Nantes Université

Projet “élimination au baseball”

Graphes II et Réseaux

Table des matières

Juanfer Mercier – Adrien PICHON

Université de Nantes — UFR Sciences et Techniques
Master informatique parcours "Optimisation en Recherche Opérationnelle"
Année académique 2022-2023

1 Présentation du problème

1.1 Introduction

Aux Etats-Unis, le jeu de baseball est organisé en deux ligues, chacune contenant trois divisions de 4-5 équipes chacune. Pendant une saison, chaque équipe joue 162 matchs, qui ont lieu contre des équipes diverses : 76 matchs contre des équipes de sa propre division, 66 matchs contre des équipes de sa propre ligue mais d'autres divisions et 20 matchs contre des équipes de l'autre ligue. Le but de chaque équipe est de se qualifier pour les barrages, et pour cela il faut gagner que l'équipe gagne sa division. Nous indiquerons dans ce rapport comment nous avons implémenter une méthode à base de flots pour vérifier, à tout moment du championnat, quelles sont les équipes déjà éliminées (qui ne peuvent donc plus être premières de leur division) et lesquelles font encore partie de la compétition.

1.2 Notations

Equipe	**Matchs remportés**	**Matchs restants**	**Matchs contre**			
			NY	BOS	Tor	Bal
New York Yankees	93	8	-	1	6	1
Boston Red Sox	89	4	1	-	0	2
Toronto Blue Jays	88	7	6	0	-	1
Baltimore Orioles	86	5	1	3	1	-

TABLE 1 – Exemple de division dans laquelle 4 équipes s'affrontent

La TABLE 1 est un exemple de division dans laquelle quatre équipes de baseball s'affrontent.

Définition 1

Une équipe gagne sa division si, à la fin de tous les matchs, aucune autre équipe de la division n'a gagné plus de matchs qu'elle. Plusieurs équipes peuvent gagner la même division mais tout match est soit gagné soit perdu, sans égalité possible.

Définition 2

Une équipe est éliminée à un moment donné du championnat si elle ne peut plus gagner sa division, quels que soient les matchs qu'elle gagnerait ou perdrait par la suite (mais elle joue tous les matchs restants même si elle ne peut plus gagner sa division).

Soit T l'ensemble des équipes de la division et soient les notations suivantes pour toute équipe i de T :

- w_i le nombre de victoires actuelles pour l'équipe i
- g_i le nombre de matchs restant à jouer pour l'équipe i
- g_{ij} le nombre de matchs restant entre l'équipe i et l'équipe j

Il est possible de tester si une équipe k est éliminée en créant un réseau de transport RT_k et en résolvant un problème de flot (voir sous-section 1.3).

Cependant, l'algorithme le plus efficace n'a pas besoin de tester toutes les équipes de la manière susmentionnée grâce au lemme suivant :

Lemme 1

Si une équipe k est éliminée, alors toute équipe h telle que $w_k + g_k \geqslant w_h + g_h$ est également éliminée.

Il en découle que $O(log|T|)$ constructions de réseaux et calculs de flots sont suffisants pour tester quelles équipes sont déjà éliminées et lesquelles non.

1.3 Réseau de transport

On cherche à vérifier si une équipe k a été éliminée. Pour cela on construit le graphe suivant en calculant la valeur de flux maximum.

La première étape pour construire le réseau consiste a créer des noeuds de "matches" entre deux équipes et cela pour tout les matches possibles en dehors de ceux de l'équipe pour laquelle on étudie l'élimination potentielle. la capacité des arcs entre la sources et ces noeuds est égale au nombre de matches restant entre ces deux équipes (g_{ij}).

Ensuite nous avons besoin des noeuds représentant les équipes. Ces noeuds sont liées aux noeuds "matches" dans lesquelles l'équipe joue et le puits. Les capacités de ces arcs sont respectivement infinie et égales au nombre de matches gagnés par l'équipe k plus le nombre de matches qu'il lui reste à jouer moins le nombre de matches gagnés par l'équipe du noeud.

Vérifier si une équipe est éliminée consiste à prouver qu'il n'existe pas un flux de valeur $G = \sum_{i,j \in T-k} g_{ij}$.

