Étude d'un Problème Algorithmique Intervenant dans la Reconfiguration des Réseaux WDM

Phạm Quang Cường
encadré par
David Coudert et Stéphane Pérennes

Projet MASCOTTE, CNRS-I3S-INRIA
Inria Sophia Antipolis

15 septembre 2004

Table des matières

1 Introduction

1.1 Origine du problème

Le principe du WDM (Wave Division Multiplexing ou Multiplexage en Longueurs d'Onde) est de permettre à une même fibre optique de porter plusieurs longueurs d'onde, augmentant ainsi son débit. Évidemment, deux requêtes empruntant la même fibre doivent alors circuler sur des longueurs d'onde différentes.

Du fait des ajouts et des suppressions de requêtes dûs à l'évolution du trafic, il est parfois nécessaire de reconfigurer le réseau, comme le montre l'exemple suivant :

On dispose d'une fibre à deux longueurs d'onde l_1 et l_2 reliant quatre noeuds du réseau A, B, C, D. Considérons la séquence d'ajouts/suppressions ci-dessous (voir la figure 1) :

0. initialement, A→B sur l_1
1. ajout de A→D, nécessairement sur l_2
2. ajout de C→D, nécessairement sur l_1
3. suppression de A→D
4. ajout de A→C, nécessairement sur l_2

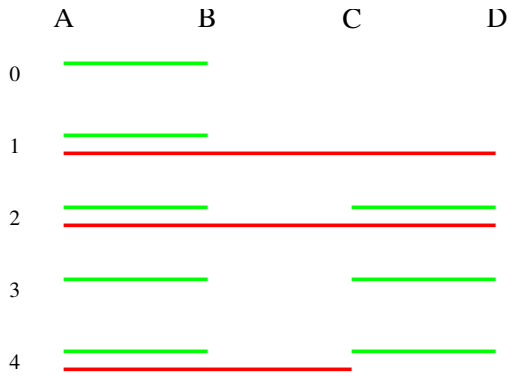

FIG. 1 – Exemple d'une séquence d'ajouts/suppressions

Supposons qu'on veuille à présent ajouter la requête B→D. Cela est impossible alors qu'une autre configuration permet de le faire (A→C, C→D sur l_1, A→B, B→D sur l_2). Il est donc nécessaire de reconfigurer le réseau, en attribuant, par exemple, la longueur d'onde l_1 à A→C et l_2 à A→B. Cependant, il n'est pas possible de déplacer A→C sur l_1 car elle entrerait en conflit avec A→B. Il est facile de voir qu'on ne peut pas résoudre ce problème sans interrompre temporairement une des deux requêtes en question.

Dans le cadre de ce stage, nous considérons un modèle simple qui permet de surmonter cette difficulté. Nous supposons l'existence d'une longueur d'onde supplémentaire, l_3, qu'on peut utiliser uniquement lors de la reconfiguration. Le problème

précédent pourrait se résoudre ainsi : on *déplace* A→C sur l_3, puis A→B sur l_2, et enfin A→C sur l_1. Nous insistons sur le verbe « déplacer » car techniquement, un déplacement peut se faire sans interruption de communication, contrairement à une suppression suivie d'un ajout. Nous supposons de plus qu'une telle longueur d'onde de « secours », appelée mémoire temporaire dans la suite, ne peut supporter qu'une seule requête à la fois. Le problème est de savoir, étant données deux configurations C et C', quelle est le nombre minimal de mémoires temporaires permettant de passer de C à C'.

La modélisation peut se faire ainsi : à chaque requête on associe un sommet d'un graphe orienté G. Considérons deux requêtes r et s. Supposons que dans C', r demande une ressource occupée par s dans C. Dans ce cas, on ne peut pas traiter r avant que s soit traitée ou qu'elle soit mise en mémoire temporaire. On matérialise cette dépendance par un arc orienté (r, s). Cependant, une requête r qu'on n'a pas à déplacer ne se voit pas attribuer d'arc (r, r).

La section suivante présente une formalisation qui abstrait du problème toute référence à la reconfiguration des réseaux WDM, mais le lecteur pourra facilement rétablir les correspondances à l'aide du paragraphe précédent.

1.2 Formalisation et notations

Considérons un graphe orienté $G = (V, A)$ (avec éventuellement des boucles, mais sans arc multiple) dont les sommets représentent les tâches à effectuer et les arcs, les dépendances entre ces tâches : A contient l'arc (u, v) si et seulement si la tâche représentée par u doit être traitée *après* que celle représentée par v est traitée ou mise en mémoire temporaire.

On veut traiter toutes les tâches (i.e. supprimer tous les sommets du graphes) en respectant les règles suivantes :

- on peut à tout moment marquer un sommet non encore marqué (mise en mémoire temporaire). Ce faisant, on supprime tous les arcs *entrant* dans ce sommet.
- on peut supprimer un sommet s'il n'a aucun successeur (traitement d'une tâche qui ne dépend d'aucune autre). Si le sommet était initialement marqué, cela correspond à la libération d'une place en mémoire temporaire.

Si l'on peut supprimer tous les sommets du graphe, tel qu'à tout moment, le nombre de sommets marqués simultanément [1] est au plus k, le graphe est dit k-*mémoires*. On note $K(G)$ le plus petit k tel que G soit k-mémoires. Le problème qui nous intéresse se pose en ces termes : étant donné un graphe orienté G, calculer $K(G)$.

Notations Dans la suite, nous utiliserons les notations suivantes :

Pour un graphe orienté G :

- si rien est précisé, V et A représentent l'ensemble des sommets et des arcs de G, avec $|V| = n$, $|A| = m$.

[1] On dira aussi « le nombre de mémoires utilisées simultanément ».

- pour $u \in V$, $\Gamma^-(u) = \{v \in V | (v,u) \in A\}$ (l'ensemble des prédécesseurs de u) et $\Gamma^+(u) = \{v \in V | (u,v) \in A\}$ (l'ensemble des successeurs de u).
- pour $U \subset V$, $\Gamma^-(U) = \bigcup_{u \in U} \Gamma^-(u)$, de même pour $\Gamma^+(U)$.
- pour $U \subset V$, $\delta(U) = \Gamma^+(U) - U$ (bord externe de U).
- pour $u \in V$, on notera abusivement $G - u$ le graphe induit sur G par $V - \{u\}$.

1.3 Travail du stage

Dans un premier temps, nous nous sommes appliqué à comprendre, à programmer et à prouver les algorithmes pour tester si un graphe est 0,1,2-mémoire(s). Nous n'avons cependant pas réussi à adapter nos méthodes pour résoudre le cas 3-mémoires et, plus généralement, le cas k-mémoires, pour $k \geq 3$ (partie 2).

Cet échec nous a conduit à étudier ensuite, à défaut des valeurs exactes de $K(G)$, des encadrements de celui-ci dans des cas particuliers : grilles, graphes planaires, graphes à mineur exclu, ... (partie 3).

Enfin, une nouvelle manière d'aborder le problème a mis en évidence le lien avec la classe des problèmes d'arrangement. Cela nous a permis de prouver sa NP-complétude et de le résoudre quasi-complètement dans le cas des graphes orientés symétriques (partie 4).

