Algorithmique des graphes

11 — Branch and bound

Anthony Labarre

5 mai 2021

Université Gustave Eiffel

Motivations

- Les problèmes qu'on a couverts jusqu'à présent étaient tous faciles . . .

Motivations

- Les problèmes qu'on a couverts jusqu'à présent étaient tous faciles ...
- ...d'un point de vue algorithmique ; ce qui signifie qu'on arrive à les résoudre en temps polynomial en la taille de l'entrée ;

Motivations

- Les problèmes qu'on a couverts jusqu'à présent étaient tous faciles ...
- ...d'un point de vue algorithmique ; ce qui signifie qu'on arrive à les résoudre en temps polynomial en la taille de l'entrée ;
- De (très) nombreux autres problèmes sont “difficiles” : on ne connaît pas d'algorithme efficace pour les résoudre ;

Motivations

- Les problèmes qu'on a couverts jusqu'à présent étaient tous faciles . . .
- . . . d'un point de vue algorithmique ; ce qui signifie qu'on arrive à les résoudre en temps polynomial en la taille de l'entrée ;
- De (très) nombreux autres problèmes sont "difficiles" : on ne connaît pas d'algorithme efficace pour les résoudre ;
- Aujourd'hui, on va voir comment s'y prendre pour résoudre des problèmes a priori difficiles ;

Techniques pour les problèmes difficiles

Comme on l'a vu, on doit généralement choisir entre :

Techniques pour les problèmes difficiles

Comme on l'a vu, on doit généralement choisir entre :

1. résoudre le problème **vite** :

Techniques pour les problèmes difficiles

Comme on l'a vu, on doit généralement choisir entre :

1. résoudre le problème **vite** :

2. résoudre le problème **bien** :

Techniques pour les problèmes difficiles

Comme on l'a vu, on doit généralement choisir entre :

1. résoudre le problème **vite** :
   - heuristiques ;

2. résoudre le problème **bien** :

Techniques pour les problèmes difficiles

Comme on l'a vu, on doit généralement choisir entre :

1. résoudre le problème **vite** :
   - heuristiques ;
   - approximations avec garanties ;
2. résoudre le problème **bien** :

Techniques pour les problèmes difficiles

Comme on l'a vu, on doit généralement choisir entre :

1. résoudre le problème **vite** :
   - heuristiques ;
   - approximations avec garanties ;
   - ...
2. résoudre le problème **bien** :

Techniques pour les problèmes difficiles

Comme on l'a vu, on doit généralement choisir entre :

1. résoudre le problème **vite** :
   - heuristiques ;
   - approximations avec garanties ;
   - ...
2. résoudre le problème **bien** :
   - programmation dynamique ;

Techniques pour les problèmes difficiles

Comme on l'a vu, on doit généralement choisir entre :

1. résoudre le problème **vite** :
   - heuristiques ;
   - approximations avec garanties ;
   - ...
2. résoudre le problème **bien** :
   - programmation dynamique ;
   - recherche exhaustive ;

Techniques pour les problèmes difficiles

Comme on l'a vu, on doit généralement choisir entre :

1. résoudre le problème **vite** :
    - heuristiques ;
    - approximations avec garanties ;
    - ...
2. résoudre le problème **bien** :
    - programmation dynamique ;
    - recherche exhaustive ;
    - **branch and bound** ;

Techniques pour les problèmes difficiles

Comme on l'a vu, on doit généralement choisir entre :

1. résoudre le problème **vite** :
   - heuristiques ;
   - approximations avec garanties ;
   - ...
2. résoudre le problème **bien** :
   - programmation dynamique ;
   - recherche exhaustive ;
   - **branch and bound** ;
   - SAT, programmation linéaire, ... (voir l'an prochain)

Techniques pour les problèmes difficiles

Comme on l'a vu, on doit généralement choisir entre :

1. résoudre le problème **vite** :
    - heuristiques ;
    - approximations avec garanties ;
    - ...
2. résoudre le problème **bien** :
    - programmation dynamique ;
    - recherche exhaustive ;
    - **branch and bound** ;
    - SAT, programmation linéaire, ... (voir l'an prochain)
    - algorithmes paramétrés ;

Techniques pour les problèmes difficiles

Comme on l'a vu, on doit généralement choisir entre :

1. résoudre le problème **vite** :
   - heuristiques ;
   - approximations avec garanties ;
   - ...
2. résoudre le problème **bien** :
   - programmation dynamique ;
   - recherche exhaustive ;
   - **branch and bound** ;
   - SAT, programmation linéaire, ... (voir l'an prochain)
   - algorithmes paramétrés ;
   - ...

