Version bottleneck des problèmes d'ordonnancement classique

Jean-Alexandre Anglès d'Auriac

10 avril 2009

Table des matières

De nombreux problèmes d'ordonnancement cherchent à minimiser ou maximiser le poids de structures de données comme des arbres, des chemins, des cycles, des ensembles de sommets, ... Le poids d'une tel structure est alors naturellement défini par la somme des poids de ses composantes (en général, sommets ou arrêtes). Cependant, il est parfois plus intéressant de définir le poids comme étant égal au poids du plus petit (ou du plus grand) composant. Pour donner un exemple assez parlant, si on modélise un réseau par un graphe, où les poids des différentes arrêtes correspondent au flux maximal qu'elles peuvent laisser passer, alors on souhaiterait que le poids d'un chemin corresponde aussi au flux maximal qu'il peut laisser passer. Le débit du chemin est limité par son goulot d'étranglement, il est donc le même que celui du moins rapide des liens qui le composent. Toujours dans ce contexte de lien dans un réseau, on pourra rechercher le chemin le plus rapide, donc de celui de plus grand poids, entre deux noeuds.

Nous allons donc étudier quel est l'effet de la substitution de ce nouveau type de fonction de poids sur les problèmes d'optimisation classique, comme la recherche d'un plus court chemin, celle d'un arbre couvrant de poids minimum, celle d'une couverture par les sommets de poids minimal, celle d'une clique de poids minimal ...

Nous verrons tout d'abord ce qu'il advient de problèmes dont la version classique est déjà polynomiale, puis nous chercherons à exhiber des exemples de problèmes dont la version classique est NP-complète, mais dont la version modifiée est polynomial. Puis nous verrons un cas où même la version modifiée est NP-complète. Pour finir, nous nous interrogerons sur la possibilité de trouver un exemple de problème dont la version classique est polynomiale, mais dont la version modifiée est NP-complète.

1 Problèmes dont la version classique est polynomiale

1.1 Recherche d'un plus court chemin

On peut facilement adapter la plupart des algorithmes de recherche de plus court chemin classique à la version modifiée du problème. Voici un exemple, avec l'algorithme de Dijkstra :

```
plus_court_chemin_bottleneck(V,E,s,t) :
/* plus_court_chemin(V,E,s,t) calcule le plus court chemin du sommet
s vers le sommet t dans le graphe (V,E)
A est l'ensemble des sommets dont on a déjà calculé le plus court chemin
B est celui de ceux dont on ne l'a pas calculé
le tableau suivant contient pour chaque sommet le sommet suivant dans le plus
court chemin qui les relie à t */
A := {t}
B := V\t
while (s not in B) do
```

```
        forall sommet in B do

            forall (depart,sommet) in V with depart in A do


                if distance(sommet)> poids(arrete)+distance(depart)


                    distance(sommet) := poids(arrete)+distance(depart)
                    suivant(sommet) := depart
        forall sommet in B do

            if sommet_min > sommet do sommet_min := sommet
        A := A ∪ sommet_min
        B := V\sommet_min
    return suivant
```

devient

```
    plus_court_chemin_bottleneck(V,E,s,t) :
    /* plus_court_chemin(V,E,s,t) calcule le plus court chemin du sommet
    s vers le sommet t dans le graphe (V,E)
    A est l'ensemble des sommets dont on a déjà calculé le plus court chemin
    B est celui de ceux dont on ne l'a pas calculé
    le tableau suivant contient pour chaque sommet le sommet suivant dans le plus
    court chemin qui les relie à t */
    A := {t}
    B := V\t
    while (s not in B) do
        forall sommet in B do

            forall (depart,sommet) in V with depart in A do


                if distance(sommet)> max(poids(arrete),distance(depart))


                    distance(sommet) := max(poids(arrete),distance(depart))
                    suivant(sommet) := depart
        forall sommet in B do

            if sommet_min > sommet do sommet_min := sommet
        A := A ∪ sommet_min
        B := V\sommet_min
```

```
return suivant
```

De même, de nombreux autres algorithmes peuvent être adaptés simplement en remplaçant la formule pour obtenir le plus court chemin vers un sommet de la même façon que dans l'exemple.