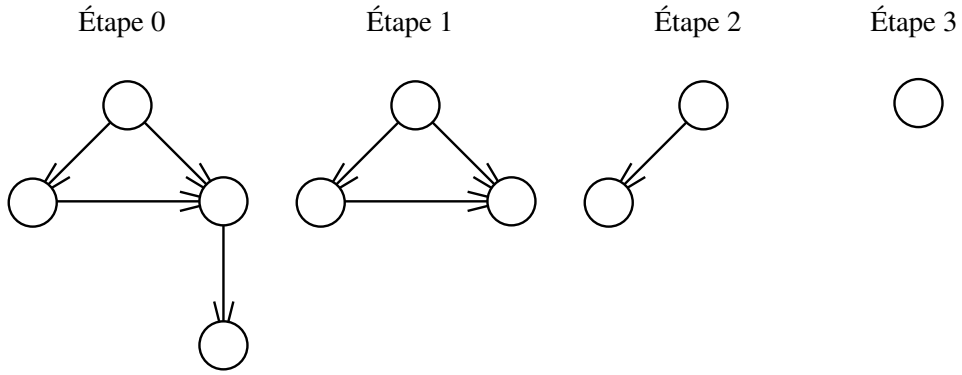

FIG. 2 – Exemple d'un graphe 0-mémoire

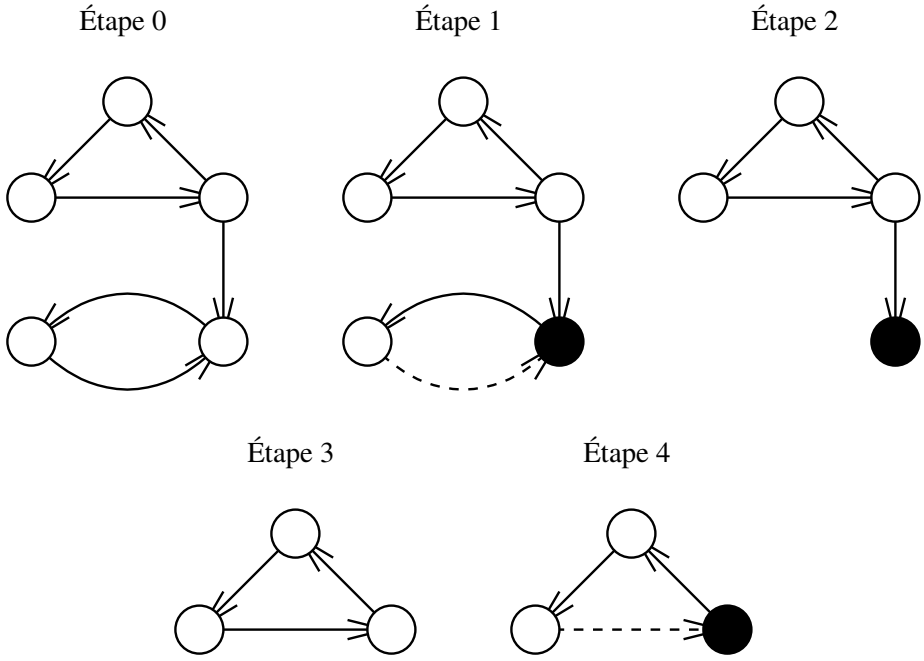

FIG. 3 – Exemple d'un graphe 1-mémoire

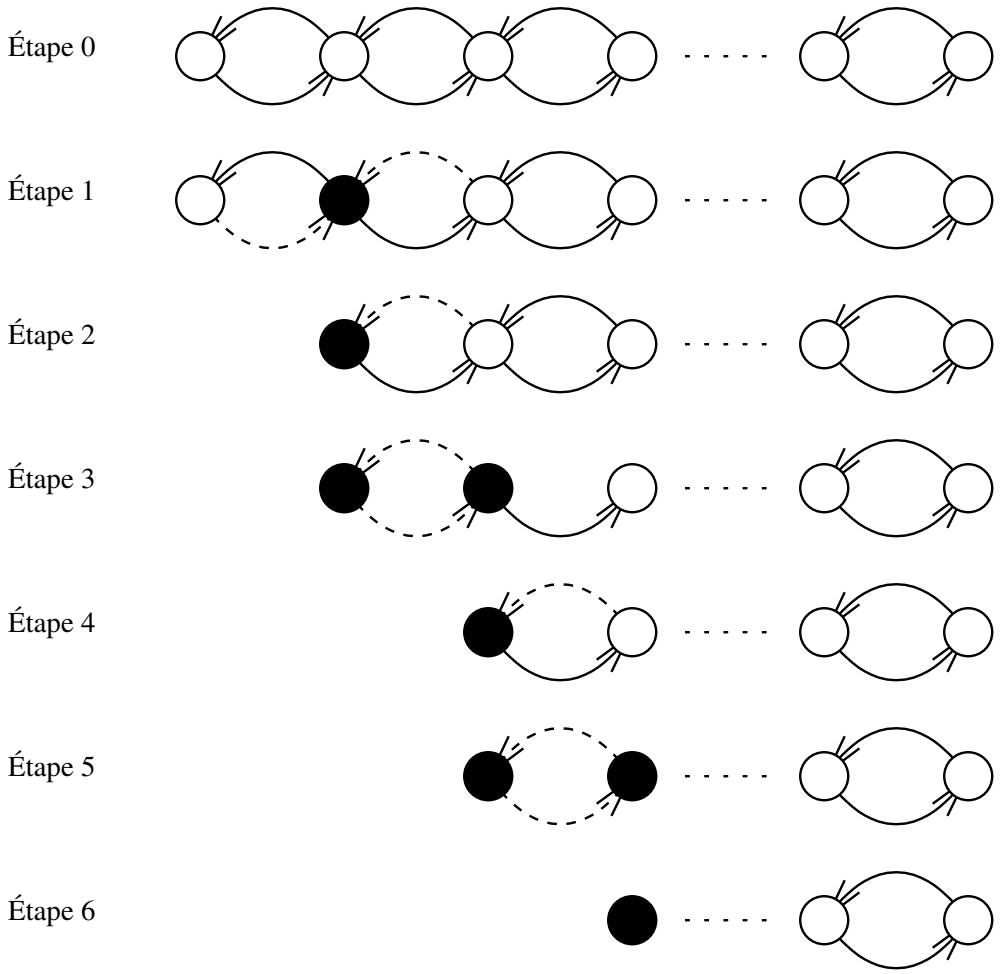

FIG. 4 – Exemple d'un graphe 2-mémoires

2 Caractérisation des graphes k-mémoires, k=0,1,2

2.1 Caractérisation des graphes 0-mémoire

La principale difficulté réside dans l'existence des cycles. Un graphe acyclique se traite très facilement comme le montre la proposition 1. Notre objectif est de briser tous les cycles en posant des mémoires (voir la proposition 9), mais la réutilisation possible des mémoires constitue une difficulté non négligeable.

Proposition 1 *Un graphe est 0-mémoire si et seulement s'il est acyclique. On peut ainsi tester si un graphe est 0-mémoire en* $O(m+n)$.

Démonstration ($\Rightarrow$) Supposons que le graphe n'est pas acyclique, il possède alors au moins un cycle et il est bien évident que l'on ne peut pas traiter les sommets appartenant à ce cycle avec 0 mémoire.

($\Leftarrow$) Supposons à présent que le graphe est acyclique. On peut alors le trier topologiquement ([6] pp. 476-478). Il suffit ensuite de supprimer les sommets dans cet ordre topologique, sans utiliser de mémoire.

On peut tester si un graphe est acyclique en effectuant un parcours en profondeur sur ce graphe, ce qui prend un temps $O(m+n)$ ([6] pp.468-478). On en déduit un algorithme EST-0-MÉMOIRE de complexité $O(m+n)$ qui teste si un graphe peut être traité avec 0 mémoire. □

Dans le même ordre d'idée, la proposition suivante nous permet de nous restreindre par la suite au cas des graphes fortement connexes.

Proposition 2 *Un graphe est k-mémoires si et seulement si toutes ses composantes fortement connexes (CFC) sont k-mémoires.*

Démonstration ($\Rightarrow$) Supposons qu'il existe une CFC qui n'est pas k-mémoires, c'est-à-dire qu'on ne peut pas la traiter en utilisant k mémoires seulement. Mais alors le graphe lui-même, étant plus gros, ne peut non plus se traiter avec k mémoires.

($\Leftarrow$) Supposons maintenant que toutes les CFC d'un graphe G sont k-mémoires. On peut construire un graphe G^{CFC} dont les sommets sont les CFC de G et qui contient un arc $(\hat{u},\hat{v})$ si, dans G, la CFC représentée par $\hat{u}$ comprend au moins un sommet qui pointe sur un sommet de la CFC représentée par $\hat{v}$. Par construction, G^{CFC} est acyclique ([6] p. 485). On peut alors le trier topologiquement et traiter les différentes CFC dans cet ordre topologique, en remarquant qu'à chaque fois qu'on finit le traitement d'une CFC, on se retrouve de nouveau avec k mémoires libres. □

2.2 Caractérisation des graphes 1-mémoire

2.2.1 Un algorithme naïf

Considérons un graphe G fortement connexe. Nous voulons savoir si l'on peut traiter G à l'aide d'une unique mémoire. Remarquons d'abord que cette mémoire ne peut être utilisée qu'une seule fois.