Techniques pour les problèmes difficiles

- On va voir aujourd'hui comment résoudre des problèmes difficiles de manière exacte ;

Techniques pour les problèmes difficiles

- On va voir aujourd'hui comment résoudre des problèmes difficiles de manière exacte ;
- La technique connue sous le nom de **branch and bound** [3] est très générale et s'applique à beaucoup de problèmes ;

Techniques pour les problèmes difficiles

- On va voir aujourd'hui comment résoudre des problèmes difficiles de manière exacte ;
- La technique connue sous le nom de **branch and bound** [3] est très générale et s'applique à beaucoup de problèmes ;
- On va l'illustrer sur deux problèmes difficiles célèbres : VERTEX COVER et TRAVELING SALESPERSON (TSP) ;

Le problème VERTEX COVER

VERTEX COVER

Entrée : un graphe (non orienté, non pondéré) $G = (V, E)$;
Sortie : un sous-ensemble $U \subseteq V$ tel que :

Le problème VERTEX COVER

VERTEX COVER

Entrée : un graphe (non orienté, non pondéré) $G = (V, E)$;
Sortie : un sous-ensemble $U \subseteq V$ tel que :

1. $\forall \{u, v\} \in E : |\{u, v\} \cap U| \geq 1$;

Le problème VERTEX COVER

VERTEX COVER

Entrée : un graphe (non orienté, non pondéré) $G = (V, E)$;
Sortie : un sous-ensemble $U \subseteq V$ tel que :

1. $\forall \{u, v\} \in E : |\{u, v\} \cap U| \geq 1$;
2. $|U|$ est minimum ;

Le problème VERTEX COVER

VERTEX COVER

Entrée : un graphe (non orienté, non pondéré) $G = (V, E)$;
Sortie : un sous-ensemble $U \subseteq V$ tel que :

1. $\forall \{u, v\} \in E : |\{u, v\} \cap U| \geq 1$;
2. $|U|$ est minimum ;

Autrement dit : U *couvre* toutes les arêtes de G.

Le problème VERTEX COVER

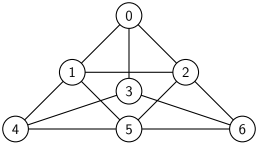
VERTEX COVER

Entrée : un graphe (non orienté, non pondéré) $G = (V, E)$;
Sortie : un sous-ensemble $U \subseteq V$ tel que :

1. $\forall \ \{u, v\} \in E : |\{u, v\} \cap U| \geq 1$;
2. $|U|$ est minimum ;

Autrement dit : U *couvre* toutes les arêtes de G.

Exemple 1



Le problème VERTEX COVER

VERTEX COVER

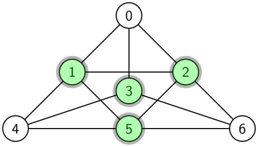
Entrée : un graphe (non orienté, non pondéré) $G = (V, E)$;

Sortie : un sous-ensemble $U \subseteq V$ tel que :

1. $\forall \{u, v\} \in E : |\{u, v\} \cap U| \geq 1$;
2. $|U|$ est minimum ;

Autrement dit : U *couvre* toutes les arêtes de G.

Exemple 1



VERTEX COVER

Le problème VERTEX COVER n'a pas l'air méchant ; mais :

VERTEX COVER

Le problème VERTEX COVER n'a pas l'air méchant ; mais :

- il est NP-complet $\Rightarrow$ l'existence d'un algorithme polynomial est très peu probable ;

VERTEX COVER

Le problème VERTEX COVER n'a pas l'air méchant ; mais :

- il est NP-complet $\Rightarrow$ l'existence d'un algorithme polynomial est très peu probable ;
  - même si G est cubique, ou planaire de degré maximum 3 ;

VERTEX COVER

Le problème VERTEX COVER n'a pas l'air méchant ; mais :

- il est NP-complet ⇒ l'existence d'un algorithme polynomial est très peu probable ;
  - même si G est cubique, ou planaire de degré maximum 3 ;
- la meilleure approximation a un taux de 2 ;

VERTEX COVER

Le problème VERTEX COVER n'a pas l'air méchant ; mais :

- il est NP-complet $\Rightarrow$ l'existence d'un algorithme polynomial est très peu probable ;
  - même si G est cubique, ou planaire de degré maximum 3 ;
- la meilleure approximation a un taux de 2 ;
- il est très peu probable qu'on puisse faire mieux que 2 ;

VERTEX COVER

Le problème VERTEX COVER n'a pas l'air méchant ; mais :

