Devoir 3 : Flots maximums et coupes minimums, application.

Ce document décrit l'application que vous devez traiter avec l'algorithme de flot maximum.

1 Le problème

Le problème consiste en un ensemble de tâches à réaliser. Chaque tâche dispose d'un coût de réalisation, et d'un profit si on la réalise. De fait, certaines tâches auront un coût supérieur au profit, et donc auront un impact négatif (tâches déficitaires), d'autres auront un profit supérieur au coût et auront dont un impact positif (tâches bénéficiaires). Nous souhaitons maximiser la somme des profits moins la somme des coûts des tâches que nous allons choisir de réaliser.

Si on en reste là, la solution optimale est évidemment de ne réaliser que (et toutes) les tâches bénéfitaires. La complexité vient du fait que les tâches sont soumises à une relation de précédence : il peut être nécessaire d'avoir fait un certaine tâche pour pouvoir en réaliser une autre. Du coup, certains tâches bénéficiaires ne peuvent être réalisées que si d'autres tâches déficitaires ont aussi été réalisées.

Nous allons nous placer dans un cadre un peu plus précis mais qui appartient à cette catégorie de problèmes. Dans le cadre de l'extraction minière, certaines mines sont à ciel ouvert. L'industriel creuse la mine à partir d'un sol plat en creusant vers le bas, créant un trou conique. Avant de commencer l'exploitation de la mine, les ingénieurs effectuent des sondages pour évaluer la quantité de minerai exploitable et la localisation de ce minerai dans le sous-sol, obtenant une cartographie de ce sous-sol. Le sous-sol est découpé en bloc, et pour chaque bloc la quantité de minerai exploitable ainsi que le coût d'extraction du bloc sont connus. De plus, pour pouvoir exploiter un bloc, il faut retirer les blocs qui sont situés au-dessus afin de pouvoir y accéder.

L'industriel souhaite donc savoir quels blocs retirer et quels blocs laisser. Chaque bloc consiste en une tâche, avec un coût (le coût d'extraction) et un profit (la quantité de minerai fois la valeur du minerai). Les précédences correspondent aux contraintes de blocs situés au-dessus les uns des autres.

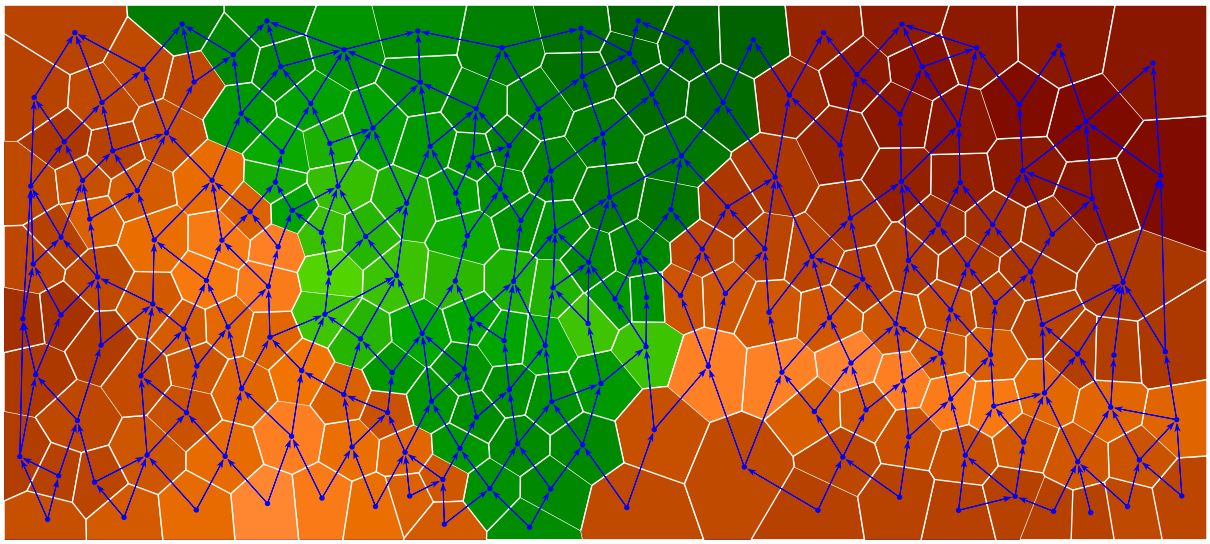

FIGURE 1 – Un exemple de mine, plus un bloc est clair, plus il contient de minerai. Les blocs en vert sont les blocs extraits. Le graphe en bleu modélise les contraints de précédence. Les coins supérieurs ne peuvent pas être exploité.