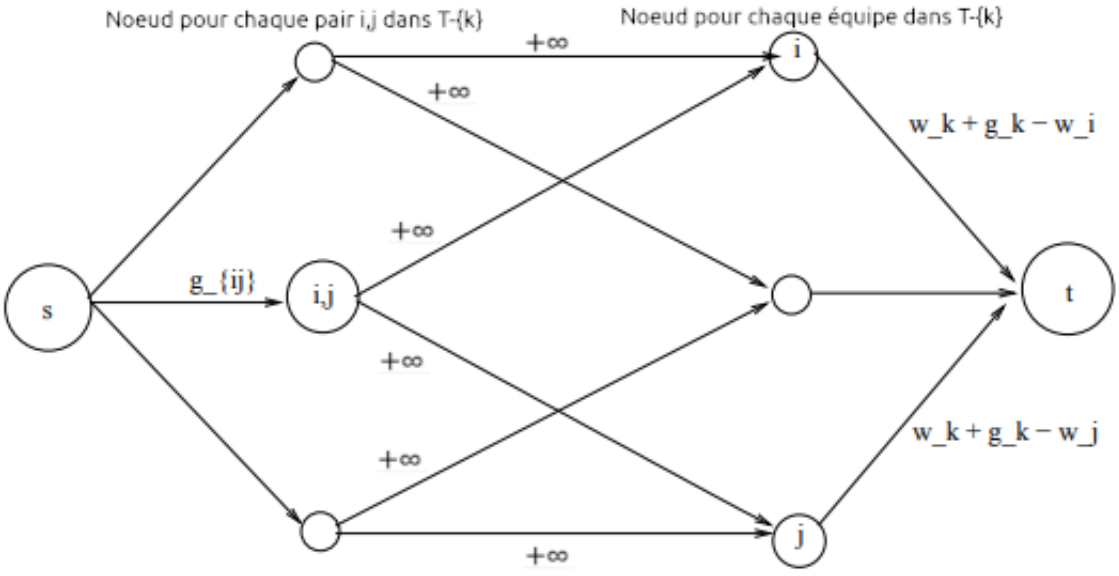

FIGURE 1 – Réseau de transport pour décider si l'équipe k est éliminée.

1.4 Exemple didactique

Pour la division présentée TABLE 1, on note les équipes de 1 à 4 dans l'ordre d'apparition des équipes dans la division (donc les Yankees sont notés 1, les Red Sox 2, ainsi de suite). La FIGURE 2 représente le réseau de transport obtenu pour l'équipe de Boston. On remarque sur ce réseau que la capacité de l'arête 1 — t est nulle tandis que pour l'arête 3 — t la capacité est égale à 5. Ainsi, il sera possible d'envoyer un flot de 5 par l'arête 3 — t mais il restera une unité qui ne pourra pas passer par l'arête 1 — t (car elle est de capacité nulle) et l'arête s — 1,3 ne sera donc pas saturée (le flot ne sera que de 5 à travers cette arrête). On conclut donc que l'équipe de Boston est éliminée car le flot maximum pouvant être obtenu est de 7 (il aurait fallu qu'il soit de 8 pour que Boston ne soit pas éliminée).

Enfin, on remarque que le lemme s'applique ici ($93 = 89 + 4 \geqslant 86 + 5 = 91$) et Baltimore est aussi éliminée.