- il est NP-complet $\Rightarrow$ l'existence d'un algorithme polynomial est très peu probable ;
  - même si G est cubique, ou planaire de degré maximum 3 ;
- la meilleure approximation a un taux de 2 ;
- il est très peu probable qu'on puisse faire mieux que 2 ;
  - on sait qu'on ne peut pas faire mieux que $\sqrt{2} \approx 1.4142\ldots$;

VERTEX COVER

Le problème VERTEX COVER n'a pas l'air méchant ; mais :

- il est NP-complet $\Rightarrow$ l'existence d'un algorithme polynomial est très peu probable ;
    - même si G est cubique, ou planaire de degré maximum 3 ;
- la meilleure approximation a un taux de 2 ;
- il est très peu probable qu'on puisse faire mieux que 2 ;
    - on sait qu'on ne peut pas faire mieux que $\sqrt{2} \approx 1.4142\ldots$;
- le meilleur algorithme paramétré est en $O(1.2738^k + kn)$ ($k =$ taille de la solution) ;

Recherche exhaustive

- La recherche exhaustive porte bien son nom : pour résoudre le problème donné, on passe en revue toutes les options possibles ;

Recherche exhaustive

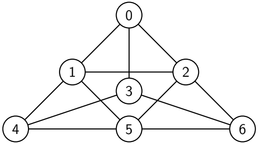
- La recherche exhaustive porte bien son nom : pour résoudre le problème donné, on passe en revue toutes les options possibles ;
- Le temps de calcul est bien souvent prohibitif ;

Recherche exhaustive

- La recherche exhaustive porte bien son nom : pour résoudre le problème donné, on passe en revue toutes les options possibles ;
- Le temps de calcul est bien souvent prohibitif ;
- Le branch and bound s'appuie sur ce principe, mais “en mieux” ;

Recherche exhaustive

- La recherche exhaustive porte bien son nom : pour résoudre le problème donné, on passe en revue toutes les options possibles ;
- Le temps de calcul est bien souvent prohibitif ;
- Le branch and bound s'appuie sur ce principe, mais “en mieux” ;
- Illustrons son principe sur notre exemple :



Recherche exhaustive pour VERTEX COVER

- Écrivons un algorithme récursif pour résoudre VERTEX COVER ; quels sont les cas de base ?

Recherche exhaustive pour VERTEX COVER

- Écrivons un algorithme récursif pour résoudre VERTEX COVER ; quels sont les cas de base ?
  - s'il n'y a pas d'arête, il n'y a rien à couvrir $\Rightarrow$ renvoyer $\emptyset$;

Recherche exhaustive pour VERTEX COVER

- Écrivons un algorithme récursif pour résoudre VERTEX COVER ; quels sont les cas de base ?
  - s'il n'y a pas d'arête, il n'y a rien à couvrir $\Rightarrow$ renvoyer $\emptyset$;
- Que faire dans le cas général ?

Recherche exhaustive pour VERTEX COVER

- Écrivons un algorithme récursif pour résoudre VERTEX COVER ; quels sont les cas de base ?
  - s'il n'y a pas d'arête, il n'y a rien à couvrir $\Rightarrow$ renvoyer $\emptyset$;
- Que faire dans le cas général ?
  - chaque sommet peut a priori faire partie de la solution ;

Recherche exhaustive pour VERTEX COVER

- Écrivons un algorithme récursif pour résoudre VERTEX COVER ; quels sont les cas de base ?
  - s'il n'y a pas d'arête, il n'y a rien à couvrir $\Rightarrow$ renvoyer $\emptyset$;
- Que faire dans le cas général ?
  - chaque sommet peut a priori faire partie de la solution ;
  - donc, pour chaque sommet v :

Recherche exhaustive pour VERTEX COVER

- Écrivons un algorithme récursif pour résoudre VERTEX COVER ; quels sont les cas de base ?
  - s'il n'y a pas d'arête, il n'y a rien à couvrir $\Rightarrow$ renvoyer $\emptyset$;
- Que faire dans le cas général ?
  - chaque sommet peut a priori faire partie de la solution ;
  - donc, pour chaque sommet v :
    - on construit la solution partielle $\{v\}$;