2 Résolution avec une coupe minimum

Pour résoudre le problème, on se ramène au problème de trouver une coupe minimum dans un graphe. C'est un problème de minimisation, donc on va chercher à construire un graphe avec deux sommets s et t, tel qu'une s,t-coupe corresponde à un ensemble de blocs, et que la capacité d'une coupe correspond au déficit que génére l'exploitation de ces blocs. Si le déficit est minimisé, c'est que le profit est maximisé.

Le graphe est obtenu en prenant un sommet par bloc, plus les sommets s et t. Ainsi tout ensemble de bloc est bien représentable par une s,t-coupe, et inversement. Il faut ensuite :

— ajouter des arcs pour forcer les contraintes de précédence. Si le bloc A est en-dessous du bloc B, et que le sommet pour A est dans la coupe minimum, il faut que le sommet pour B soit aussi dans la coupe minimum. Quel arc ajouté pour cela, et avec quelle capacité ?

— ajouter pour chaque bloc un arc pour encoder le coût d'exploitation de ce bloc. S'il est positif et que ce bloc est pris dans la coupe minimum, son coût doit être considérer dans la capacité de la coupe. Si son coût est négatif, et que le bloc n'est pas exploité, alors on considère cela comme une perte, et cette perte doit être comptée dans la capacité de la coupe. Quels arcs ajouter pour cela ?

Par ailleurs, il est interdit de retirer l'un des deux blocs des coins supérieurs gauche et droit.

Réflechissez un moment à comment modéliser le problème par un graphe, et si vous ne trouvez pas, demandez à votre chargé de TP.

Une fois le graphe construit, on calcule le s,t-flot maximum, et on en déduit la s,t-coupe minimum.

3 Instances

Nous vous fournissons un jeu d'instances de différentes tailles. Votre algorithme doit pouvoir toutes les résoudre, avec un temps de calcul négligeable. Nous vous fournissons aussi une classe pour visualiser une solution.

Les instances sont des fichiers au format `json`. Vous pouvez ouvrir un des petits fichiers (le numéro figurant dans le nom du fichier est le nombre de blocs de la mine), et voir comment les instances sont données. En particulier les blocs possèdent un identifiant unique, qui couvrent un intervalle d'entiers consécutifs à partir de 0. Pour lire ces fichiers, nous vous donnons une classe `OpenPitInstance` qui le fait automatiquement à partir du nom du fichier. Elle est basée sur le paquet `com.google.gson` qui permet de lire et écrire facilement des fichiers `json`. Ce paquet doit être installé (depuis votre IDE). Soyez curieux et regarder comment cette classe `OpenPitInstance` est écrite, cela pourra vous être utile plus tard.

La classe de visualisation `OpenPitVisualisation` est décrite par :

— un constructeur, qui attend un dictionnaire associant à chaque identifiant de bloc, le bloc correspondant,
— une méthode `setSolution` permettant de définir l'ensemble des blocs à exploiter,
— une méthode `addBlockToSolution` permettant d'ajouter un bloc dans la solution,
— une méthode `startVisual` pour créer la fenêtre d'affichage,
— une méthode `repaint` pour redessiner la solution.

```
public OpenPitVisualisation(Map<Integer,OpenPitInstance.Block> instance) {
  ...
}

public void setSolution(Collection<Integer> minCut) {
  ...
}

public void addToSolution(int blockId) {
  ...
}

public void startVisual((JFrame window) {
  ...
}

public void repaint() {}
```

L'argument de `startVisual` est obtenu en ajoutant dans la classe `Main` :

```
private final static JFrame window = new JFrame("visual");
```

4 Extras

C'est bien d'avoir les blocs à exploiter, mais c'est mieux si on sait dans quel ordre les extraire. Vous pouvez donc ajouter un algorithme de tri topologique, pour trouver l'ordre d'extraction.