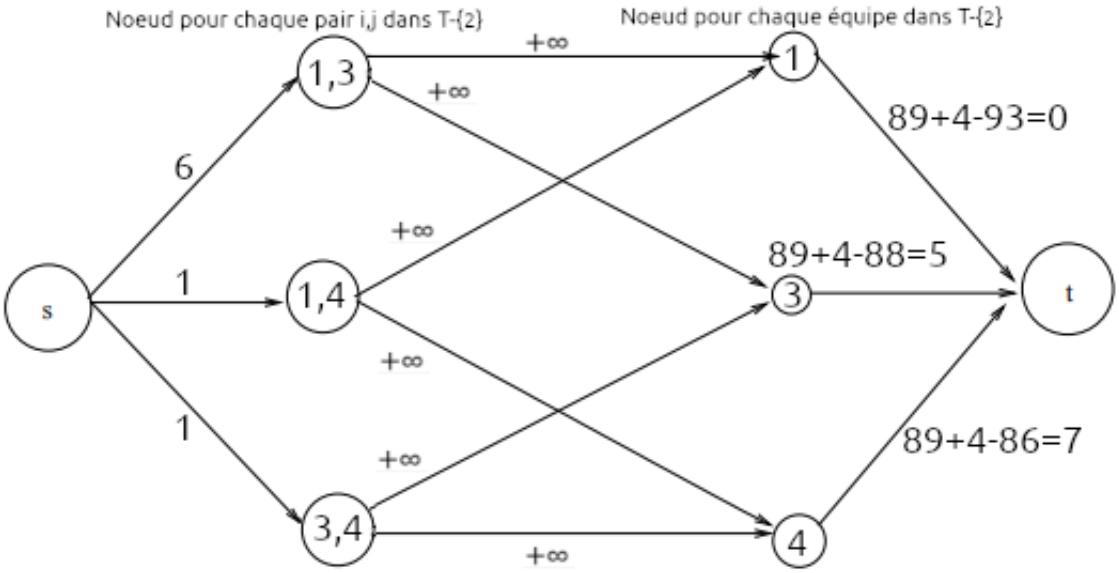

FIGURE 2 – Réseau de transport de l'équipe de Boston.

2 Instances et compilation du programme

2.1 Présentation des instances

Nos instances de test sont tirées de `https://github.com/ananya77041/baseball-elimination/tree/master/bin` . Nous avons adapté ces instances à notre formulation du problème ce qui donne 7 instances avec des nombres variés d'équipes (1, 4, 5, 10, 24 et jusqu'à 29 équipes). Dans notre arborescence de fichiers, le dossier "instances" est situé dans le même répertoire que *Baseball.java* et *FlowNetwork.java* et contient les différentes instances utilisées.

Une instance est découpée comme suit :

- Sur la première ligne, le nombre n d'équipes
- Sur les n lignes suivantes : pour chaque équipe, son indice i (dans l'ordre croissant de 1 à n), son nom (sans espaces), le nombre de match gagné par l'équipe i jusqu'ici, le nombre total de matches restants pour l'équipe i et, pour toute équipe j d'indice allant de 1 à n dans cet ordre, le nombre de matches à jouer entre i et j

2.2 Commande de compilation du programme

Commande 1

▷ javac Baseball.java FlowNetwork.java

Cette commande compile les trois classes nécessaires à la résolution. Pour l'exécuter il est impératif de se trouver dans le même répertoire que les fichiers sources *Baseball.java* et *FlowNetwork.java*.

2.3 Commande d'exécution du programme

Commande 2

▷ java Baseball.java *chemin-vers-instance*

Où *chemin-vers-instance* est le chemin vers le fichier représentant l'instance sur laquelle il faut lancer le programme. Pour exécuter cette commande, il est impératif de se trouver dans le même répertoire que les fichiers sources *Baseball.java* et *FlowNetwork.java*.

Exemple :

Commande 3

▷ java Baseball.java instance/teams29.txt

3 Algorithme de résolution

Pour résoudre le problème d'élimination au baseball nous avons implémenté l'algorithme de Ford-Fulkerson. L'algorithme se divise en quatre méthodes principales :

- `FordFulkerson`, la méthode de Ford-Fulkerson.
- `ConstructionReseau`, qui construit le réseau de transport d'une équipe k selon la description donnée FIGURE 1.
- `TestEliminationEquipe`, qui réalise le test impliquant le réseau de transport pour une équipe k.
- `TestEliminationToutes`, qui décide pour toutes les équipes si elles sont éliminées ou non.

La résolution d'une instance commence d'abord par la lecture de celle-ci. Nous chargeons les noms des équipes dans un tableau de chaînes de caractères et les données numériques (w_i, g_i et g_{ij}) dans une matrice à n lignes et $n+2$ colonnes (avec $n = |T|$ le nombre d'équipes). Enfin, pour garder une trace mémoire des équipes éliminées, nous utilisons un tableau de n booléens indiquant si une équipe a été éliminée ou non. Lors de l'exécution du programme, c'est ce tableau de booléens qui sera modifié pour déterminer quelles sont les équipes en lice à la fin de la méthode. L'algorithme 1 présente un aperçu global du programme.

