TD #18 — Graphes #3 (avec solutions)

Exercice 1. *Chemin de plus forte probabilité*

On considère un graphe orienté pondéré G, dont les pondérations $\in [0,1]$ dénotent des probabilités ; la probabilité d'un chemin dans G est donnée par le **produit** des probabilités de ses arcs. Étant donnés deux sommets v, w de G, on souhaite trouver un chemin $v \rightsquigarrow w$ (que l'on supposera toujours exister) de probabilité maximale.

1. Proposez une *réduction* de ce problème à celui d'une recherche de plus court chemin pondéré dans le sens usuel. (On pourra pour cela supposer que l'on est capable de calculer exactement dans $\mathbb{R}$...)

> On peut réduire ce problème à celui d'une recherche de plus-court chemin (au sens usuel) entre v et w dans le graphe G' de mêmes sommets & arcs que G, où l'on a pondéré chaque arc de probabilité x par $-\log(x)$: le poids (usuel) dans G' d'un chemin de longueur N de probabilité $p := \prod_{i=0}^{N-1} p_i$ dans G est de $\ell_p := -\sum_{i=0}^{N-1} \log(p_i) = -\log(p)$. Puisque $x \mapsto -\log(x)$ est décroissante et positive sur $[0,1]$, un chemin de probabilité maximale dans G est bien donné par un chemin de poids minimum (un plus-court-chemin) dans G', et puisque ces poids sont positifs on peut par exemple utiliser l'algorithme « de Dijkstra » pour calculer un tel chemin.

Exercice 2. *FW : détection et reconstruction de cycles de poids négatif*

On considère un graphe orienté pondéré (sur les arcs) G (sans boucle).

1. Montrez l'invariant pour la boucle principale de FW que si G ne contient pas de cycle de poids strictement négatif **de sommets intermédiaires dans** $[\![k]\!]$, les entrées hors diagonale $\boldsymbol{D}_{i,j}$ de la matrice des distances calculée par l'algorithme contiennent les longueurs (pondérées) de (plus-court) chemins **élémentaires** $i \rightsquigarrow j$ de sommets intermédiaires dans $[\![k]\!]$.

> L'initialisation est évidente pour $\boldsymbol{D}$ égale à la matrice d'adjacence de G.
> Pour la conservation, il suffit de considérer le cas où G ne possède pas de cycle etc. et où $\boldsymbol{D}_{i,j}$ est modifiée en $\delta := \boldsymbol{D}_{i,k} + \boldsymbol{D}_{k,j}$ (sinon il n'y a rien à faire). Par hypothèse on a alors que $\boldsymbol{D}_{i,k}$ (resp. $\boldsymbol{D}_{k,j}$) sont les longueurs de chemins élémentaires $i \rightsquigarrow k$ (resp. $k \rightsquigarrow j$) de sommets intermédiaires dans $[\![k]\!]$. Puisque δ correspond à la longueur du chemin $p := i \rightsquigarrow k \rightsquigarrow j$ de sommets intermédiaires dans $[\![k+1]\!]$ obtenu en concaténant ces deux chemins et que la variable k est incrémentée en fin d'itération, il suffit de montrer que si p n'est pas élémentaire alors G possède un cycle de poids strictement négatif de sommets intermédiaires dans $[\![k+1]\!]$. Si p n'est pas élémentaire, il s'écrit nécessairement comme $i \rightsquigarrow v \rightsquigarrow k \rightsquigarrow v \rightsquigarrow j$ pour un certain v, et puisque c'est un plus-court-chemin $i \rightsquigarrow j$ cela implique que $v \rightsquigarrow k \rightsquigarrow v$ de sommets intermédiaires dans $[\![k+1]\!]$ est de poids **strictement** négatif (s'il était simplement de poids nul, aucune mise à jour du chemin $i \rightsquigarrow v \rightsquigarrow j$ de sommets intermédiaires dans $[\![k]\!]$ ne serait nécessaire). Enfin il est toujours possible dans ce cas de choisir v de façon à ce que $v \rightsquigarrow k \rightsquigarrow v$ soit élémentaire (et donc simple, le graphe étant orienté, et donc un cycle), car tout éventuel sommet répété dans p ne peut apparaître que deux fois et « de part et d'autre » de k, puisque les chemins $i \rightsquigarrow v \rightsquigarrow k$ et $k \rightsquigarrow v \rightsquigarrow j$ sont élémentaires.
>
> **N.B.** Cet invariant est juste une version un peu plus forte de l'invariant « de base » de FW, où l'on veut pouvoir exprimer quelque chose sur le comportement de l'algorithme en cas de présence de cycles de poids strictement négatif, mais avant que ces cycles ne soient « considérés ».