La preuve de l'algorithme ci-dessus s'obtient très facilement à partir de la preuve d'un Dijkstra classique.

L'adaptation des preuves depuis l'algorithme du cas classique vers celui du cas Bottleneck fait bien ressortir deux propriétés qui rendent cette adaptation possible :

- Le poids d'un chemin d'un sommet 1 à un sommet n passant par les sommets 2, 3, ..., $n-1$ doit pouvoir être calculé simplement en fonction du poids du chemin 1, 2, 3, ..., $n-1$ et du poids de l'arrête $(n-1,n)$
- Le poids d'un chemin doit augmenter (pas nécessairement strictement, ce n'est pas le cas avec la fonction de poids Bottleneck) lors de l'ajout d'une arrête supplémentaire.

On pourrait, de la même manière, adapter Dijkstra à toute fonction de poids respectant les deux propriétés ci-dessus.

1.2 Arbre couvrant de poids minimal

L'algorithme de Kruskal permet d'exhiber un arbre couvrant de poids classique minimal :

```
kruskal(V,E) :
/* kruskal(V,E) calcule un arbre couvrant de poids minimal dans le graphe (V,E)
A est l'ensemble des sommets dont on n'a pas encore décidé s'il seront inclus
dans l'arbre couvrant.
arretes_de_l_arbre est celui des arretes dont on sait qu'elles seront incluse
dans l'arbre.
sans_cycle(arretes) est une fonction qui répond vrai si le graphe (V,arretes)
est sans cycle, et non sinon */
A := {V}
arretes_de_l_arbre := {}
while (A != {}) do
    arrete_min := arrete_de_poids_min(A)
    if (sans_cycle(arretes_de_l_arbre∪arretemin))
         arretes_de_l_arbre := arretes_de_l_arbre∪arretemin
    A := A \ arrete_min
return arretes_de_l_arbre
```

Prouvons que cet algorithme marche aussi pour la fonction de poids modifiée.

En effet, l'algorithme crée manifestement un arbre couvrant (aucun cycle n'est créé, par construction, et si le graphe est connexe, pour tout sommet, au moins une des arrêtes dont il est l'extrémité est ajoutée), et nous allons montrer qu'il est bien de poids minimal. Si l'on considère une solution optimale, soit arrete_max l'arrête de poids maximal. Montrons que lorsque toutes les arrêtes

d'un poids inférieur à celui de arrete_max ont été considérée par l'algorithme, celui-ci a déjà trouvé l'arbre couvrant.

Les arrêtes de l'arbre couvrant optimal relient tous les sommets en une composante connexe. Hors, après avoir considéré une arrête de l'arbre couvrant optimal par l'algorithme, on sait que ses deux extrémités sont reliées : soit l'arrête a été ajoutée à l'arbre en construction, soit elle aurait crée un cycle, c'est à dire que les deux extrémités en question faisaient déjà partie de la même composante connexe.

Ainsi, on a bien un arbre couvrant lorsque l'algorithme considère l'arrête arrete_max. Ainsi, toutes les arrêtes de l'arbre en question sont de poids inférieur ou égal à arrete_max. L'arbre est donc de poids inférieur ou égal à l'arbre de poids minimal : c'est donc bien un arbre couvrant de poids minimal.

1.3 Couverture par les sommets de poids minimal

Contrairement à la version classique, pour la version modifiée, il peux exister une solution qui ne soit pas minimale en nombre de sommet : en effet, si on trouve une solution minimale de poids m, et qu'il existe des sommets de poids inférieur à m qui ne font pas partie de la couverture, on peut les y rajouter, pour obtenir un autre couverture, sans augmenter le poids de cet nouvelle couverture ainsi obtenue.