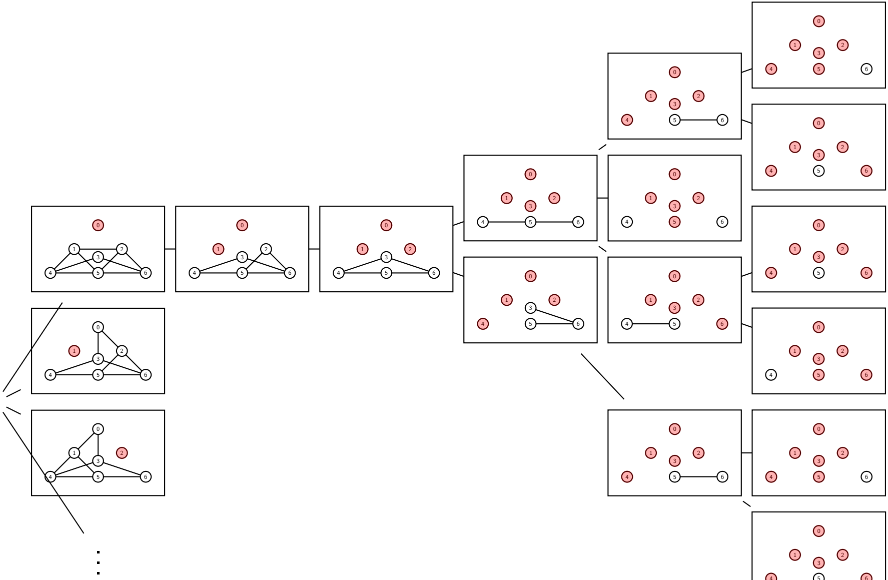
Recherche exhaustive pour VERTEX COVER

- Écrivons un algorithme récursif pour résoudre VERTEX COVER ; quels sont les cas de base ?
  - s'il n'y a pas d'arête, il n'y a rien à couvrir $\Rightarrow$ renvoyer $\emptyset$;
- Que faire dans le cas général ?
  - chaque sommet peut a priori faire partie de la solution ;
  - donc, pour chaque sommet v :
    - on construit la solution partielle $\{v\}$;
    - on cherche une solution optimale S' pour le graphe résiduel $G' = G \setminus \{v\}$;

Recherche exhaustive pour VERTEX COVER

- Écrivons un algorithme récursif pour résoudre VERTEX COVER ; quels sont les cas de base ?
  - s'il n'y a pas d'arête, il n'y a rien à couvrir $\Rightarrow$ renvoyer $\emptyset$;
- Que faire dans le cas général ?
  - chaque sommet peut a priori faire partie de la solution ;
  - donc, pour chaque sommet v :
    - on construit la solution partielle $\{v\}$;
    - on cherche une solution optimale S' pour le graphe résiduel $G' = G \setminus \{v\}$;
    - et on retient $\{v\} \cup S'$ si sa taille est inférieure à ce qu'on connaît déjà ;

Schéma de la recherche exhaustive



Recherche exhaustive pour VERTEX COVER

En pseudocode, cela donne :

Algorithme 1 : $\textsc{VCExhaustif}(G)$

Entrées : un graphe G.
Sortie : un "vertex cover" de taille minimum pour G.

si G*.nombre_arêtes() = 0* **alors renvoyer** $\emptyset$;
$S \leftarrow G$.sommets();
pour chaque $v \in G$*.sommets()* **faire**
    $S' \leftarrow \{v\} \cup \textsc{VCExhaustif}(G \setminus \{v\})$;
    si $|S'| < |S|$ **alors** $S \leftarrow S'$;
renvoyer S;

Complexité :

Recherche exhaustive pour VERTEX COVER

En pseudocode, cela donne :

Algorithme 1 : $\textsc{VCExhaustif}(G)$

Entrées : un graphe G.
Sortie : un "vertex cover" de taille minimum pour G.

si G*.nombre_arêtes() = 0* **alors renvoyer** $\emptyset$;
$S \leftarrow G$.sommets();
pour chaque $v \in G$*.sommets()* **faire**
  $S' \leftarrow \{v\} \cup \textsc{VCExhaustif}(G \setminus \{v\})$;
  si $|S'| < |S|$ **alors** $S \leftarrow S'$;
renvoyer S;

Complexité : cet algorithme passe en revue toutes les permutations de V, donc on a du $O(|V|!)$.

Recherche exhaustive pour VERTEX COVER

En pseudocode, cela donne :

Algorithme 1 : $\textsc{VCExhaustif}(G)$

Entrées : un graphe G.
Sortie : un "vertex cover" de taille minimum pour G.

si $G.nombre\_arêtes() = 0$ **alors renvoyer** $\emptyset$;
$S \leftarrow G$.sommets();
pour chaque $v \in G.sommets()$ **faire**
  $S' \leftarrow \{v\} \cup \textsc{VCExhaustif}(G \setminus \{v\})$;
  si $|S'| < |S|$ **alors** $S \leftarrow S'$;
renvoyer S;

Complexité : cet algorithme passe en revue toutes les permutations de V, donc on a du $O(|V|!)$.