En effet, puisque le graphe est fortement connexe, on ne peut supprimer aucun sommet sans utiliser de mémoire. Ainsi, on doit commencer par installer la mémoire sur un sommet, par exemple u. Soit v un sommet distinct de u. Comme le graphe est fortement connexe, il existe un chemin $(u = u_1, u_2, ...u_{k-1}, u_k = v)$ tel que $\forall 1 \leq i \leq k-1, (u_i, u_{i+1}) \in A$. Pour pouvoir récupérer la mémoire posée sur u, il faut que l'on ait supprimé u_1 antérieurement sans utiliser de mémoire, et pour supprimer u_2, il faut avoir supprimé u_3, etc. jusqu'à $u_k = v$ Ainsi, nous avons montré que pour pouvoir récupérer la mémoire, il faut avoir supprimé tous les sommets du graphe : la mémoire n'a donc été utilisé qu'une seule fois.

Un algorithme évident est de choisir un sommet du graphe, de poser une mémoire dessus, et de voir si l'on peut traiter tous les autres sommets avec 0 mémoire. Le graphe est 1-mémoire si et seulement si un tel sommet existe.

Algorithme 1 EST-1-MÉMOIRE-NAÏF(G)

Requiert: Un graphe G fortement connexe
Fournit: **VRAI** si G est 1-mémoire, **FAUX** sinon

```
res ← FAUX
tant que res = FAUX ∧ il reste des sommets non encore choisis faire
    Choisir un sommet u non encore choisi
    res ← EST-0-MÉMOIRE(G − u)
fin tant que
retourner res
```

Proposition 3 *EST-1-MÉMOIRE-NAÏF se termine, est correct et a une complexité en* $O(n(m+n))$.

Démonstration Il y a n choix possibles pour le sommet sur lequel on pose la mémoire. Ensuite, pour tester si le graphe résultant est 0-mémoire, on utilise EST-0-MÉMOIRE qui est en $O(n+m)$. Au final EST-1-MÉMOIRE-NAÏF est bien en $O(n(n+m))$.

Remarquons qu'on a testé $G - u$ au lieu de « G dans lequel on a marqué u », mais cela est équivalent car si u est marqué, il ne peut intervenir dans aucun cycle. □

2.2.2 Un algorithme plus efficace

Considérons un sommet u ayant un unique successeur v [2]. Dès qu'on met en mémoire ou qu'on supprime v, on peut supprimer immédiatement u. En quelque sorte, la présence de u n'ajoute rien à la « sémantique » du graphe. Moyennant quelques précautions, on peut donc éliminer u (voir la figure 5). Plus formellement :

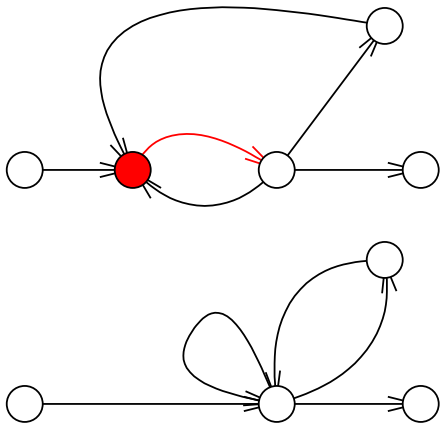

FIG. 5 – Élimination d'un sommet

Proposition 4 *Soit $G = (V, A)$ un graphe orienté fortement connexe et $u, v \in V$ tels que $\Gamma^+(u) = \{v\}$. Considérons $G' = (V', A')$, où $V' = V - \{u\}$ et où A' est constitué :*

- *des arcs de A non incidents à u*
- *d'un nouvel arc (w, v) pour tout arc $(w, u) \in A$*

Alors $K(G) = K(G')$.

Démonstration Montrons d'abord $K(G') \geq K(G)$. Soit une stratégie s' de traitement de G' qui utilise $K(G')$ mémoires. Transformons s' en une stratégie s de traitement de G comme suit :

- si dans s' on marque v, dans s, on marque v et on supprime immédiatement u (car u n'a plus de successeur).
- si dans s' on supprime v, dans s on supprime v et on supprime immédiatement u (si ce n'est pas déjà fait).
- on ne change pas les autres actions.

Il est facile de voir s est valable et n'utilise pas plus de mémoire que s'.

Montrons maintenant $K(G) \geq K(G')$. Soit une stratégie s de traitement de G qui utilise $K(G)$ mémoires. Deux cas possibles :

[2] Puisque le graphe est fortement connexe, le cas $u = v$ intervient uniquement quand le graphe a un seul sommet. Cela arrive éventuellement à la fin de l'algorithme, comme nous le verrons plus tard.

– v n'est jamais marqué. Alors, v est supprimé avant u. On peut supposer sans perte de généralité que u est supprimé immédiatement après v. Dans G' il suffit alors de supprimer v. Toutes les autres actions sur G se transforment naturellement en actions sur G'.
– v est marqué à un moment donné. On peut supposer sans perte de généralité que u est supprimé immédiatement après que v est marqué. Dans G' il suffit alors de marquer v. Toutes les autres actions sur G se transforment naturellement en actions sur G'.

Il est facile de voir s' est valable et n'utilise pas plus de mémoire que s. □

Contraction d'un graphe L'algorithme CONTRACTER-GRAPHE présenté ci-dessous réitère au maximum l'action d'éliminer des sommets à unique successeur pour transformer le graphe initial en un graphe équivalent du point de vue « mémoires » mais plus simple à analyser.

L'ensemble des sommets incidents à un sommet u est fourni sous la forme de deux vecteurs booléens de taille n : $\Gamma^-(u)$ et $\Gamma^+(u)$ qui gardent en mémoire respectivement les prédécesseurs et les successeurs de u. On construit et maintient également une file F contenant à chaque itération tous les sommets à unique successeur(l. 5).

Tant que la file F est non vide (l. 8) (i.e. qu'il reste des sommets à unique successeur), on extrait un élément u de la file (l. 9) et on le « fusionne » avec son unique successeur v. Concrètement, on supprime u :

– de la liste des sommets (l. 11)
– de celle des prédécesseurs de v (l. 12)
– de celle des successeurs de w pour tout w prédécesseur de u (l. 14).

Si initialement un sommet w prédécesseur de u n'était pas prédécesseur de v, il le sera désormais (l. 16,17). Dans le cas contraire, son degré sortant diminue de 1, et si ce dernier atteint 1, w est ajouté à la file F (l. 19-22).

Proposition 5 *CONTRACTER-GRAPHE se termine et a une complexité en* $O(n^2)$ *dans le pire cas.*

Démonstration Le degré sortant de chaque sommet ne peut pas augmenter durant l'exécution de l'algorithme. Ainsi, chaque sommet est ajouté au plus une fois à la file F. Et puisqu'à chaque itération de la boucle **tant que** (l. 8) on consomme un élément de cette file, l'algorithme termine.

La boucle d'initialisation (l. 2) est exécutée n fois. A l'intérieur de cette boucle, le calcul du degré sortant initial est en $O(n)$, ENFILER est en $O(1)$. Au total, l'initialisation se fait en $O(n^2)$.

Puisque chaque sommet n'est ajouté qu'une seule fois à la file F, la boucle **tant que** (l. 8) est exécutée au plus n fois. La boucle intérieure **pour tout** (l. 13) est exécutée autant de fois que u a de prédécesseurs, c'est-à-dire au plus n fois. A l'intérieur de cette dernière, les différentes instructions AJOUTER, SUPPRIMER

Algorithme 2 CONTRACTER-GRAPHE(G)

Requiert: Un graphe $G = (V, \Gamma^-, \Gamma^+)$ fortement connexe
Fournit: Le graphe G contracté

```
1: F ← CRÉER-FILE()
2: pour tout u sommet de G faire
3:    clé(u) ← degré-sortant(u)
4:    si clé(u)=1 alors
5:       ENFILER(F, u)
6:    fin si
7: fin pour
8: tant que F non vide ∧ |V| ≥ 2 faire
9:    u ← DÉFILER(F)
10:   v ← l'unique élément de Γ+(u)
11:   SUPPRIMER(V, u)
12:   SUPPRIMER(Γ−(v), u)
13:   pour tout w ∈ Γ−(u) faire
14:      SUPPRIMER(Γ+(w), u)
15:      si w ∉ Γ−(v) alors
16:         AJOUTER(Γ−(v), w)
17:         AJOUTER(Γ+(w), v)
18:      sinon
19:         Décrémenter clé(w)
20:         si clé(w)=1 alors
21:            ENFILER(F, w)
22:         fin si
23:      fin si
24:   fin pour
25: fin tant que
```

et ENFILER se font en temps constant. Ainsi, l'algorithme a un temps d'exécution en $O(n^2)$. □

Algorithme 3 EST-1-MÉMOIRE-ÉVOLUÉ(G)

Requiert: Un graphe G fortement connexe
Fournit: **VRAI** si G est 1-mémoire, **FAUX** sinon
1: CONTRACTER-GRAPHE(G)
2: **si** nombre de sommets de $G \leq 1$ **alors**
3: **retourner VRAI**
4: **sinon**
5: **retourner FAUX**
6: **fin si**

Proposition 6 *EST-1-MÉMOIRE-ÉVOLUÉ se termine, est correct et a une complexité en* $O(n^2)$.