3.1 Les procédures

Algorithme 1 Aperçu global du programme

```
noms, éliminées, données ← lectureFichier()    ▷ on stocke les noms, les états (éliminée ou non) et les données relatives aux équipes
éliminées ← TestEliminationToutes(éliminées, données)
Pour i dans 1...n, faire :    ▷ Affichage des équipes restants dans la division
    Si non éliminées[i] alors
        Afficher(noms[i])
    Fin si
Fin pour
```

Algorithme 2 Procédure `TestEliminationToutes`

```
Requiert : éliminées, données
Pour i dans 1...n, faire :
    Si non éliminées[i] alors
        RT_i ← ConstructionReseau(i, données, éliminées)
        TestEliminationEquipe(i, données, RT_i)    ▷ RT_i est un réseau de transport suivant la description donnée précedemment
    Fin si
Fin pour
```

Algorithme 3 Procédure `ConstructionReseau`

Requiert : k, données, éliminées ▷ k l'indice de l'équipe dont on construit le réseau

$L \leftarrow$ `nouvelleListeAdjacence()`

$a \leftarrow 0$ ▷ Variable indiquant l'indice du noeud que l'on ajoute actuellement à L

$b \leftarrow \frac{(n-1)*(n-2)}{2}$, $c \leftarrow 0$ ▷ Remplisse la même fonction que a

Pour i **dans** $1 \dots n-2$, **faire** :

$b \leftarrow b+1$, $c \leftarrow b$

Pour j **dans** $i+1 \dots n-1$, **faire** :

Si $i \neq j \neq k$ et non éliminées[i] et non éliminées[j] **alors**

$a \leftarrow a+1$, $c \leftarrow c+1$

`AjoutArête(`L`,` s`,` a`,` données`[`i`][`j`+2],` `0)` ▷ Nouvelle arête de la source s au noeud "match" numéro a de capacité g_{ij} (données[i][j+2]) et de flot nul

`AjoutArête(`L`,` a`,` b`,` $+\infty$`,` `0)` ▷ Arêtes avec capacité infinie

`AjoutArête(`L`,` a`,` c`,` $+\infty$`,` `0)`

Si $i == 1$ **alors**

capacité $\leftarrow$ données[k][0] + données[k][1] - données[i][0]

`AjoutArête(`L`,` $\frac{(n-1)*(n-2)}{2}+j$`,` p`,` capacité`,` `0)` ▷ Arêtes vers le puits p de capacité $w_k + g_k - w_i$

Fin si

Fin si

Fin pour

Fin pour

Algorithme 4 Procédure `TestEliminationEquipe`

Requiert : k, données, éliminées, RT_k

Si not `existeFlot?(`RT_k`)` **alors**

éliminées[k] $\leftarrow$ *true*

Fin si

Si éliminées[k] **alors**

`utiliserLemme(`k`,` données, éliminées`)`

Fin si

Algorithme 5 Prédicat `existeFlot?`

Requiert : RT_k

flot $\leftarrow$ `FordFulkerson(`RT_k`)`

`retourne(flot ==` G`)`

Algorithme 6 Procédure `utiliserLemme`

Requiert : k, données, éliminées
  Pour i **dans** $1 \dots n$, **faire** : ▷ Utilisation du lemme
    Si $i \neq k$ et non éliminées[i] et non éliminées[k] et données[i][0] + données[i][1] $\leqslant$ données[k][0] + données[k][1] **alors**
      éliminées[i] $\leftarrow$ *true* ▷ La condition $w_i + g_i \leqslant w_k + g_k$ est vraie donc selon le lemme l'équipe i est aussi éliminée
    Fin si
  Fin pour

Pour chaque équipe i non éliminée, la procédure `TestEliminationToutes` (algorithme 2) fait appel à la procédure `ConstructionReseau` (algorithme 3) pour construire le réseau de transport associé à l'équipe i. Ce réseau est construit en $\frac{(n-1)(n-2)}{2}$ itérations dans la procédure `ConstructionReseau`. Ce nombre représente le minimum d'étapes nécessaires pour constituer les couples d'équipes formant les noeuds "match". Grâce à l'utilisation astucieuse de compteurs et des valeurs des itérateurs de boucle on crée le réseau de transport de l'équipe considérée en utilisant le moins d'itérations possible (complexité en $O(n^2)$ dans le meilleur et pire cas).

Une fois le réseau de transport construit, `TestEliminationToutes` fait appel à la procédure `TestEliminationEquipe` (algorithme 4) qui lance le calcul du flot dans le réseau de transport (à travers le prédicat `existeFlot?` (5) qui lui même fait appel à la méthode `FordFulkerson` et vérifie si la valeur du flot est bien égal à $G = \sum_{i,j \in T-k,\, i<j} g_{ij}$). Si le flot trouvé n'est pas égal à G alors l'équipe k est éliminée et la procédure `utiliserLemme` (algorithme 6 en $O(n)$ dans le pire des cas) est appelée pour éliminer toute équipe h telle que $w_h + g_h \leqslant w_k + g_k$.