On peut faire mieux en générant tous les sous-ensembles de toutes les tailles ; mais ça donne quand même du $\sum_{k=1}^{n} \left\{ {n \atop k} \right\} = O(2^n)$.

Borner la taille d'une solution optimale

- Dans tous les problèmes d'optimisation, il est important de pouvoir borner la taille d'une solution ;

Borner la taille d'une solution optimale

- Dans tous les problèmes d'optimisation, il est important de pouvoir borner la taille d'une solution ;
- Si $OPT(\cdot)$ désigne la taille d'une solution optimale, alors :

Borner la taille d'une solution optimale

- Dans tous les problèmes d'optimisation, il est important de pouvoir borner la taille d'une solution ;
- Si $OPT(\cdot)$ désigne la taille d'une solution optimale, alors :
  1. une **minoration** est une fonction m telle que pour toute instance I, on a $m(I) \leq OPT(I)$;

Borner la taille d'une solution optimale

- Dans tous les problèmes d'optimisation, il est important de pouvoir borner la taille d'une solution ;
- Si $OPT(\cdot)$ désigne la taille d'une solution optimale, alors :
  1. une **minoration** est une fonction m telle que pour toute instance I, on a $m(I) \leq OPT(I)$;
  2. une **majoration** est une fonction M telle que pour toute instance I, on a $M(I) \geq OPT(I)$;

Borner la taille d'une solution optimale

- Dans tous les problèmes d'optimisation, il est important de pouvoir borner la taille d'une solution ;
- Si $OPT(\cdot)$ désigne la taille d'une solution optimale, alors :
  1. une **minoration** est une fonction m telle que pour toute instance I, on a $m(I) \leq OPT(I)$;
  2. une **majoration** est une fonction M telle que pour toute instance I, on a $M(I) \geq OPT(I)$;
- Idéalement, on veut que $m(\cdot)$ renvoie la plus grande valeur possible, et que $M(\cdot)$ renvoie la plus petite valeur possible ;

Limiter l'espace de recherche

- Si $m(I) = M(I)$, alors on connaît $OPT(I)$;

Limiter l'espace de recherche

- Si $m(I) = M(I)$, alors on connaît $OPT(I)$;
- Mais même en cas d'inégalité, ces quantités nous permettent de réduire l'espace de recherche ;

Limiter l'espace de recherche

- Si $m(I) = M(I)$, alors on connaît $OPT(I)$;
- Mais même en cas d'inégalité, ces quantités nous permettent de réduire l'espace de recherche ;
- Pour une instance I de VERTEX COVER, on n'examine "que" les sous-ensembles de V de taille $m(I), m(I) + 1, \ldots, M(I)$;

Limiter l'espace de recherche

- Si $m(I) = M(I)$, alors on connaît $OPT(I)$;
- Mais même en cas d'inégalité, ces quantités nous permettent de réduire l'espace de recherche ;
- Pour une instance I de VERTEX COVER, on n'examine "que" les sous-ensembles de V de taille $m(I), m(I)+1, \ldots, M(I)$;
- Il reste à trouver des bornes satisfaisantes ;

Limiter l'espace de recherche

- Si $m(I) = M(I)$, alors on connaît $OPT(I)$;
- Mais même en cas d'inégalité, ces quantités nous permettent de réduire l'espace de recherche ;
- Pour une instance I de VERTEX COVER, on n'examine "que" les sous-ensembles de V de taille $m(I), m(I) + 1, \ldots, M(I)$;
- Il reste à trouver des bornes satisfaisantes ;
- Dans le cas de VERTEX COVER, les **couplages** qu'on déjà vus vont nous aider ;

Le rôle des couplages

Rappel (cf. cours sur les flots) : un **couplage** dans un graphe $G = (V, E)$ est un ensemble $F \subseteq E$ d'arêtes deux à deux disjointes ; il est :

1. **maximal** si on ne peut pas lui rajouter d'arêtes,
2. **maximum** s'il n'existe aucun autre couplage possédant plus d'arêtes.

Le rôle des couplages

Rappel (cf. cours sur les flots) : un **couplage** dans un graphe $G = (V, E)$ est un ensemble $F \subseteq E$ d'arêtes deux à deux disjointes ; il est :

1. **maximal** si on ne peut pas lui rajouter d'arêtes,
2. **maximum** s'il n'existe aucun autre couplage possédant plus d'arêtes.

Lemme 1

Les sommets couverts par un couplage maximal M forment une solution valide pour VERTEX COVER.

Le rôle des couplages

Rappel (cf. cours sur les flots) : un **couplage** dans un graphe $G = (V, E)$ est un ensemble $F \subseteq E$ d'arêtes deux à deux disjointes ; il est :

1. **maximal** si on ne peut pas lui rajouter d'arêtes,
2. **maximum** s'il n'existe aucun autre couplage possédant plus d'arêtes.

