Un problème NP-complet ne peut a priori pas être résolu en complexité polynomiale. On peut chercher des alternatives :

- Un algorithme d'approximation qui renvoie une solution approchée en complexité polynomiale.
- Un algorithme exact qui accélère une méthode de recherche exhaustive (tout en restant en complexité exponentielle), comme la méthode Branch and Bound.

I Problème d'optimisation

Définition : Problème d'optimisation

Un problème d'optimisation est un triplet (I, S, f) avec I un ensemble d'instances, $S \subset I$ un ensemble de solutions et f une fonction objectif à optimiser (minimiser ou maximiser).
Un algorithme A résout ce problème d'optimisation si pour toute instance $i \in I$, $A(i)$ renvoie une solution $s \in S$ telle que $f(s)$ est optimum.

Exemple :

STABLE (optimisation)

Instance : un graphe G
Solution : un ensemble stable de G (ensemble de sommets sans arêtes entre eux)
Optimisation : maximiser la taille de l'ensemble stable

Un problème d'optimisation peut être transformé en un problème de décision :

STABLE (décision)

Instance : un graphe G et un entier k
Question : G contient-il un ensemble stable de taille k ?

II Algorithme d'approximation

On ne sait pas résoudre un problème NP-complet en complexité polynomiale.
À la place, on peut chercher un algorithme d'approximation :

Définition : Algorithme d'approximation

Soit $\Pi = (I, S, f)$ un problème d'optimisation.
On dit qu'un algorithme A est une α-approximation de Π si pour toute instance x de Π :

- $A(x)$ termine et renvoie une solution $s \in S$
- si Π est un problème de maximisation : $f(s) \geq \alpha \cdot f(s^*)$
- si Π est un problème de minimisation : $f(s) \leq \alpha \cdot f(s^*)$

où s^* est une solution optimale de Π.

Exemple : Une 2-approximation d'un problème de minimisation doit renvoyer une solution dont la valeur est au plus deux fois plus grande que la solution optimale.

Exercice 1.
Soit $G = (S, A)$ un graphe non orienté. Une couverture par sommets de G est un ensemble $S' \subset S$ tel que pour toute arête $\{u, v\} \in A$, $u \in S'$ ou $v \in S'$.

VERTEX-COVER

Instance : un graphe G et un entier k
Question : G contient-il une couverture par sommets de taille k ?

1. On admet que CLIQUE est NP-complet. Montrer que VERTEX-COVER est NP-complet.
2. On appelle VERTEX-COVER-OPT le problème d'optimisation associé à VERTEX-COVER.
   Montrer que l'algorithme suivant n'est pas une α-approximation de VERTEX-COVER-OPT, quel que soit α :

```
S' ← ∅
Tant que A ≠ ∅ :
    Choisir {u, v} ∈ A
    S' ← S' ∪ {u}
    Retirer de A toutes les arêtes incidentes à u ou v
Renvoyer S'
```

3. Montrer que l'algorithme suivant est une 2-approximation de VERTEX-COVER-OPT :

```
S' ← ∅
Tant que A ≠ ∅ :
    Choisir {u, v} ∈ A
    S' ← S' ∪ {u, v}
    Retirer de A toutes les arêtes incidentes à u ou v
Renvoyer S'
```

4. Avec quelle complexité peut-on implémenter cet algorithme ?

Exercice 2.
On considère le problème :

COUPLAGE-POIDS-MAX

Instance : un graphe $G = (S, A)$ pondéré par $p : A \longrightarrow \mathbb{R}^+$
Solution : un couplage C de G
Optimisation : maximiser $f(C) = \sum_{\{u,v\} \in C} p(\{u, v\})$

Montrer que l'algorithme suivant est une 2-approximation de COUPLAGE-POIDS-MAX :