Démonstration La terminaison et la complexité de EST-1-MÉMOIRE-ÉVOLUÉ sont claires.

Un graphe ayant au plus un sommet est évidemment 1-mémoire. Il reste à prouver qu'un graphe fortement connexe contracté (donc tel que tous ses sommets a un degré sortant ≥ 2) ayant au moins deux sommets n'est pas 1-mémoire. Pour cela, voir la proposition 11. □

2.3 Caractérisation des graphes 2-mémoires

L'algorithme EST-2-MÉMOIRES-CONTRAINT présenté ci-dessous teste si on peut traiter un graphe avec deux mémoires, sous la contrainte que la première mémoire soit posée sur un sommet déterminé.

Proposition 7 *EST-2-MÉMOIRES-CONTRAINT se termine, est correct et a une complexité en* $O(n^3)$.

Démonstration Le graphe argument de EST-2-MÉMOIRES-CONTRAINT voit son nombre de sommets diminuer d'au moins 1 à chaque appel récursif, la terminaison en découle.

La décomposition de $G - u$ en composantes fortement connexes (l. 5) requiert un temps $O(n+m)$. L'appel de EST-1-MÉMOIRE sur ces composantes (l. 9) se fait en un temps $O(n^2)$ (voir la proposition 6), de même qu'une éventuelle contraction de C (l. 23). Soit $T(n)$ le temps d'exécution dans le pire cas de EST-2-MÉMOIRES-CONTRAINT sur un graphe à n sommets. On a alors $T(n) \leq Cn^2 + T(n-1)$, d'où $T(n) = O(n^3)$.

Examinons la validité de la réponse de EST-2-MÉMOIRES-CONTRAINT dans chaque cas (voir la figure 6) :

Algorithme 4 EST-2-MÉMOIRES-CONTRAINT(G, u)

Requiert: Un graphe G fortement connexe, un sommet $u \in G$
Fournit: **VRAI** si G est 2-mémoires, sous la contrainte que la première mémoire soit posée sur u, **FAUX** sinon

1: **si** nombre-de-sommets(G)≤ 1 **alors**
2: **retourner VRAI** {Cas 1}
3: **fin si**
4: poser une mémoire sur u
5: décomposer $G - u$ en un DAG de CFC
6: $c \leftarrow 0$
7: **tant que** $c \leq 1 \wedge$ il reste des CFC non encore choisies **faire**
8: choisir une CFC C non encore choisie
9: **si** EST-1-MÉMOIRE(C) = **FAUX alors**
10: incrémenter c
11: **fin si**
12: **fin tant que**
13: **si** $c = 0$ **alors**
14: **retourner VRAI** {Cas 2}
15: **fin si**
16: **si** $c > 1$ **alors**
17: **retourner FAUX** {Cas 3}
18: **fin si**
{il y a une unique CFC non 1-mémoire}
19: $C \leftarrow$ cette unique CFC non 1-mémoire
20: **si** degré entrant de C dans le DAG > 0 **alors**
21: **retourner FAUX** {Cas 4}
22: **fin si**
23: CONTRACTER-GRAPHE(C)
24: **si** $|\Gamma^+(u) \cap C| > 1$ **alors**
25: **retourner FAUX** {Cas 5}
26: **fin si**
{u pointe sur un unique sommet de C}
27: $v \leftarrow$ cet unique sommet de C sur lequel pointe u
28: **retourner** EST-2-MÉMOIRES-CONTRAINT(C, v) {Cas 6}

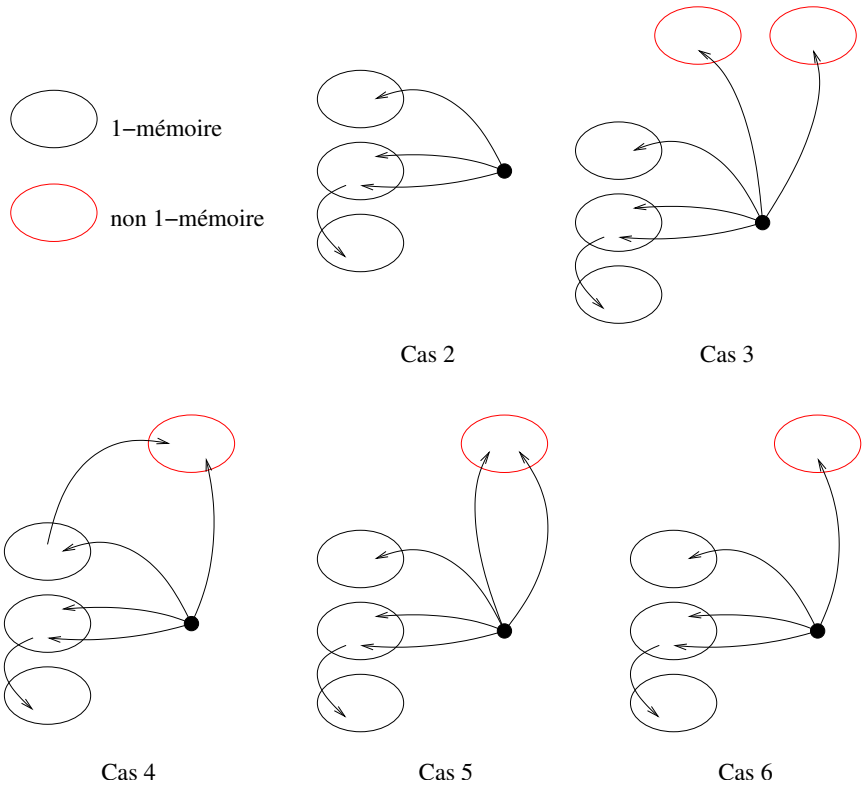

FIG. 6 – Différents cas possibles pour EST-2-MÉMOIRES-CONTRAINT

Cas 1 : Un graphe ayant au plus un sommet est bien sûr 1-mémoire.

Cas 2 : Nous sommes en présence d'un DAG de CFC 1-mémoire. Comme il nous reste une mémoire libre, nous pouvons les traiter tous dans l'ordre topologique (voir la proposition 2).

Cas 3 : Nous avons au moins deux CFC non 1-mémoire, par exemple C et D. On peut supposer sans perte de généralité que C est traitée *avant* D. La suite de la démonstration est identique au cas 4.

Cas 4 : Nous avons une seule CFC C non 1-mémoire mais celle-ci a un degré entrant non nul dans le DAG. Soit D une CFC qui pointe sur C. Par définition, on doit traiter C *avant* D. Ainsi, au moment où l'on traite C, D n'est pas encore traité, donc la première mémoire posée initialement sur u n'a pas encore été récupérée. On dispose donc d'une seule mémoire pour traiter C, ce qui est insuffisant.

Cas 5 : Nous avons une seule CFC C non 1-mémoire mais u pointe sur au moins deux sommets de C. L'idée essentielle est qu'on ne peut pas récupérer la mémoire posée sur u immédiatement, et donc on est contraint de commencer à travailler avec une unique mémoire. Il faut prouver qu'on ne peut pas faire grande chose, étant donné que C est non 1-mémoire contracté, mais par manque de temps, nous n'avons pas fini de rédiger cette preuve.

Cas 6 : Nous avons une seule CFC C non 1-mémoire, de degré entrant nul, et tel que u a un unique successeur $v \in C$ (il y en a au moins un car G est initialement fortement connexe). Si l'appel récursif renvoie **VRAI**, c'est ok. En effet, il suffit de traiter d'abord toutes les autres CFC avec une mémoire. Cela étant fait, on a récupéré cette mémoire, qu'on pose sur v. Comme v est l'unique successeur restant de u, on peut libérer la mémoire posée sur u, et traiter C avec deux mémoires dont une posée initialement sur v. Supposons maintenant que l'appel récursif revoie **FAUX**. Supposons par l'absurde qu'il existe une stratégie de traiter C. Dans cette stratégie, on doit commencer par poser la mémoire restante quelque part, disons w. Mais il est évident qu'on ne perd rien en la posant d'abord sur v pour récupérer la mémoire de u, et de poser ensuite cette dernière sur w. Puisque cette dernière stratégie ne fonctionne pas, la première ne peut pas fonctionner non plus, contradiction.