Lemme 1

Les sommets couverts par un couplage maximal M forment une solution valide pour VERTEX COVER*.*

Démonstration.

Si M est maximal, on ne peut par définition pas lui rajouter d'arête ; donc chaque arête possède une extrémité dans M, et sélectionner tous les sommets de M permet de couvrir toutes les arêtes du graphe. □

Le rôle des couplages

Rappel (cf. cours sur les flots) : un **couplage** dans un graphe $G = (V, E)$ est un ensemble $F \subseteq E$ d'arêtes deux à deux disjointes ; il est :

1. **maximal** si on ne peut pas lui rajouter d'arêtes,
2. **maximum** s'il n'existe aucun autre couplage possédant plus d'arêtes.

Lemme 1

Les sommets couverts par un couplage maximal M forment une solution valide pour VERTEX COVER*.*

Démonstration.

Si M est maximal, on ne peut par définition pas lui rajouter d'arête ; donc chaque arête possède une extrémité dans M, et sélectionner tous les sommets de M permet de couvrir toutes les arêtes du graphe. $\square$

$\Rightarrow$ pour toute solution S de VERTEX COVER, on a $|S| \leq 2|M|$.

Le rôle des couplages

Lemme 2

Soit S une solution pour VERTEX COVER*, et M un couplage maximal ; alors $|S| \geq |M|$.*

Le rôle des couplages

Lemme 2

Soit S une solution pour VERTEX COVER*, et M un couplage maximal ; alors $|S| \geq |M|$.*

Démonstration.

Si M est un couplage maximal, ses sommets couvrent toutes les arêtes du graphe. On est donc obligé de sélectionner au moins une extrémité de chaque élément de M pour couvrir toutes les arêtes. $\square$

Le rôle des couplages

Lemme 2

Soit S une solution pour VERTEX COVER*, et M un couplage maximal ; alors $|S| \geq |M|$.*

Démonstration.

Si M est un couplage maximal, ses sommets couvrent toutes les arêtes du graphe. On est donc obligé de sélectionner au moins une extrémité de chaque élément de M pour couvrir toutes les arêtes. $\square$

$\Rightarrow$ pour toute solution S de VERTEX COVER, on a $|S| \geq |M|$. Donc :

Le rôle des couplages

Lemme 2

Soit S une solution pour VERTEX COVER, *et M un couplage maximal ; alors $|S| \geq |M|$.*

Démonstration.

Si M est un couplage maximal, ses sommets couvrent toutes les arêtes du graphe. On est donc obligé de sélectionner au moins une extrémité de chaque élément de M pour couvrir toutes les arêtes. □

$\Rightarrow$ pour toute solution S de VERTEX COVER, on a $|S| \geq |M|$. Donc :

- même la recherche exhaustive peut se limiter aux sous-ensembles de taille $|M|, |M| + 1, \ldots, 2|M|$.

Le rôle des couplages

Lemme 2

Soit S une solution pour VERTEX COVER, *et M un couplage maximal ; alors $|S| \geq |M|$.*

Démonstration.

Si M est un couplage maximal, ses sommets couvrent toutes les arêtes du graphe. On est donc obligé de sélectionner au moins une extrémité de chaque élément de M pour couvrir toutes les arêtes. □

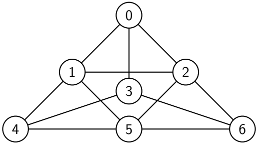
$\Rightarrow$ pour toute solution S de VERTEX COVER, on a $|S| \geq |M|$. Donc :

- même la recherche exhaustive peut se limiter aux sous-ensembles de taille $|M|, |M| + 1, \ldots, 2|M|$.
- la recherche d'un couplage maximal nous donne une 2-approximation pour VERTEX COVER, puisque $|M| \leq |S| \leq 2|M|$ pour toute solution S.