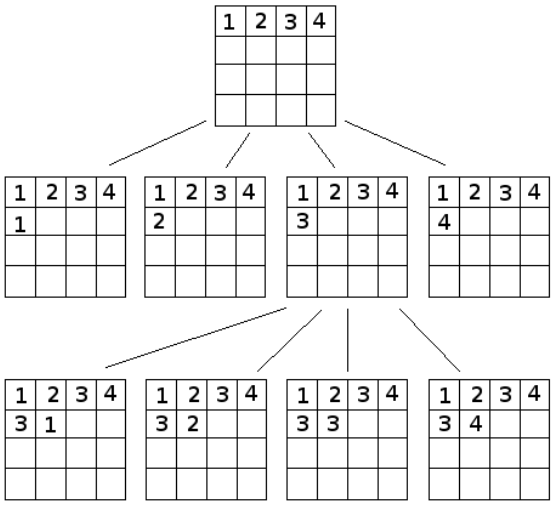
```
C ← ∅
Pour a ∈ A par ordre décroissant de poids :
    Si C ∪ {a} est un couplage :
        C ← C ∪ {a}
Renvoyer C
```

III Branch and Bound

Définition : Backtracking

Un algorithme par backtracking (retour sur trace) construit une solution petit à petit, en revenant en arrière s'il n'est pas possible de l'étendre en une solution complète.

Un backtracking revient à faire un parcours en profondeur sur l'arbre des solutions partielles, en passant à une branche suivante lorsqu'il n'est pas possible d'étendre une solution partielle :



Exemple : L'algorithme de Quine résout SAT en trouvant une valuation satisfaisant une formule logique φ par backtracking :

1. Choisir une variable x restante dans φ.
2. Tester récursivement si $\varphi[x \leftarrow 1]$ est satisfiable.
3. Si non, tester récursivement si $\varphi[x \leftarrow 0]$ est satisfiable.

L'algorithme simplifie la formule à chaque assignation de variable, ce qui peut permettre de couper des branches (si la formule est insatisfiable).
Cet algorithme est donc souvent plus efficace que la recherche exhaustive, même s'il reste en complexité exponentielle dans le pire cas.

Définition : Branch and Bound

Un algorithme par Branch and Bound (séparation et évaluation) permet d'accélérer la résolution par backtracking de Π en utilisant une heuristique h que si x est une solution partielle qui peut être étendue en une solution $y \in S$ alors $f(y) \leq h(x)$.
L'algorithme conserve la meilleure solution trouvée s^* et explore un sous-arbre x que si $h(x) \geq f(s^*)$.

Remarques :

- Le principe est le même pour un problème de minimisation en inversant les inégalités.

- On obtient une solution exacte (contrairement à un algorithme d'approximation) mais la complexité peut être exponentielle.

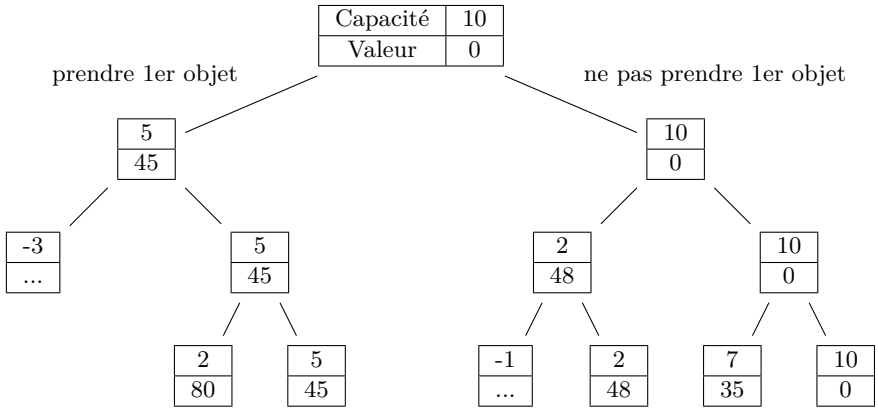
III.1 Exemple : Sac à dos

SAC-A-DOS

Instance : un ensemble d'objets de poids $p_1, \ldots, p_n$ et de valeurs $v_1, \ldots, v_n$ et une capacité P.
Solution : un sous-ensemble d'objets de poids total inférieur à P.
Optimisation : maximiser la valeur totale des objets.

Arbre des solutions partielles pour l'instance suivante de SAC-A-DOS :

Poids	5	8	3
Valeur	45	48	35



Exercice 3.

1. Donner une heuristique possible pour le problème du sac à dos.
2. Indiquer sur l'arbre ci-dessus les branches qui ne seront pas explorées par l'algorithme de Branch and Bound.