Comme les cas ci-dessus sont exhaustifs, on a bien montré la correction de EST-2-MÉMOIRES-CONTRAINT. □

Proposition 8 *EST-2-MÉMOIRES se termine, est correct et a une complexité en* $O(n^4)$.

Démonstration Immédiat d'après la proposition 7. □

Remarque La méthode utilisée ci-dessus ne peut pas s'appliquer telle quelle au cas 3-mémoires. La difficulté réside dans le cas où u pointe sur deux sommets v et w d'une CFC C non 2-mémoires. Dans ce cas, deux approches sont possibles :

Algorithme 5 EST-2-MÉMOIRES(G)

Requiert: Un graphe G fortement connexe
Fournit: **VRAI** si G est 2-mémoires, **FAUX** sinon

```
1: res ← FAUX
2: tant que res = FAUX ∧ il reste des sommets non encore choisis faire
3:     Choisir un sommet u non encore choisi
4:     res ← EST-2-MÉMOIRE-CONTRAINT(G, u)
5: fin tant que
6: retourner res
```

- ou bien on pose les deux mémoires restantes sur v et w et récupérer la mémoire posée sur u.
- ou bien on utilise les deux mémoires restantes pour traiter une partie de C, en espérant pouvoir récupérer la mémoire posée sur u à un moment ultérieur.

Contrairement au cas 6 de la proposition précédente, la deuxième approche peut fonctionner. Si la première approche peut se traiter par induction comme précédemment, cette deuxième approche est plus délicate, car le choix de u comme première mémoire ne nous impose pas un candidat pour la deuxième mémoire. Il faudrait ainsi tester tous les sommets, ce qui peut prendre un temps exponentiel.

Pour l'instant, nous n'avons pas encore surmonté cette difficulté qui nous empêche de franchir le palier $k = 3$ [3].

[3] Comme cela arrive assez souvent dans les problèmes combinatoires sur les graphes.

3 Encadrements de $K(G)$

À partir de maintenant, et sauf mention explicite du contraire, les graphes orientés considérés seront sans boucle (i.e. sans arc (u, u)). On peut transformer un graphe avec boucles en graphe sans boucle en remplaçant tout sommet u tel que $(u, u) \in A$ par deux sommets u_1, u_2 qui prennent en charge respectivement les prédécesseurs et les successeurs de u, et qui sont reliés entre eux (voir la figure 7). Cette transformation est bien valide en vertu de la proposition 4.

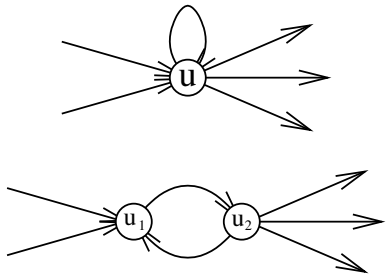

FIG. 7 – Élimination d'une boucle

3.1 Majorations triviales

Considérons un graphe $G = (V, A)$ avec $|V| = n$. Remarquons que, trivialement, $n - 1$ est un majorant de $K(G)$. Un majorant un peu moins évident est fourni par la proposition suivante :

Proposition 9 *Soit un graphe $G = (V, A)$. Un Feedback Vertex Set (FVS) de G est un sous-ensemble U de V tel que tout cycle de G contient au moins un élément de U (autrement dit : un sous-ensemble U de V tel que le graphe induit sur G par $V - U$ est acyclique). Un Minimum Feedback Vertex Set (MFVS) est un FVS de cardinal minimal. Alors $K(G)$ est inférieur ou égal au cardinal d'un MFVS.*

Démonstration La mise en mémoire des sommets d'un FVS brise tous les cycles de G. Le graphe résultant est acyclique, donc 0-mémoire (voir la proposition 1). □

Remarque Le problème de trouver le cardinal d'un MFVS a été démontré NP-complet [10]. De plus, ce majorant est assez grossier car on ne tient pas compte du fait que les mémoires peuvent être réutilisées.

3.2 Minorations de $K(G)$

Proposition 10 *Soit G un graphe orienté. Notons*

$$M(G,i) = \min_{\substack{U \subset V \\ |U|=i}} |\delta(U)|$$

$$M(G) = \max_{1 \leq i \leq n} M(G,i)$$

Alors $K(G) \geq M(G)$.

Démonstration Voir la section 4.1.

Nous sommes maintenant en mesure d'achever la preuve de la proposition 6 :

Proposition 11 *Un graphe fortement connexe (comportant éventuellement des boucles) ayant au moins deux sommets et tel que tout sommet a un degré sortant* ≥ 2 *n'est pas 1-mémoire.*

Démonstration Supposons par l'absurde qu'il existe un graphe $G = (V, E)$ fortement connexe ayant au moins deux sommets, tel que tout sommet a un degré sortant ≥ 2 et qui est 1-mémoire. Considérons une stratégie permettant de supprimer tous les sommets de G en utilisant au plus une mémoire. Considérons le premier sommet u qu'on supprime. Il ne peut avoir plus d'un successeur en vertu de la proposition 10, et ce successeur v (qui existe par connexité forte et parce que le graphe a au moins 2 sommets) est marqué au moment de cette suppression. Comme de plus u est de degré sortant 2, il possède une boucle. Mais alors il doit aussi être marqué au moment de sa suppression, contradiction.

Le nombre $M(G)$ a fait objet de nombreuses études dans la littérature scientifique, notamment dans le cas des graphes produits. Nous donnons ici, pour l'exemple, un résultat concernant les grilles orientées symétriques :

Théorème 1 (Chvátalová [5]) *Soit* $G = (V, E)$ *un graphe non orienté tel que* $V = \{1 \dots m\} \times \{1 \dots n\}$, *avec* $\{(i,j),(r,s)\} \in E$ *ssi* $(i = r \wedge |j - s| = 1) \vee (j = s \wedge |i - r| = 1)$ *(de manière plus intuitive, une « grille » non orientée* $m \times n$*). Alors* $M(G) \geq \min(m, n)$.

Corollaire 1 *Soit* G *une grille orientée symétrique* $m \times n$ *(voir la figure 8). On a* $K(G) \geq \min(m, n)$.

D'autres minorations sont possibles en considérant des *inégalités isopérimétriques discrètes* [4]. Pour un graphe G et un sous-ensemble U des sommets de ce graphe, une telle inégalité donne une minoration de $|\delta(U)|$ en fonction de $|U|$ et des paramètres du graphe [4]. En faisant varier $|U|$ et en calculant le maximum

[4] On peut y voir une analogie avec le cas continu où, par exemple, le bord est remplacé par le périmètre et le cardinal de l'ensemble par la surface.

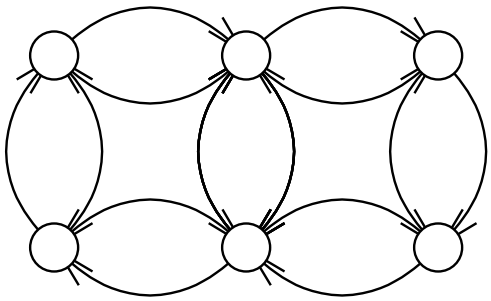

FIG. 8 – Une grille orientée symétrique 3×2

des minorants ainsi obtenus, on obtient un minorant de $K(G)$ comme le montre la proposition 10. Ce calcul devrait être faisable pour des graphes assez simples, mais nous ne l'avons pas tenté par manque de temps.

3.3 Majorations de $K(G)$

Soit $G = (V, A)$ un graphe orienté. Considérons le graphe $G_u = (V, E)$ non orienté tel que $\{u, v\} \in E$ ssi $(u, v) \in A$ ou $(v, u) \in A$.

Les règles du jeu pour les graphes non orientés sont :

- on peut à tout moment marquer un sommet non encore marqué (mise en mémoire temporaire).
- on peut supprimer un sommet s'il n'a aucun voisin ou si tous ses voisins sont marqués. Si ce sommet était initialement marqué, cela correspond à la libération d'une place en mémoire temporaire.

On définit aussi pour un graphe non orienté G_u le nombre minimal $K(G_u)$ de mémoires nécessaires pour traiter G_u.

On voit bien que $K(G) \leq K(G_u)$. Une majoration de $K(G_u)$ nous donnera alors immédiatement une majoration du $K(G)$ correspondant. Dans la suite de ce paragraphe, les graphes G considérés seront donc des graphes non orientés.