2. Montrez que si G possède un cycle de poids strictement négatif, soit $i \rightsquigarrow k \rightsquigarrow i$ l'un parmi ces cycles minimisant k l'on aura que $\boldsymbol{D}_{i,k} + \boldsymbol{D}_{k,i} < 0$ à la $k^{\text{ième}}$ itération de la boucle principale.

> Par hypothèse G ne possède aucun cycle de poids strictement négatif de sommets intermédiaires dans $[\![k]\!]$, mais en possède un $i \rightsquigarrow k \rightsquigarrow i$; on note $\ell_{i,k}$ (resp. $\ell_{k,i}$) les longueurs des portions (élémentaires) $i \rightsquigarrow k$ (resp. $k \rightsquigarrow i$) de ce cycle, dont tous les sommets intermédiaires sont dans $[\![k]\!]$. Par la question précédente, au début de la $k^{\text{ième}}$ itération de la boucle principale de FW $\boldsymbol{D}_{i,k}$ et $\boldsymbol{D}_{k,i}$ contiennent respectivement les longueurs de plus-court-chemins élémentaires $i \rightsquigarrow k$ et $k \rightsquigarrow i$ à sommets intermédiaires dans $[\![k]\!]$, ce qui donne $\boldsymbol{D}_{i,k} \leq \ell_{i,k}$, $\boldsymbol{D}_{k,i} \leq \ell_{k,i}$, et l'on a donc bien $\boldsymbol{D}_{i,k} + \boldsymbol{D}_{k,i} \leq \ell_{i,k} + \ell_{k,i} < 0$.

3. Conclure sur la capacité de FW à détecter la présence d'un cycle de poids strictement négatif, et à reconstruire un éventuel tel cycle.

Par la question précédente, la présence d'un cycle $i \rightsquigarrow k \rightsquigarrow i$ de poids strictement négatif minimisant k pourra toujours être détectée lors de la $k^{\text{ième}}$ itération de la boucle principale par le fait que $\boldsymbol{D}_{i,k} + \boldsymbol{D}_{k,i} < 0$. Par contraposée, si pour toute itération k et tout sommet i l'on a $\boldsymbol{D}_{i,k} + \boldsymbol{D}_{k,i} \geq 0$, alors G ne possède pas de cycle de poids strictement négatif.
Dans le cas contraire, on va montrer qu'il existe toujours un cycle de poids strictement négatif et que l'on peut reconstruire celui-ci. On considère l'itération k minimale telle qu'il existe un i avec $\boldsymbol{D}_{i,k} + \boldsymbol{D}_{k,i} < 0$. Par minimalité de k et ci-dessus, G ne contient pas de cycles de sommets intermédiaires dans $[\![k]\!]$ et donc par la première question les chemins $i \rightsquigarrow k$ et $k \rightsquigarrow i$ induits par $\boldsymbol{D}_{i,k}$ et $\boldsymbol{D}_{k,i}$ sont élémentaires ; on considère deux cas :

— soit $i \rightsquigarrow k \rightsquigarrow i$ est élémentaire et il donne le cycle voulu ;
— sinon il est nécessairement de la forme $i \rightsquigarrow v \rightsquigarrow k \rightsquigarrow v \rightsquigarrow i$ pour un certain v tel que $v \rightsquigarrow k \rightsquigarrow v$ est un cycle (par le même raisonnement qu'à la première question). La longueur totale du chemin $i \rightsquigarrow i$ (qui n'est pas un cycle) est (pour des notations évidentes) $\ell_{i,v} + \ell_{v,k} + \ell_{k,v} + \ell_{v,i} < 0$. Par $v < k$ et la minimalité de k pour la présence d'un cycle de poids strictement négatif, l'on a $\ell_{i,v} + \ell_{v,i} > 0$ (que le chemin $i \rightsquigarrow v \rightsquigarrow i$ correspondant soit élémentaire ou non, puisque les éventuels cycles qu'il contiendrait seraient aussi de poids > 0). Il s'ensuit que $\ell_{v,k} + \ell_{k,v} < 0$ et que $v \rightsquigarrow k \rightsquigarrow v$ est le cycle voulu.

