent d'un graphe dynamique, puis développent la notion de point de vue par rapport à l'évolution du graphe. Pour une arête, l'évolution se cantonne à une variation de disponibilité (l'arête est-elle présente ou non) et de latence. Pour un nœud, il s'agit de changements dans le voisinage. Enfin, par rapport au graphe, l'évolution est considérée comme un séquence de graphes statiques $G_1, G_2, \ldots$, où chaque graphe statique est défini par un événement topologique (et $G_i \neq G_{i+1}$), ou bien par un instantané au temps $t = i$ (et on peut avoir $G_i = G_{i+1}$). On parle d'*evolving graphs* pour représenter un graphe dynamique par une séquence de graphes statiques.

Les auteurs rappellent les définitions de sous-graphe d'un graphe dynamique, de chemin, de distance, et d'autres concepts.

La contribution principale de l'article est la classification des TVG selon des propriétés. En voici quelques-unes :

- $\exists u \in V : \forall v \in V, u \rightsquigarrow v$. On peut diffuser dans tout le graphe à partir d'un nœud.
- $\exists u \in V : \forall v \in V, v \rightsquigarrow u$. Il existe un nœud qui peut être atteint par tous les autres. C'est une condition nécessaire pour un calcul réparti sur tout le réseau, avec un nœud de sortie.
- $\forall u, v \in V, \exists \mathcal{J}_1 \in \mathcal{J}^*_{(u,v)}, \exists \mathcal{J}_1 \in \mathcal{J}^*_{(v,u)}$: $arrival(\mathcal{J}_1) \leq departure(\mathcal{J}_2)$. (*round connectivity*) Chaque nœud peut joindre tous les autres, et peut être joint ultérieurement. Cette condition peut être requise pour s'assurer explicitement que certains algorithmes terminent.
- $\forall e \in E, \forall t \in T, \forall k \in \mathbb{N}, \rho(e,t) = \rho(e, t + kp)$, pour un $p \in T$, et G est connexe. Cette propriété de périodicité peut être retrouvée dans les réseaux de transports, ou les DTN. On peut s'en servir pour réutiliser des calculs de chemins, modulo p.

Sont présentés les phénomènes de *small world*, où la distance (en termes de sauts) entre deux nœuds croît logarithmiquement par rapport au nombre de nœuds du graphe ; et la *fairness and balance*, l'écart-type de l'excentricité temporelle des nœuds.

En conclusion, les auteurs remarquent que peu de problèmes autour des graphes dynamiques ont été étudiés, mais si une situation peut être modélisée par un graphe dynamique, alors on peut se servir du formalisme et des phénomènes présentés pour l'étudier. Une situation intéressante est celle où on modélise les interactions d'un groupe social par un graphe dynamique et de on tente de minimiser l'éloignement temporel moyen afin d'accélérer les échanges.

5.2 Dynamic Networks : Models and Algorithms [5]

Il s'agit essentiellement d'une revue de la littérature, mais qui présente néanmoins deux résultats. Tout d'abord, les auteurs définissent leur cadre d'étude : des réseaux dynamiques dans lesquels les changements sont incessants, incontrôlables. Les algorithmes doivent terminer.

Le modèle utilisé est celui des *evolving graphs*, le temps est discrétisé. La fonction de présence des arêtes est dynamique, et déterminée par un **modèle d'adversaire**. On a l'*adaptative worst-case adversary* qui génère le graphe à la volée d'après l'évolution passée du graphe ; l'*oblivious worst-case adversary* impose un graphe prédéterminé ; et le *random-graph adversary*, lequel n'est pas vraiment un adversaire mais une loi de probabilité déterminant l'évolution du graphe.

Croissance de nœuds Les auteurs définissent la *vertex growth function* $g : \mathbb{N} \to \mathbb{N}$, qui généralise la notion de graphe expanseur. Un graphe (statique) $G = (V, E)$ a une croissance de nœuds g si pour tout ensemble de nœuds $S \subset V$ de taille $s = |S| \leq |V|/2$, l'ensemble des voisins $N(S) := \{v \in V \setminus S \mid \exists u \in S : (u, v) \in E\}$ est de taille au moins $|N(S)| \geq g(|S|)$.

Par exemple, si G est connexe, sa croissance est au moins 1 ; s'il est k-connexe, sa croissance est au moins $g(s) = k$; si G est un α-expanseur, alors $g(s) = \alpha s$.