Pour supprimer les sommets, on peut procéder ainsi : on trouve un ensemble de sommets C qui « sépare » le graphe en deux parties non connectées entre elles. On marque les sommets de C, divisant G ainsi en deux parties « indépendantes », puis on traite séparément ces deux parties, en récupérant les mémoires utilisées par la première avant de traiter la seconde (voir la figure 9). Tout le problème est donc de trouver un ensemble de petit cardinal qui sépare le graphe en deux parties « équilibrées », de telle sorte qu'on puisse appliquer la récursion de manière efficace. Pour cela, nous allons utiliser quelques célèbres théorèmes de séparation :

Théorème 2 (Lipton, Tarjan [9]) *Soit $G = (V, E)$ un graphe* planaire *à n sommets. Alors il existe une partition de V en trois sous-ensembles A, B, C telle que :*

– *aucune arête ne joint un sommet de* A *à sommet de* B.
– $|A| \leq 2n/3$, $|B| \leq 2n/3$, $|C| \leq 2\sqrt{2}\sqrt{n}$.
De plus, une telle partition peut être calculée en un temps $O(n)$.

Corollaire 2 *Soit* $G = (V, E)$ *un graphe* planaire *à* n *sommets. Alors* $K(G) \leq 2\sqrt{2}\alpha\sqrt{n}$ *(*$\simeq 15,4\sqrt{n}$*), où* $\alpha = \frac{\sqrt{3}}{\sqrt{3}-\sqrt{2}}$. *De plus, on peut calculer en un temps* $O(n^{1,71})$ *une séquence de suppressions utilisant au plus* $2\sqrt{2}\alpha\sqrt{n}$ *mémoires.*

Démonstration Démontrons par induction sur n que pour tout graphe planaire G à n sommets, $K(G) \leq 2\sqrt{2}\alpha\sqrt{n}$.

Pour $n = 1$, on peut bien sûr traiter un graphe à un seul sommet avec $0 \leq 2\sqrt{2}\alpha$ mémoire.

Supposons pour un certain $n \geq 2$ que la propriété énoncée est vraie pour tout $1 < i < n$. Soit un G graphe planaire à n sommets. Soit A, B, C la partition donnée par le Théorème 2. Posons des mémoires sur tous les sommets de C (voir la figure 9). Considérons les graphes G_A et G_B induits sur G par A et B respectivement. G_A est évidemment planaire et ses sommets ne dépendent d'aucun sommet extérieur (car par construction aucune arête ne joint un sommet de A à un sommet de B, et tous les sommets de C sont marqués). Par hypothèse d'induction, on peut traiter G_A avec $K(G_A) \leq 2\sqrt{2}\alpha\sqrt{2n/3}$ mémoires. Une fois le traitement de G_A fini, on aura récupéré toutes ces $K(G_A)$ mémoires, qu'on peut réutiliser pour traiter G_B. Après cela, il ne reste plus que les sommets de C, qu'on peut supprimer sans utiliser de mémoire supplémentaire.

Finalement, on a utilisé moins de

$$|C| + \max(K(G_A), K(G_B)) \leq 2\sqrt{2}\sqrt{n} + 2\sqrt{2}\alpha\sqrt{2n/3} = 2\sqrt{2}\alpha\sqrt{n} \quad \text{mémoires.}$$

L'algorithme pour trouver une séquence de suppression utilisant moins de $2\sqrt{2}\alpha\sqrt{n}$ mémoires se déduit directement de la démonstration. Soit $T(n)$ son temps d'exécution à l'ordre n. On a $T(n) = 2 \times T(2n/3) + O(n)$, d'où $T(n) = O(n^{\lg_{3/2} 2}) = O(n^{1,71})$. □

Graphes à mineur exclu : On dit qu'un graphe H est *mineur* d'un graphe G si on peut obtenir H à partir de G en effectuant une séquence d'opérations suivantes :
– suppression d'un sommet
– suppression d'une arête
– fusion de deux sommets adjacents

Les graphes planaires peuvent être vus comme des cas particuliers de graphes sans mineur fixé. En effet, le théorème Kuratowski-Wagner affirme que les graphes planaires sont exactement les graphes qui n'admettent pas K_5 et $K_{3,3}$ comme mineurs [3].

Théorème 3 (Alon, Seymour, Thomas [1]) *Soient* H *et* $G = (V, E)$ *deux graphes à* h *et* n *sommets respectivement, et tels que* G *est sans* H*-mineur. Alors il existe une partition de* V *en trois sous-ensembles* A, B, C *telle que :*

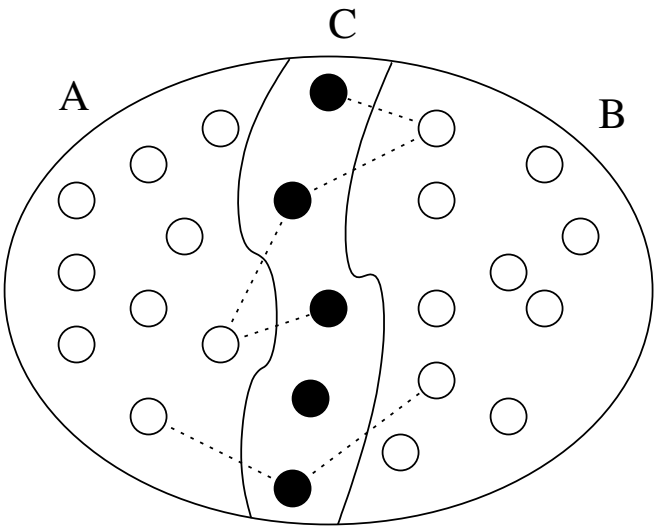

FIG. 9 – Séparation d'un graphe

– *aucune arête ne joint un sommet de* A *à sommet de* B.
– $|A| \leq 2n/3$, $|B| \leq 2n/3$, $|C| \leq h^{3/2}\sqrt{n}$.

De plus, une telle partition peut être calculée en un temps $O(\sqrt{h}\sqrt{n}m')$ *où* $m' = |V| + |E|$.

Corollaire 3 *Soient* H *et* $G = (V, E)$ *deux graphes à* h *et* n *sommets respectivement, et tels que* G *est sans* H*-mineur. Alors* $K(G) \leq \alpha h^{3/2}\sqrt{n}$, *où* $\alpha = \frac{\sqrt{3}}{\sqrt{3}-\sqrt{2}}$. *De plus, on peut calculer en un temps* $O(\sqrt{h}n^{5/2})$ *une séquence de suppressions utilisant au plus* $\alpha h^{3/2}\sqrt{n}$ *mémoires.*

Démonstration La démonstration est essentiellement la même que précédent : on utilise $h^{3/2}\sqrt{n}$ mémoires pour séparer G en deux sous-graphes de taille $\leq 2n/3$, qu'on peut traiter par hypothèse d'induction avec $\alpha h^{3/2}\sqrt{2n/3}$ mémoires (pour que la récurrence puisse fonctionner, il faut remarquer en outre que si G est sans H-mineur, tout sous-graphe de G est sans H-mineur). Au total, on a utilisé moins de

$$h^{3/2}\sqrt{n} + \alpha h^{3/2}\sqrt{2n/3} = \alpha h^{3/2}\sqrt{n} \quad \text{mémoires.}$$

Pour la complexité, remarquons que $m' = O(n^2)$. Donc si $T(n)$ dénote le temps d'exécution de l'algorithme à l'ordre n, on a $T(n) = 2 \times T(2n/3) + O(\sqrt{h}n^{5/2})$, d'où $T(n) = O(\sqrt{h}n^{5/2})$. □

4 NP-complétude du problème

4.1 L'idée principale et ses implications

Condidérons les règles suivantes :

- on peut à tout moment marquer un sommet non encore marqué (mais on ne supprime pas les arcs entrant dans ce sommet).
- on peut supprimer un sommet s'il n'a aucun successeur ou *si tous ses successeurs sont marqués*.

Il est facile de voir que ces règles sont équivalentes aux règles énoncées à la section 1.2. Cette approche « duale » s'avère pourtant plus féconde car elle garde la trace des dépendances vis-à-vis des sommets marqués.

