Séparation et évaluation

Recherche opérationnelle
Dimitri Watel - ENSIIE

2024

La méthode de séparation et d'évaluation ou *Branch and Bound* and anglais est une méthode permettant de construire un algorithme exact pour résoudre un problème d'optimisation combinatoire. L'idée est d'utiliser un algorithme d'énumération totale sous forme arborescente auquel on ajoute de l'information pour éviter d'énumérer toutes les solutions

1 Exemples

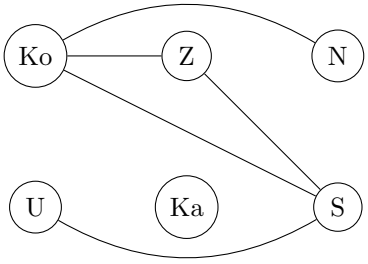
Dans cette section, on va décrire la méthode au travers de deux exemples. La section suivante synthétisera ces exemples et détaillera les algorithmes. Les deux exemples donnés ici sont des problèmes de graphe mais, bien entendu, la technique peut s'appliquer à n'importe quel problème combinatoire (où la décision est discrète).

1.1 Le problème de stable maximum

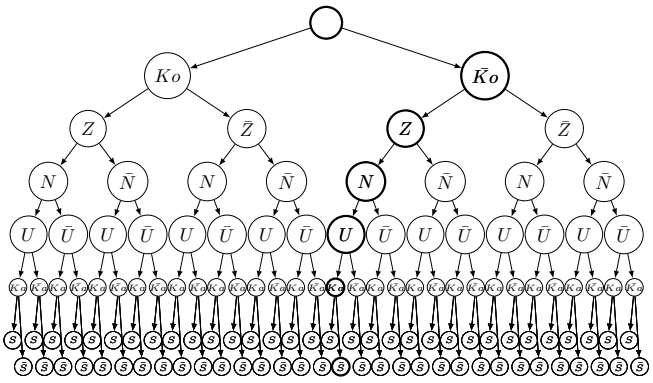
Un *stable* dans un graphe $G = (V, E)$ est un sous-ensemble de nœuds $V' \subset V$ tels que deux nœuds de V' ne sont pas reliés par une arête de E.

Problème 1. Le problème du stable maximum consiste, connaissant un graphe $G = (V, E)$ à trouver un stable V' de G de taille $|V'|$ maximum.

On peut résoudre assez simplement ce problème en testant tous les sous-ensembles V' possibles. Une manière de générer ces sous-ensembles est d'utiliser une représentation arborescente des solutions, c'est l'étape de séparation ou de branchement. Considérons l'exemple suivant:



Un arbre de séparation serait le suivant, où à chaque nœud interne de l'arbre, on prend la décision de mettre ou de ne pas mettre un des sommets du graphe dans V'. Chaque branche de l'arbre reliant la racine à une feuille est une solution. Cette solution peut être non réalisable (comme c'est le cas avec la branche de gauche). Une solution optimale peut ensuite être extraite de l'arbre en regardant, parmi les solutions réalisables, celles qui maximisent le nombre de nœuds dans V'.



L'objectif d'un algorithme de séparation et d'évaluation est de ne pas explorer, construire, mettre en mémoire tout cet arbre en coupant des branches dont on sait qu'elles ne mèneront pas à une solution optimale. Il y a deux méthodes pour couper une branche.

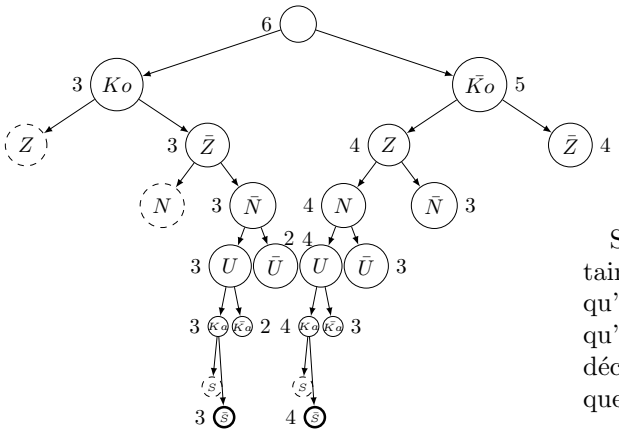
- La première consiste à dévoiler des décisions non réalisables. Par exemple ici, toute feuille dont la branche commence par Ko et Z est non réalisable car aucun stable ne contient ces deux nœuds.