On note $g^{(f)}$ l'itérée après f tours, avec $g^{(0)}(s) = s$, et pour $f > 0$, $g^{(f)}(s) = g^{(f-1)}(s) + g(g^{(f-1)}(s))$.

Si chaque graphe statique est un expanseur, le diamètre dynamique est $O(\log n)$; s'ils sont k-connexe, le diamètre est $O(n/k)$.

Lemme Soit $G = (V, E)$ un graphe dynamique tel que pour chaque tour r, le graphe statique $G(r)$ est g-croissant. Le diamètre dynamique de G est au plus $2d$, où d est le plus petit entier tel que $g^{(d)}(1) > |V|/2$.

Les auteurs considèrent ensuite le problème de la synchronisation d'horloges dans un réseau. Cela peut être utile pour discrétiser le temps, par exemple pour effectuer une action tous les t instants.

Les auteurs prouvent les bornes inférieure et supérieure du diamètre dynamique pour les graphes dont l'évolution est déterminée par une chaîne de Markov.

Afin de pouvoir résoudre divers problèmes dans les graphes dynamiques (par exemple compter le nombres de nœuds), les auteurs explorent un problème intermédiaire, le k-comité. Ce problème suppose que le graphe est toujours connecté.

k-comité Les nœuds doivent se partitionner en ensembles, appelés **comités**, tels que :

- La taille de chaque comité est au plus k,
- Si $k \geq n$, alors il n'existe qu'un comité contenant tous les nœuds.

Chaque comité dispose d'un unique identifiant, et le but est que chaque nœud émette un identifiant de comité validant les deux conditions.

Le k-comité peut être modifié pour résoudre d'autres problèmes, et dispose d'un algorithme de vérification, lui-même facilement modifiable.

6 Conclusion

Les graphes dynamiques fournissent un formalisme pour décrire des interactions au cours du temps. Bien que les auteurs proposent des modèles ou des dénominations légèrement différents, les concepts fondamentaux restent identiques. Les graphes dynamiques sont un sujet d'étude jeune, avec ses questions et ses pistes de recherche. Quels sont les algorithmes corrects et efficaces sur les problèmes déjà identifiés ? Quels mystères nous réservent les graphes dynamiques ? Au-delà de fournir un langage commun, peut-on utiliser les résultats de cette théorie pour l'analyse de réseaux informatiques ? routiers ? biologiques ?

Références

[1] Binh-Minh Bui-Xuan, Afonso Ferreira et Aubin Jarry. « Computing Shortest, Fastest, and Foremost Journeys in Dynamic Networks ». In : *Int. J. Found. Comput. Sci.* 14.2 (2003), p. 267-285. DOI : 10.1142/S0129054103001728. URL : https://doi.org/10.1142/S0129054103001728.

[2] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi et Nicola Santoro. « Time-varying graphs and dynamic networks ». In : *IJPEDS* 27.5 (2012), p. 387-408. DOI : 10.1080/17445760.2012.668546. URL : https://doi.org/10.1080/17445760.2012.668546.

[3] Vinton G. Cerf, Scott C. Burleigh, Adrian Hooke, Leigh Torgerson, Robert C. Durst, Keith L. Scott, Kevin Fall et Howard S. Weiss. « Delay-Tolerant Networking Architecture ». In : *RFC* 4838 (2007), p. 1-35. DOI : 10.17487/RFC4838. URL : https://doi.org/10.17487/RFC4838.

[4] Fabian Kuhn, Nancy A. Lynch et Rotem Oshman. « Distributed computation in dynamic networks ». In : *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*. 2010, p. 513-522. DOI : 10.1145/1806689.1806760. URL : http://doi.acm.org/10.1145/1806689.1806760.

[5] Fabian Kuhn et Rotem Oshman. « Dynamic networks : models and algorithms ». In : *SIGACT News* 42.1 (2011), p. 82-96. DOI : 10.1145/1959045.1959064. URL : http://doi.acm.org/10.1145/1959045.1959064.

[6] Matthieu Latapy, Tiphaine Viard et Clémence Magnien. « Stream Graphs and Link Streams for the Modeling of Interactions over Time ». In : *CoRR* abs/1710.04073 (2017). arXiv : 1710.04073. URL : http://arxiv.org/abs/1710.04073.