En effet, considérons l'ensemble U des sommets déjà supprimés à un moment donné du jeu. La règle 2 implique alors que tous les sommets de $\delta(U)$ doivent être marqués à ce moment (voir figure 10). Nous pouvons démontrer à présent la proposition 10 :

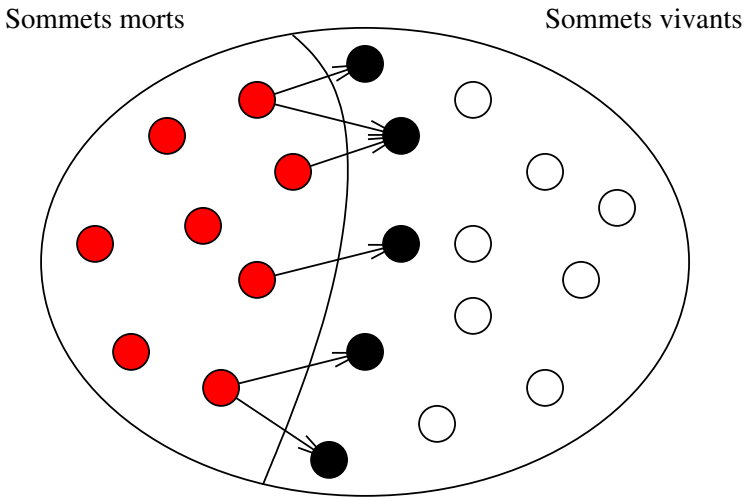

FIG. 10 – Le front d'élimination à un instant donné

Démonstration de la proposition 10 Soit $i \in \{1 \dots n\}$. Supposons qu'on a pu supprimer tous les sommets de G à l'aide de k mémoires. Considérons U_i l'ensemble des i premiers sommets supprimés. Comme on l'a remarqué, tous les sommets de $\delta(U_i)$ doivent être marqués à ce moment-là du jeu. D'où

$$k \geq |\delta(U_i)| \geq \min_{\substack{W \subset V \\ |W|=i}} |\delta(W)|$$

Ceci étant vrai pour tout $i \in \{1 \dots n\}$, on a bien montré l'inégalité énoncée. □

Proposition 12 *Soit $G = (V, A)$ un graphe orienté. Soit f un* arrangement *de V (i.e. une fonction $V \to \{1 \dots n\}$). Notons*

$$\phi(G, f) = \max_{1 \leq i \leq n-1} |\delta(\{f^{-1}(1) \dots f^{-1}(i)\})|$$

$$\phi(G) = \min_{f \ arrangement} \phi(G, f)$$

Alors $\phi(G) \leq K(G) \leq \phi(G) + 1$.

Démonstration Pour démontrer la première inégalité, considérons un arrangement f telle qu'en supprimant les sommets de G dans l'ordre suggéré par f, on ait besoin d'au plus $K(G)$ mémoires. Pour $i \in \{1 \dots n\}$, considérons le moment du jeu où l'on a supprimé $U_i = \{f^{-1}(1) \dots f^{-1}(i)\}$. Comme on l'a remarqué, à ce moment là, les sommets de $\delta(U_i)$ sont nécessairement marqués, d'où $K(G) \geq |\delta(U_i)|$. Ceci étant vrai pour i quelconque dans $\{1 \dots n\}$, on a bien montré que $K(G) \geq \phi(G, f)$. Puisque $\phi(G, f) \geq \phi(G)$, on obtient bien $K(G) \geq \phi(G)$.

Pour démontrer la deuxième inégalité, considérons un arrangement f tel que $\phi(G, f) = \phi(G)$. Pour alléger l'écriture, notons $u_1 = f^{-1}(1) \dots u_n = f^{-1}(n)$. Montrons par récurrence sur $i \in \{1 \dots n\}$ qu'en supprimant $u_1 \dots u_i$, dans cet ordre, on a besoin d'au plus $\phi(G) + 1$ mémoires.

Pour $i = 1$, il suffit de marquer les sommets de $\delta(\{u_1\})$ pour pouvoir supprimer u_1. Il y a au plus $\phi(G)$ de tels sommets.

Supposons pour un certain $i \in \{1 \dots n-1\}$ qu'on a déjà supprimé $U_i = \{u_1 \dots u_i\}$. Ceci implique qu'on a récupéré toutes les mémoires précédemment posées sur les sommets appartenant à U_i et que les seules mémoires utilisées sont posées sur $\delta(U_i)$. Deux cas possibles (voir la figure 11) :

Cas 1 : $u_{i+1} \notin \delta(U_i)$, alors $\delta(U_i) \subset \delta(U_{i+1})$. Puisque $|\delta(U_{i+1})| \leq \phi(G)$, il suffit de poser des mémoires supplémentaires sur $\delta(U_{i+1}) - \delta(U_i)$, sans jamais dépasser $\phi(G)$ (donc *a fortiori* $\phi(G)+1$) mémoires utilisées simultanément. Cela permet de supprimer ensuite u_{i+1}.

Cas 2 : $u_{i+1} \in \delta(U_i)$, alors $(\delta(U_i) - \{u_{i+1}\}) \subset \delta(U_{i+1})$. De la même façon que précédemment, on pose des mémoires supplémentaires sur $\delta(U_{i+1}) - \delta(U_i)$, et on peut supprimer ensuite u_{i+1}. Remarquons que, contrairement au cas précédent, juste avant de supprimer u_{i+1}, on était entrain d'utiliser $|\delta(U_{i+1})| + 1$ mémoires, mais cela reste inférieur à $\phi(G) + 1$.

Puisque $\phi(G)+1$ mémoires sont suffisantes pour traiter G, on a bien $K(G) \leq \phi(G) + 1$. □

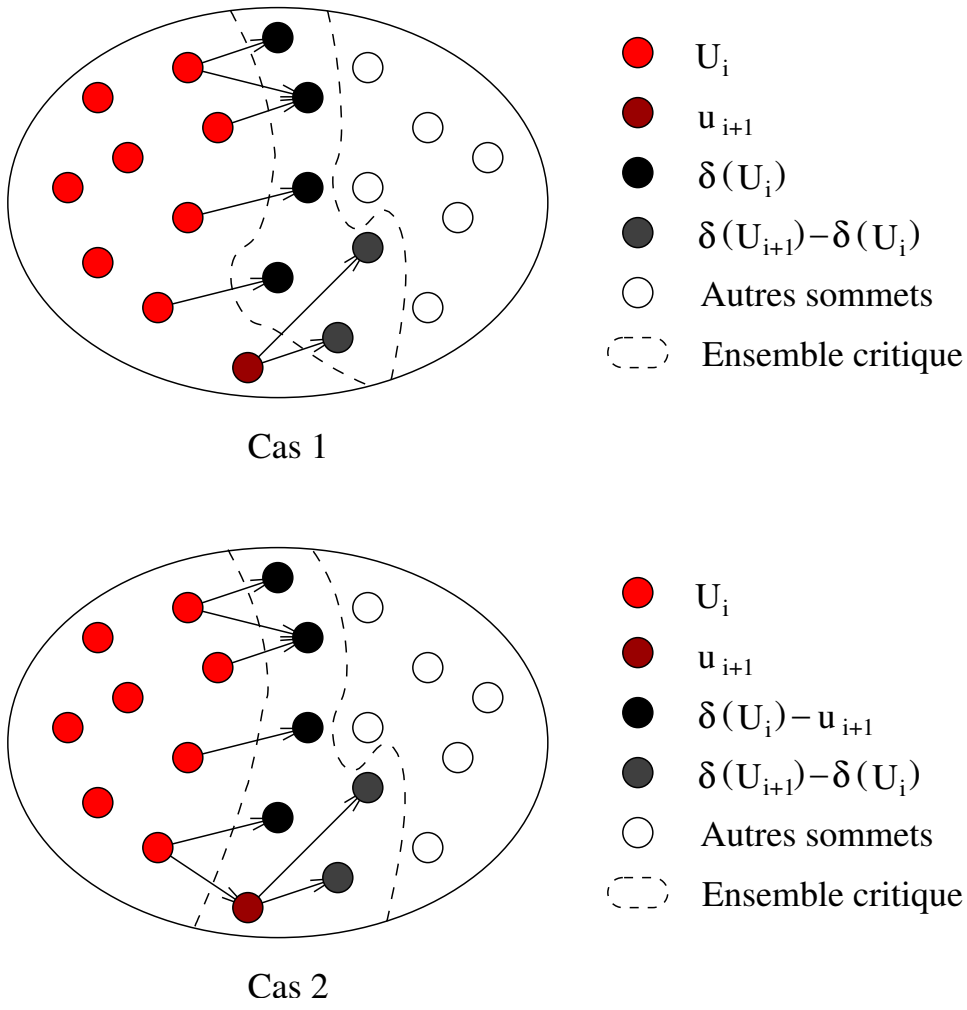

FIG. 11 – Deux cas possibles quand on passe de i à $i+1$

4.2 Démonstration de la NP-complétude

Formellement, considérons le problème MINIMUM-TEMPORARY-MEMORY-SIZE[5] (MTMS) qui se pose en ces termes : « étant donné un graphe orienté G, calculer $K(G)$ ». Considérons aussi sa version décisionnelle, MTMSD : « étant donné un graphe orienté G et un entier k, décider si $K(G) \leq k$ ». Il est clair que ces deux problèmes sont polynomialement équivalents (en la taille du graphe *et* en la taille de k en ce qui concerne MTMSD). Remarquons par ailleurs que MTMSD (donc MTMS) sont dans NP. En effet, on peut facilement vérifier en temps polynomial si $K(G) \leq k$, étant donné un ordre de suppression des sommets.