Enfin les chemins induits par les entrées de $\boldsymbol{D}$ peuvent facilement être reconstruits récursivement en maintenant séparément une matrice $\boldsymbol{P}$ qui affecte $\boldsymbol{P}_{i,j}$ à k à chaque itération k où $\boldsymbol{D}_{i,j}$ est modifié.

Exercice 3. *Rencontre au milieu*

Soit $G = S, A$ un graphe (éventuellement orienté) pondéré à pondération positive et $v, w \in S$, on cherche un sommet m se trouvant le plus « au milieu » de v et w dans G, c'est à dire un qui minimise le maximum des distances (pondérées) $d(v, m)$ et $d(w, m)$.

1. Proposez une modification de l'algorithme « de Dijkstra » qui permette de résoudre ce problème efficacement. En particulier, pour G représenté par tableau de listes d'adjacence, le coût de l'algorithme modifié doit être un $O(\#A \log \#S + \#S)$.

Une idée consiste à entrelacer deux recherches de plus court chemin (l'une depuis v et l'autre depuis w) et à s'arrêter dès que l'on visite un sommet depuis les deux parcours (et renvoyer celui-ci). Ceci peut s'implémenter efficacement en mutualisant la file de priorité de l'algorithme « de Dijkstra » mais en maintenant deux tableaux de visite séparés (l'un pour la recherche depuis v et l'autre pour celle depuis w). Lorsque l'on insère un sommet dans la file on mémorise pour celui-ci l'origine de son parcours (v ou w), ce qui quand on défile un sommet permet de tester s'il a déjà été découvert depuis l'*autre* origine, et sinon de mettre à jour le « bon » tableau de visite.
La correction de cet algorithme modifié peut se prouver d'une façon similaire à celle de celui « de Dijkstra ». Notamment, au moment où l'on défile pour sa seconde origine (disons w) le premier sommet m visité à la fois depuis v et w, les éventuels autres sommets encore dans la file sont nécessairement à une distance à v ou w supérieure à $d(w, m) > d(v, m)$, et donc supérieur au maximum des deux. De même, les éventuels sommets encore non visités (pour l'une ou l'autre des origines) et non présents dans la file ne pourront qu'être (éventuellement) ajoutés dans la file avec une distance supérieure à $d(w, m)$. Il s'ensuit que m est bien l'un des sommets recherchés.
Enfin, le surcoût de cet algorithme par rapport à celui « de Dijkstra » n'est que d'un facteur constant, puisque l'on effectue juste deux recherches de plus court chemin. C'est donc bien un $O(\#A \log \#S + \#S)$ pour une implémentation efficace de la file de priorité.

2. Implémentez cet algorithme en OCaml (vous pouvez supposer qu'une structure de données de file de priorité minimum (par exemple impérative) est déjà disponible).

On propose :

```
let mitmsp g v v' =
    let n = Array.length g in
    let dist = Array.make n (-1) in
    let dist' = Array.make n (-1) in
    let pq = MinPq.create () in
    let push_cond dt d s (w, dvw) =
        if dt.(w) = -1 then MinPq.add pq (d + dvw, s, w)
    in
    let rec loop
        = function
        | None -> None
```

```
        | Some (d, s, w)
              -> let dt, dt' = if s = v then dist, dist' else dist', dist in
                 if dt'.(w) <> -1 then
                     Some (w, (max d dt'.(w)))
                 else begin
                     if dt.(w) = -1 then begin
                         dt.(w) <- d ;
                         List.iter (push_cond dt d s) g.(w)
                     end ;
                     loop (MinPq.pop_min pq)
                 end
    in
    let () = MinPq.add pq (0, v, v) in
    let () = MinPq.add pq (0, v', v') in
    loop (MinPq.pop_min pq)
```

Exercice 4. *Graphe équivalent minimum d'un DAG*

Soit $G = S, A$ un graphe orienté acyclique connexe, on définit sont *graphe équivalent (pour l'accessibilité) minimum* comme un sous-graphe $G' = S, A' \subseteq A$ de G de même fermeture transitive que G (c'est à dire tel que pour tout $v, w \in S$, il existe un chemin $v \rightsquigarrow w$ dans G' ss.'il en existe un dans G) et minimisant $\#A'$. Autrement dit, G' est de même fermeture transitive que G, et ce n'est le cas d'aucun de ses sous-graphes stricts $G'' = S, A'' \subset A'$.