- Une seconde idée consiste à regarder le poids de la meilleure solution et le poids de la solution courante. Si on explore l'arbre en profondeur à gauche, la première solution qu'on va dévoiler est $Ko, \bar{Z}, \bar{N}, U, Ka, \bar{S}$. On a donc une solution avec 3 nœuds. Considérons maintenant la branche $\bar{K}o, Z, \bar{N}$ donc toutes les solution contenant Z et ne contenant ni Ko ni N. Quel que soit le choix fait pour U, Ka et S, on ne pourra jamais avoir plus de 3 sommets dans le stable. En effet, Z empêche de sélectionner S, ce stable contiendra, au mieux Z U et Ka. Puisqu'on a déjà trouvé une solution avec 3 nœuds, il est inutile d'explorer les solutions issues

des choix $\bar{Ko}, Z, \bar{N}$ (on rappelle qu'on veut une solution optimale, pas toutes les solutions optimales).

Considérons pour exemple une autre branche, la branche $\bar{Ko}$ et $\bar{Z}$. Au mieux, toute solution réalisable issu de cette décision ne peut contenir que 4 sommets. En effet, les sommets Ko et Z ne sont pas sélectionnés. Il en reste donc 4 qu'on peut espérer mettre dans une solution réalisable. Si on a déjà trouvé une solution avec 4 sommets ou plus, il n'est pas nécessaire d'explorer cette branche plus loin. Notez avec cet exemple que cette valeur 4 est ici une borne supérieure. Il n'existe pas de solution de taille 4 ne contenant pas Ko ni Z. En effet, U et S sont incompatibles. On parle donc bien de borne supérieure de la meilleure solution possible et non de la valeur de la meilleure solution possible. La seule manière de connaître cette valeur est d'explorer tous les nœuds issus de la branche.

En associant ces deux idées, on obtient l'arbre exploré suivant où la borne supérieure est indiquée à côté du nœud. Les nœuds en pointillé sont des décisions non réalisables.



On retrouve bien la solution optimale, mais on a exploré beaucoup moins de nœuds. On a coupé 4 branches car la solution n'était pas réalisable. On a coupé 6 branches car la borne supérieure indiquait qu'on ne ferait pas mieux que la meilleure solution. Et on a dévoilé deux solutions rélalisables, respectivement de valeur 3 et 4.

On a ici effectué une exploration en prodonfeur à gauche. On a exploré 13 nœuds (les nœuds qui n'ont pas été coupés). On aurait pu faire une exploration en profondeur à droite, ou une exploration en profondeur aléatoire. Le résultat aurait été le même mais le nombre de nœuds coupés et explorés aurait été différent. Dans le pire des cas, on est contraint de dévoiler tout l'arbre. Dans le meilleur cas, on tombe directement sur la solution optimale et si les bornes sont basses, on coupe tous les autres nœuds.

On va décrire ci-après un autre exemple avec une autre méthode d'exploration.

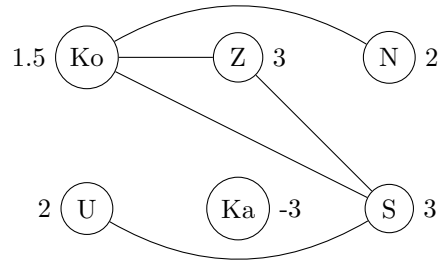
1.2 Le problème de sélection de poids maximum

On considère une variante relâchée du problème de stable maximmum. On peut maintenant prendre deux nœuds dans V' qui sont voisins mais cela coûte une pénalité.

Problème 2. Le problème du sélection de poids maximum consiste, connaissant un graphe $G = (V, E)$ et des poids $\omega : V \to \mathbb{R}$ à trouver un ensemble $V' \subset V$ maximisant $\sum_{v \in V'} \omega(v) - |E[V']|$ où $E[V']$ sont les arêtes du graphe induit par V'.

On peut encore résoudre assez simplement ce problème en testant tous les sous-ensembles V' possibles avec une exploration arborescente. On peut utiliser la technique de séparation et d'évaluation, mais il nous faut une nouvelle borne supérieure.

Considérons l'exemple suivant où $\omega(v)$ est inscrit à côté du nœud v.



Supposons qu'on ait déjà pris une décision pour certains nœuds, notons W^+ les nœuds dont on a décidé qu'ils seraient dans V' et W^- ceux dont on a décidé qu'ils ne seraient pas dans V'. Il reste à prendre une décision pour les autres. On peut déjà compter la valeur que rapportent ces nœuds:

$$b \leftarrow \sum_{v \in W^+} \omega(v) - |E[W^+]|$$

Considérons par exemple plus haut que $W^+ = \{K_o\}$ et $W^- = \emptyset$, alors b démarre avec 1.5.