Ainsi, on peut en déduire l'algorithme suivant :

```
couverture_par_sommet(V,E) :
/* couverture_par_sommet(V,E) calcule une couverture par sommets de poids
minimal dans le graphe (V,E)
A est l'ensemble des sommets choisi pour faire partie de la couverture */
A := {}
forall (x,y) in E do
    if poids(x)>poids(y)
       then A := A ∪ {y}
       else A := A ∪ {x}
return A
```

On vérifie facilement qu'on obtient ce faisant une couverture, et qu'elle est de poids minimal :

- Pour chaque arrête, on a ajouté au moins une de ces deux extrémités lorsque on l'a considérée dans la boucle principale. Chaque arrête est donc bien couverte. Il s'agit donc bien d'une couverture par les sommets.
- Pour obtenir une couverture par les arrêtes, il faut nécessairement qu'une des deux extrémités de chaque arrête soit incluse dans la couverture. Soit B une couverture minimale. Considérons chacune des arrêtes : si son extrémités la plus légère n'est pas dans la couverture, on peux l'y ajouter sans augmenter le poids total de la couverture. On obtiens ainsi une couverture qui inclus tous les sommets de A, et de poids minimal. Hors, en enlevant des sommets à la couverture, on ne peux que diminuer le poids total. Donc, A est aussi de poids minimal.

2 Problèmes dont la version classique est NP-complète

2.1 Problèmes résolubles en temps polynomial

2.1.1 Clique de poids maximum

La recherche d'une clique de poids maximum est un «exemple-jouet» du fait qu'un problème dont la version classique est NP-complète puisse devenir polynomial, et même linéaire, dans sa version modifiée. En effet, avec la nouvelle définition du poids, seule l'arête la plus lourde de la clique compte. Il suffit donc de trouver l'arête de plus grand poids dans la graphe : elle constitue à elle seule une clique de taille 2, et aucune autre clique ne peux avoir un poids supérieur.

2.2 Problèmes NP-complets

2.2.1 Trouver le cycle hamiltonien de poids minimum dans une clique

La recherche d'un cycle hamiltonien de poids minimum dans une clique (connu sous le nom de «problème du voyageur de commerce») est un problème NP-complet. Ce problème reste NP-complet même en utilisant une fonction de poids Bottleneck.

Pour le montrer, nous allons faire une réduction depuis le problème de la recherche d'un cycle hamiltonien dans un graphe quelconque, qui est un problème NP-complet bien connu.

En effet, si l'on a un graphe dans lequel on cherche un cycle hamiltonien, on peut obtenir une instance de la recherche d'un cycle hamiltonien de poids minimal dans une clique :

En partant du graphe originel, on ajoute un poids 1 à toutes les arrêtes, puis on complète le graphe pour obtenir une clique en ajoutant des arrêtes de poids 2.

Trouver un cycle hamiltonien de poids 1 dans cette clique est bien équivalent à trouver un cycle hamiltonien dans la clique de départ. On a donc bien la preuve que le problème du voyageur de commerce en version Bottleneck est un problème NP-complet.

2.3 Un problème Bottleneck NP-complet dont la version classique est résoluble en temps polynomial ?

Les problèmes en version Bottleneck peuvent en général être résolu avec une complexité inférieure ou égale à celle de leur équivalent en version classique car le nombre de structures de données différentes à envisager est bien moindre : en effet, par exemple, là où dans un algorithme classique devrait considérer chaque chemin, avec un poids différent, pour l'algorithme Bottleneck, on peut souvent ne considérer que des groupes de chemin, créé en fonction de l'arête de plus haut poids, qui ont tous le même poids.

Je n'ai pas réussi à exhiber de problème Bottleneck NP-complet dont la version classique soit résoluble en temps polynomial, et j'ai tendance à penser qu'il est particulièrement difficile d'en exhiber un, mais je n'ai pas trouvé de façon efficace de formaliser le lien entre un problème Bottleneck et sa version classique, pour pouvoir en tirer un théorème.