Couplages et VERTEX COVER

Reprenons comme exemple le graphe déjà vu :

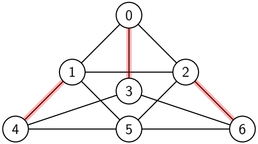
Exemple 2



Couplages et VERTEX COVER

Reprenons comme exemple le graphe déjà vu :

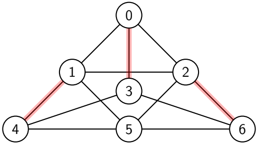
Exemple 2



Couplages et VERTEX COVER

Reprenons comme exemple le graphe déjà vu :

Exemple 2

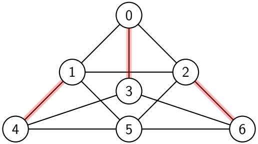


- Les extrémités du couplage nous donnent la solution $\{0, 1, 2, 3, 4, 6\}$;

Couplages et VERTEX COVER

Reprenons comme exemple le graphe déjà vu :

Exemple 2



- Les extrémités du couplage nous donnent la solution $\{0, 1, 2, 3, 4, 6\}$;
- On doit en prendre au moins la moitié, donc on examinera les sous-ensembles de taille 3, 4, 5, 6 ;

Comment trouver un couplage maximal ?

- On peut trouver un couplage **maximum** en $O(|E|\sqrt{|V|})$ [4] ;

Comment trouver un couplage maximal ?

- On peut trouver un couplage **maximum** en $O(|E|\sqrt{|V|})$ [4] ;
- Mais nous pouvons nous contenter d'un couplage **maximal**, que l'on peut obtenir comme suit :

Comment trouver un couplage maximal ?

- On peut trouver un couplage **maximum** en $O(|E|\sqrt{|V|})$ [4] ;
- Mais nous pouvons nous contenter d'un couplage **maximal**, que l'on peut obtenir comme suit :

  1. on démarre avec un couplage M vide ;

Comment trouver un couplage maximal ?

- On peut trouver un couplage **maximum** en $O(|E|\sqrt{|V|})$ [4] ;
- Mais nous pouvons nous contenter d'un couplage **maximal**, que l'on peut obtenir comme suit :
  1. on démarre avec un couplage M vide ;
  2. tant que E n'est pas vide : extraire une arête $\{u, v\}$ que l'on rajoute à M, et supprimer u et v ;

Comment trouver un couplage maximal ?

- On peut trouver un couplage **maximum** en $O(|E|\sqrt{|V|})$ [4] ;
- Mais nous pouvons nous contenter d'un couplage **maximal**, que l'on peut obtenir comme suit :
  1. on démarre avec un couplage M vide ;
  2. tant que E n'est pas vide : extraire une arête $\{u, v\}$ que l'on rajoute à M, et supprimer u et v ;
- Complexité : $O(|E|)$;

Principe du branch and bound

- Le branch and bound suit le schéma de la recherche exhaustive, mais en s'arrêtant de manière anticipée ;

Principe du branch and bound

- Le branch and bound suit le schéma de la recherche exhaustive, mais en s'arrêtant de manière anticipée ;
- La minoration nous permet d'éviter l'exploration des solutions non prometteuses ;

Principe du branch and bound

- Le branch and bound suit le schéma de la recherche exhaustive, mais en s'arrêtant de manière anticipée ;
- La minoration nous permet d'éviter l'exploration des solutions non prometteuses ;
- La majoration nous permet de démarrer avec une meilleure solution que l'approche naïve (de taille $|V|$ pour VERTEX COVER) ;

Principe du branch and bound

- Le branch and bound suit le schéma de la recherche exhaustive, mais en s'arrêtant de manière anticipée ;
- La minoration nous permet d'éviter l'exploration des solutions non prometteuses ;
- La majoration nous permet de démarrer avec une meilleure solution que l'approche naïve (de taille $|V|$ pour VERTEX COVER) ;
- Et si la taille d'une solution S atteint la minoration, on sait qu'elle est optimale et on arrête les recherches ;

Rôle de la minoration

- La minoration nous permet d'éliminer certains candidats non prometteurs ;

Rôle de la minoration

- La minoration nous permet d'éliminer certains candidats non prometteurs ;
- Le raisonnement est le suivant (pour un problème de minimisation) :

Rôle de la minoration

- La minoration nous permet d'éliminer certains candidats non prometteurs ;
- Le raisonnement est le suivant (pour un problème de minimisation) :
  - à chaque étape de l'algorithme, on a une solution S pour l'instance I ;

Rôle de la minoration

- La minoration nous permet d'éliminer certains candidats non prometteurs ;
- Le raisonnement est le suivant (pour un problème de minimisation) :
  - à chaque étape de l'algorithme, on a une solution S pour l'instance I ;
  - on construit une nouvelle solution partielle T, et il nous reste donc à résoudre un sous-problème sur l'instance I' ;

Rôle de la minoration

- La minoration nous permet d'éliminer certains candidats non prometteurs ;
- Le raisonnement est le suivant (pour un problème de minimisation) :
  - à chaque étape de l'algorithme, on a une solution S pour l'instance I ;
  - on construit une nouvelle solution partielle T, et il nous reste donc à résoudre un sous-problème sur l'instance I' ;
  - on n'examine I' que si $|T| + m(I') < |S|$;