1. Donnez des graphes équivalents minimum de G_1 et G_2 respectivement définis par :

   — $G_1 : S = [\![3]\!], A = \{(0,1),(0,2),(2,1)\}$
   — $G_2 : S = [\![5]\!], A = \{(0,1),(0,3),(1,2),(3,4),(4,2))\}$

   — $G'_1 : A'_1 = \{(0,1),(1,2)\}$
   — $G'_2 = G_2$.

Dans toute la suite de l'exercice, on suppose (sans perte de généralité) que $S = [\![n]\!]$ pour un certain n.
On définit une fonction r sur les graphes de la façon suivante : si $G = S, A$ possède un triplet $u, v, w \in S$ tels que (u, w), $(v, w) \in A$ et qu'il existe un chemin $u \rightsquigarrow v$ dans G, alors on définit $r(G)$ comme le graphe $G' = S, A \setminus \{(u, w)\}$ obtenu en supprimant celui de ces (éventuellement multiples) triplets minimum pour l'ordre lexicographique. Sinon l'on définit $r(G)$ comme G lui-même.
On propose alors l'algorithme « glouton » suivant pour calculer G' : on renvoie le premier point-fixe obtenu en itérant r à partir de G.

2. Montrez que cet algorithme termine.

   À chaque application de r, ou bien l'algorithme termine ou bien la fonction renvoie un graphe contenant strictement moins d'arcs que son argument. Ce dernier étant minoré par zéro, il constitue un variant.

3. Montrez que cet algorithme renvoie bien un graphe équivalent minimum.

   On montre d'abord que pour tout G, $r(G)$ a la même fermeture transitive que G (dans l'algorithme esquissé ci-dessus, on a donc l'invariant de boucle que le graphe considéré à la $i$ème itération a la même fermeture transitive que le graphe initial). Si $r(G) = G$ alors il n'y a rien à montrer. Sinon il suffit de montrer que pour tout chemin $s \rightsquigarrow t$ quelconque dans G, il existe également un tel chemin dans G'. En reprenant les mêmes notations que ci-dessus, si $s \rightsquigarrow t$ n'emprunte pas l'arc (u, w) dans G alors il existe à l'identique dans G' ; sinon il suffit de remplacer cet arc par le chemin $u \rightsquigarrow v \to w$ qui est toujours présent dans G' : en effet, le graphe étant acyclique le chemin $u \rightsquigarrow v \to w$ est élémentaire ouvert (en particulier, $v \neq u$) et donc aucun de ses arcs n'a été supprimé dans G'.
   On montre maintenant qu'on ne peut pas supprimer d'arc au graphe $G' = S, A'$ renvoyé par l'algorithme sans en changer la fermeture transitive. Par définition de l'algorithme, l'on a $r(G') = G'$. Supposons par l'absurde :(qu'il existe un arc $(u, w) \in A'$ tel que $G'' = S, A' \setminus \{u, w\}$ ait la même fermeture transitive que G'. Puisqu'il existe un chemin $u \rightsquigarrow w$ (*viz.* u $\to$ w) dans G', G'' doit également posséder un tel chemin. Ce dernier ne peut pas inclure l'arc (u, w), et il existe donc un **autre** chemin $u \rightsquigarrow w$ dans G'. Celui-ci doit être de longueur au moins deux, donc de la forme $u \rightsquigarrow v \to w$ pour un certain $v \neq u$. Puisque G'' est un sous-graphe de G', ce chemin est aussi un chemin de G', et l'existence de $(v, w) \in A'$ et alors en contradiction avec le fait que $r(G') = G'$.

4. Montrez que le graphe renvoyé par l'algorithme précédent est unique. (Plus généralement, montrez que G possède un unique graphe équivalent minimum.)
Indication : on peut procéder par l'absurde, de façon assez similaire à la question précédente :(

On suppose par l'absurde :(que G possède deux sous-graphes équivalents minimaux distincts G'_1 et G'_2. Soit $(u,w) \in A'_1 \setminus A'_2$ (qui est non vide par hypothèse), puisqu'il existe un chemin $u \rightsquigarrow w$ dans G'_1 il en existe également un dans G'_2. Celui-ci est de longueur au moins deux et est donc de la forme $u \rightsquigarrow v \to w$ pour un certain v. Ceci implique à son tour l'existence de deux chemins $u \rightsquigarrow v$ et $v \rightsquigarrow w$ dans G'_1 ; mais alors l'arc (u,w) est redondant dans A'_1 et pourrait en être supprimé sans changer sa fermeture transitive, ce qui contredit la minimalité de G'_1. Un tel arc ne peut donc pas exister, et l'on a $G'_1 = G'_2$.