[5]Ce nom barbare est temporaire. Il est utilisé ici pour des raisons de clarté, en attendant de trouver mieux.

La proposition 12 met en évidence le lien avec la classe des problèmes d'arrangement très étudiés dans la littérature scientifique [6], notamment pour leurs applications à la conception des réseaux électroniques.

Pour un graphe *non orienté* $G = (V, E)$ et un arrangement f de V, notons (voir la figure 12) :

$$\hat{\phi}(G, f) = \max_{1 \leq p \leq n-1} |\{i > p : \exists j \leq p, \{f^{-1}(i), f^{-1}(j)\} \in E\}|$$

$$\hat{\phi}(G) = \min_{f \text{ arrangement}} \hat{\phi}(G, f)$$

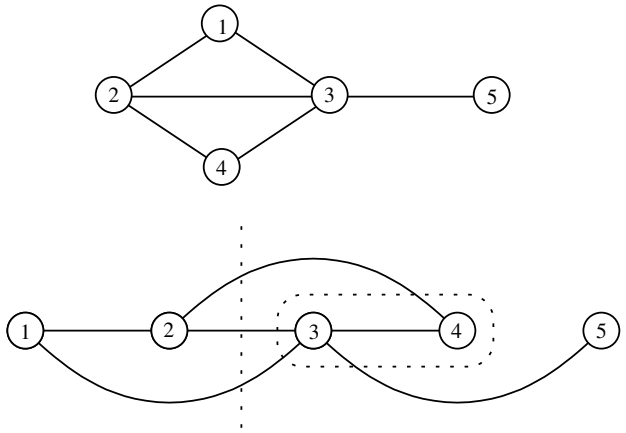

FIG. 12 – Un arrangement d'un graphe et la séparation correspondant

Le problème VERTEX-SEPARATION se pose en ces termes : « étant donné un graphe non-orienté G, calculer $\hat{\phi}(G)$ ».

Théorème 4 (Lengauer [8]) *VERTEX-SEPARATION est un problème NP-complet.*

Remarque Pour démontrer la NP-complétude de MTMS, nous avons besoin d'une version plus forte du théorème précédent. Plus précisément, nous demandons que le problème VERTEX-SEPARATION-APX-1 : « étant donné G, trouver un k tel que $k - 1 \leq \hat{\phi}(G) \leq k$ » soit également NP-complet [7].

Théorème 5 *MTMS est un problème NP-complet.*

[6] Voir [7] pour un panorama complet.

[7] La meilleure approximation connue de VERTEX-SEPARATION est à un facteur multiplicatif en $O(log^2 n)$ [7]. Il serait très surprenant qu'il existe une approximation à un facteur additif constant, mais, pour l'instant, nous n'avons pas trouvé de preuve du contraire.

Démonstration Soit $G = (V, E)$ un graphe *non orienté*, instance de VERTEX-SEPARATION-APX-1. Considérons le graphe *orienté* $G_d = (V, A)$ où chaque arête $\{u, v\} \in E$ est remplacée par deux arcs $(u, v), (v, u) \in A$. Il est facile de voir que, d'une part, la transformation est polynomiale en la taille de G, et d'autre part, pour tout f arrangement de V, on a $\hat{\phi}(G, f) = \phi(G_d, f)$, et donc $\hat{\phi}(G) = \phi(G_d)$. On a ainsi $K(G_d) - 1 \leq \hat{\phi}(G) \leq K(G_d)$ par la proposition 12. □

Remarque La réduction précédente peut se faire aussi dans l'autre sens : étant donné un graphe orienté symétrique [8] G (i.e. où pour tout arc (u, v) existe aussi un arc (v, u)), on peut construire un graphe non orienté G_u comme dans la section 3.3. On a alors $\hat{\phi}(G_u) \leq K(G) \leq \hat{\phi}(G_u) + 1$. Ainsi, tout résultat permettant de calculer ou d'approximer la VERTEX-SEPARATION peut être utilisé pour approximer la MTMS. Notamment, Bodlaender [2] a réussi à calculer en temps linéaire la VERTEX-SEPARATION des graphes à tree-width borné, ce qui montre qu'approximer MTMS à 1 près est linéaire en la taille de G [9].

[8] Si le graphe n'est pas symétrique, on obtiendra seulement une majoration de $K(G)$. Ces résultats rendraient probablement obsolète la section 3.3.

[9] Cependant, les constantes dépendantes de k sont si grandes que de tels algorithmes sont impraticables pour $k \geq 3, 4$.

5 Conclusion

Malgré quelques progrès dans la compréhension du problèmes réalisés pendant ce stage, il nous reste encore beaucoup à faire. Voici des directions de recherche que nous envisageons :

Section 2 : Trouver des algorithmes plus efficaces (linéaires ?) pour caractériser les graphes 1,2-mémoires. Trouver des algorithmes polynomiaux pour caractériser les graphes k-mémoires, $k \geq 3$.

Section 3 : Comme nous l'avons remarqué dans la section 4.2 précédente, les travaux sur VERTEX-SEPARATION rendraient probablement obsolètes les résultats de cette section. Il faut voir cela de plus près. On peut aussi étudier les inégalités isopérimétriques pour obtenir d'autres minorations.

Section 4 : Le rapprochement avec les problèmes d'arrangement ayant été fait seulement vers la fin du stage, il y a encore énormément à creuser dans cette direction. Tout d'abord, il faut raffiner les réductions pour s'affranchir de l'approximation « à 1 près ». Alternativement, nous pourrions aussi prouver (ou trouver la preuve dans la littérature) la NP-complétude de VERTEX-SEPARATION-APX-1. La NP-complétude (ou non) de quelques sous-classes (graphes à tree-width borné, sous-graphes de la grille, etc.) reste aussi à étudier.

Pebbling : Différents problèmes de « pebbling » ont déjà fait objet de recherches dans les années 70 (Search Problem, Vertex Search Problem, Black/White Pebbles Problem, etc.). Il serait intéressant de comparer ces problèmes au nôtre.

Finalement, je voudrais remercier l'équipe Mascotte pour l'accueil chaleureux qu'il m'a réservé, et en particulier, David Coudert et Stéphane Pérennes pour l'encadrement patient et de haute qualité qu'ils m'ont accordé.

Références

[1] Noga Alon, Paul Seymour, Robin Thomas, *A Separator Theorem for Graphs with an Excluded Minor and its Applications*, Proc. 22nd ACM Symposium on the Theory of Computing STOC'90, pp. 293-299, 1990.

[2] Hans Bodlaender, *A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth*, SIAM J. Comp., 25, pp. 1305-1317, 1996.

[3] John Bondy, Uppaluri Murty, *Graph Theory with Applications*, chapitre 9, The Macmillan Press, London and Basingstoke, 1978.

[4] Fan Chung Graham, *Discrete isoperimetric inequalities*, http ://www.math.ucsd.edu/∼fan/, 2004.

[5] Jarmila Chvátalová, *Optimal Labelling of a Product of Two Paths*, Discr. Math., 11, pp. 249-253, 1975.

[6] Thomas Cormen, Charles Leiserson, Ronald Rivest, *Introduction à l'Algorithmique*, Dunod, Paris, 1994.

[7] Josep Díaz, Jordi Petit, Maria Serna, *A Survey of Graph Layout Problems*, ACM Computing Surveys (CSUR), 34, pp. 313-356, 2002.

[8] Thomas Lengauer, *Black-White Pebbles and Graph Separation*, Acta Informatica, 16, pp. 465-475, 1981.

[9] Richard Lipton, Robert Tarjan, *A Separator Theorem for Planar Graphs*, SIAM J. Appl. Math., 36, pp. 177-189, 1979.

[10] Michael Garey, David Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness*, Freeman, 1979.