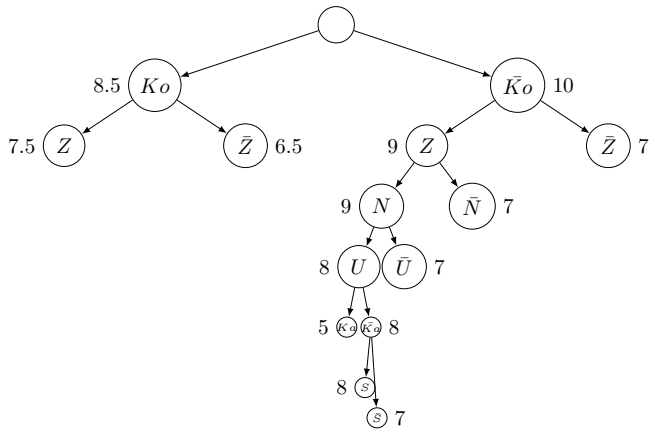
On calcule ensuite, pour chaque nœud $v \in V \backslash W^+ \cup W^-$, la valeur suivante. Elle correspond au gain et aux pénalités qui relient v et W^+. Si cette valeur est positive, on l'ajoute à b. Sinon on ne fait rien, on considère alors qu'il vaut mieux ne pas ajouter v à V'. C'est le cas par exemple si v introduit beaucoup de pénalités.

$$c_v \leftarrow \omega(v) - |\{(u, v) \in E | u \in W^+\}|$$
$$b \leftarrow b + \max(c_v, 0)$$

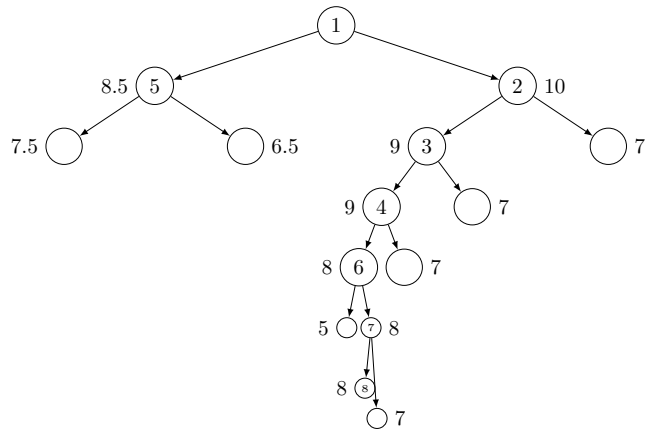
Dans notre exemple, $c_Z = 2, c_N = 1, c_U = 2, c_{Ka} = -3$ et $c_S = 2$. On a donc $b = 8.5$ à l'issue du calcul.

Encore une fois on parle bien ici de borne supérieure de la meilleure solution contenant W^+ et non de valeur optimale parmi ces solutions. Il n'existe aucune solution contenant K_o et dont le poids est 8.5 mais toutes ces solutions ont un poids inférieur à 8.5. Si on trouve une solution de valeur supérieure à 8.5, on sait qu'il est inutile d'explorer la branche démarrant en K_o.

Contrairement au premier exemple où on a exploré l'arbre en profondeur à gauche, on va ici utiliser une exploration du meilleur d'abord, on va toujours explorer le nœud qui a la borne la plus grande dans l'arbre.



On a indiqué ci-après l'ordre dans lequel les nœuds étaient explorés. On a exploré 8 nœuds et coupé 7 nœuds.



2 Synthèse

La méthode de séparation et d'évaluation est une méthode pour construire un algorithme exact pour un problème d'optimisation combinatoire.

La première étape consiste à **séparer le problème en plusieurs sous-problèmes**. Généralement, cette séparation se fait en prenant une décision pour une des variables du problèmes (dans nos exemples, sélectionner ou non un nœud), mais on peut faire des choses plus générales qui dépendent du problème qu'on souhaite résoudre.

Soit un problème d'optimisation Π dont l'ensemble des solutions réalisables est $\xi(\Pi)$. Une séparation $S(\Pi)$ en sous-problèmes est un ensemble de problèmes d'optimisations $(\Pi_1, \Pi_2, \dots \Pi_p)$ tels que

$$\bigcup_{j=1}^{p} \xi(\Pi_j) = \xi(\Pi)$$

Il faut dans la mesure du possible que les solutions des sous-problèmes s'intersectent le moins possible pour réduire l'espace de recherche inutile. Dans les exemples plus haut, on a Π qui correspond au nœud racine de l'arbre et Π_1, Π_2 qui correspondent aux nœuds K_o et $\bar{K_o}$.

La deuxième étape consiste à **choisir une borne** de l'ensemble des valeurs de toutes les solutions. Dans nos deux exemples, on avait un problème de maximisation. Dans ce cas là, il faut une borne supérieure. **Attention**, dans le cas d'un problème de minimisation, il faut une borne inférieure.