5. Quelle conséquence cette unicité peut-elle avoir sur une implémentation de l'algorithme glouton ?

L'ordre dans lequel les triplets sont considérés par l'algorithme glouton ne changent pas le fait que le résultat renvoyé est un graphe équivalent minimum (l'ordre précis n'est pas utilisé dans la preuve de correction de l'algorithme). Puisque ce résultat est en fait unique, cela veut dire que l'ordre de traitement des triplets dans l'algorithme glouton n'a pas d'importance. (Formellement, chaque ordre donne un algorithme différent mais les résultats de ceux-ci sont indistinguables.)

On suppose maintenant que G est représenté par une matrice d'adjacence $\boldsymbol{A}$ à valeur dans $\{\top, \bot\}$ et que l'on dispose également de la matrice d'adjacence $\boldsymbol{A}^F$ de sa fermeture transitive (également à valeur dans $\{\top, \bot\}$).

6. Soit $\odot$ la multiplication de matrices dans le semi-anneau $\{\vee, \wedge\}$, (on remplace l'addition (resp. la multiplication) par le « OU » (resp. le « ET ») logique) et $\boldsymbol{A}^{2+} := \boldsymbol{A} \odot \boldsymbol{A}^F$, montrez que $\boldsymbol{A}^{2+}_{v,w} = \top$ (vaut « VRAI ») ss.'il existe un chemin $v \rightsquigarrow w$ dans G de longueur **au moins deux**.

Par définition des matrices $\boldsymbol{A}$ et $\boldsymbol{A}^F$, $\boldsymbol{A}_{v,w} = \top$ ssi. (v,w) est un arc de G, et $\boldsymbol{A}^F_{v,w} = \top$ ss.'il y existe un chemin $v \rightsquigarrow w$ de longueur au moins un. Par définition du produit de matrice utilisé, $\boldsymbol{A}^{2+}_{v,w} = \vee_{i=0}^{n-1}(\boldsymbol{A}_{v,i} \wedge \boldsymbol{A}^F_{i,w})$, et vaut donc $\top$ ss.'il existe i tel que l'on a $\boldsymbol{A}_{v,i} = \boldsymbol{A}^F_{i,w} = \top$, c'est à dire ss.'il existe un arc $(v,i) \in A$ et un chemin $i \rightsquigarrow w$ de longueur au moins un dans G, soit un chemin $v \to i \rightsquigarrow w$ de longueur au moins deux entre v et w dans G.

7. Déduisez de cela (et de tout ce qui précède) un algorithme qui calcule le graphe équivalent minimum d'un graphe orienté acyclique G de n sommets en temps $O(n^3)$.

Quelque soit la représentation initiale de G, on peut en temps $O(n^2)$ en calculer une représentation $\boldsymbol{A}$ par matrice d'adjacence. On peut ensuite calculer la matrice $\boldsymbol{A}^F$ de sa fermeture transitive en temps $O(n^3)$, par exemple avec l'algorithme « de Floyd-Warshall ». Le produit $\boldsymbol{A}^{2+} := \boldsymbol{A} \odot \boldsymbol{A}^F$ se calcule également (de façon naïve) en temps $O(n^3)$. Enfin par la question précédente, les arcs (u,w) de G « supprimés » par l'algorithme glouton sont *exactement* ceux tels que $\boldsymbol{A}_{u,w} = \top$ (il existe un arc $u \to w$) et $\boldsymbol{A}^{2+}_{u,w} = \top$ (il existe un chemin $u \rightsquigarrow w$ de longueur au moins deux). Il suffit donc de calculer les n^2 coefficients de la matrice d'adjacence $\boldsymbol{A}'$ de G' comme $\boldsymbol{A}'_{i,j} := \boldsymbol{A}_{i,j} \wedge \neg \boldsymbol{A}^{2+}_{i,j}$, ce qui peut se faire en temps $O(n^2)$. (On peut remarquer au passage que ce dernier calcul peut en fait se faire de façon trivialement parallèle : le fait qu'un arc ne soit pas inclus dans G' peut se décider indépendamment de tous les autres arcs.)