Soit un problème de **minimisation** Π. On note $\omega(s)$ la valeur d'une solution réalisable s de $\xi(\Pi)$. Soit s^* une solution optimale de Π (minimisant $\omega(s^*)$).

Une borne $B(\Pi)$ est un nombre **rapide à calculer** et vérifiant

$$B(\Pi) \leq \omega(s^*)$$

Dans le cas d'un problème de maximisation, on aura $B(\Pi) \geq \omega(s^*)$. On rappelle que chaque problème se sépare en sous-problèmes qui sont eux-mêmes des problèmes d'optimisation et à qui on peut appliquer la borne B.

Comme écrit plus haut, la borne doit être rapide à calculer, car on va devoir la calculer pour chaque nœud de l'arbre qu'on va dévoiler. Cependant, la borne doit aussi être efficace, plus elle est proche de $\omega(s^*)$, plus on a de chance de couper le nœud courant. Par exemple dans le problème de stable maximum, une borne très rapide à calculer est $|V|$. Il s'agit bien d'une borne supérieure d'un stable de G mais ce n'est pas une borne très utile car aucun nœud ne sera coupé à cause de la borne. Inversement, la meilleure borne consisterait à calculer et renvoyer $\omega(s^*)$ mais cela prendrait beaucoup de temps (possiblement un temps exponentiel pour chaque nœud de l'arbre).

La dernière étape consiste à **déterminer une tactique d'exploration**. Connaissant un arbre partiellement dévoilé, quel nœud faut-il explorer ? Un détail important est de définir clairement la notion d'exploration.

Définition 1. *Explorer un nœud* Π signifie calculer $S(\Pi)$ et $B(\pi)$ pour tout $\pi \in S(\Pi)$.

Ce n'est pas parce qu'un nœud est affiché qu'il est exploré. Un nœud peut être coupé parce que sa borne

est trop petite/grande ou parce qu'il correspond à une solution non réalisable. On considère dans ce cas qu'il n'est pas exploré: on ne dévoile pas ses fils $S(\Pi)$ dans l'arbre. Le fait de calculer la borne du nœud ne fait pas partie de l'exploration, cela fait partie de l'exploration du nœud parent. Seule exception, la racine qui n'a pas de parent. Cependant, le calcul de la borne de la racine est rarement utile.

Comme on l'a vu, il existe deux techniques d'exploration simples, **l'exploration en profondeur à gauche** qui explore toujours le nœud non exploré le plus à gauche, et **l'exploration du meilleur d'abord** qui explore toujours le nœud non exploré avec la borne la plus prometteuse. La première est moins coûteuse en mémoire et trouve rapidement des solutions réalisables. La seconde tombe du premier coup sur de très bonnes solutions mais prend plus de temps et de mémoire pour y arriver. Il ne faut bien entendu pas se limiter à ces techniques. En fonction du problème, en fonction de l'instance, il faut choisir la bonne technique. On peut aussi changer de tactique au cours de l'algorithme, il n'y a pas de règle.

On termine ici par présenter deux pseudocodes pour les deux explorations standards.

Algorithme 1 Exploration en profondeur à gauche

Entrées: un problème de **minimisation** Π, et un réel $uB \geq \omega(s^*)$. Ce réel faut $+\infty$ par défaut.
Sorties: Le poids $\omega(s^*)$ d'une solution optimale s^* de Π

```
1: function BB(Π, uB)
2:     Si Π ne peut être séparé Alors Renvoyer Le poids ω(s*) d'une solution optimale s* de Π
3:     Explore Π
4:     Pour Π_j ∈ S(Π) tel que ξ(Π_j) ≠ ∅ Faire
5:         lb ← B(Π_j)
6:         Si lb < ub Alors
7:             ω ← BB(Π_j, ub)
8:             ub ← min(ub, ω)
9:     Renvoyer uB
```

Algorithme 2 Exploration du meilleur d'abord

Entrées: un problème de **minimisation** Π
Sorties: Le poids $\omega(s^*)$ d'une solution optimale s^* de Π

```
1: function BB(Π)
2:     L ← [Π] ; uB ← +∞
3:     Tant que L ≠ ∅ Faire
4:         π ← arg min_{π∈L}(B(π)) ; Remove π from L
5:         Si B(π) ≥ ub Alors
6:             Break
7:         Si π ne peut être séparé Alors
8:             ub = min(ub, Poids ω(s*) d'une sol opt s* de π)
9:         Sinon
10:            Explorer Π
11:            Ajouter tous les éléments de S(Π) à L
12:    Renvoyer uB
```