Rôle de la minoration pour VERTEX COVER

Dans le cas de VERTEX COVER ;

- on a une solution candidate S donnée par les extrémités du couplage maximal M ;

Rôle de la minoration pour VERTEX COVER

Dans le cas de VERTEX COVER ;

- on a une solution candidate S donnée par les extrémités du couplage maximal M ;
- on examine tous les sous-ensembles $T \subseteq V$ de taille $|M|$;

Rôle de la minoration pour VERTEX COVER

Dans le cas de VERTEX COVER ;

- on a une solution candidate S donnée par les extrémités du couplage maximal M ;
- on examine tous les sous-ensembles $T \subseteq V$ de taille $|M|$;
- on construit la nouvelle instance $G' = G \setminus T$, et l'on calcule un couplage maximal M' sur G' ;

Rôle de la minoration pour VERTEX COVER

Dans le cas de VERTEX COVER ;

- on a une solution candidate S donnée par les extrémités du couplage maximal M ;
- on examine tous les sous-ensembles $T \subseteq V$ de taille $|M|$;
- on construit la nouvelle instance $G' = G \setminus T$, et l'on calcule un couplage maximal M' sur G' ;
- on n'examine G' que si $|T| + |M'| < |S|$;

Branch and bound pour VERTEX COVER

En pseudocode, cela donne :

Algorithme 2 : $\textsc{VCBranchAndBound}(G)$

Entrées : un graphe G.
Sortie : un “vertex cover” de taille minimum pour G.

si G*.nombre_arêtes()* $=$ *0* **alors renvoyer** $\emptyset$;
$M \leftarrow$ couplage maximal pour G;
$S \leftarrow$ les sommets de M;
minoration $\leftarrow |M|$;
pour chaque *sous-ensemble* T *de taille* $|M|$ *de* G*.sommets()* **faire**
  si $|S| =$ *minoration* **alors renvoyer** S;
  $G' \leftarrow G \setminus T$;
  $M' \leftarrow$ couplage maximal pour G';
  si $|T| + |M'| < |S|$ **alors** `// solution prometteuse`
    $S' \leftarrow T \cup \textsc{VCBranchAndBound}(G')$;
    si $|S'| < |S|$ **alors** $S \leftarrow S'$;
renvoyer S;

VOYAGEUR DE COMMERCE

Prenons comme exemple le problème suivant :

TRAVELING SALESPERSON (VOYAGEUR DE COMMERCE)

Entrée : un graphe non orienté pondéré $G = (V, E, w)$;
Sortie : un chemin P tel que :

VOYAGEUR DE COMMERCE

Prenons comme exemple le problème suivant :

TRAVELING SALESPERSON (VOYAGEUR DE COMMERCE)

Entrée : un graphe non orienté pondéré $G = (V, E, w)$;
Sortie : un chemin P tel que :

1. P est hamiltonien : on passe par chaque sommet exactement une fois ;

VOYAGEUR DE COMMERCE

Prenons comme exemple le problème suivant :

TRAVELING SALESPERSON (VOYAGEUR DE COMMERCE)

Entrée : un graphe non orienté pondéré $G = (V, E, w)$;

Sortie : un chemin P tel que :

1. P est hamiltonien : on passe par chaque sommet exactement une fois ;
2. P est de poids minimum ;

VOYAGEUR DE COMMERCE

Prenons comme exemple le problème suivant :

TRAVELING SALESPERSON (VOYAGEUR DE COMMERCE)

Entrée : un graphe non orienté pondéré $G = (V, E, w)$;
Sortie : un chemin P tel que :

1. P est hamiltonien : on passe par chaque sommet exactement une fois ;
2. P est de poids minimum ;

(variantes : trouver un *cycle* plutôt qu'un chemin ; utiliser un graphe orienté ; . . .)

TSP

Le problème TSP n'a pas l'air méchant ; mais :

TSP

Le problème TSP n'a pas l'air méchant ; mais :

- il est NP-complet $\Rightarrow$ l'existence d'un algorithme polynomial est très peu probable ;

TSP

Le problème TSP n'a pas l'air méchant ; mais :

- il est NP-complet $\Rightarrow$ l'existence d'un algorithme polynomial est très peu probable ;
  - même dans la version “euclidienne” (les poids correspondent à des distances) ;

TSP

Le problème TSP n'a pas l'air méchant ; mais :

- il est NP-complet $\Rightarrow$ l'existence d'un algorithme polynomial est très peu probable ;
  - même dans la version "euclidienne" (les poids correspondent à des distances) ;
- il n'est **pas approximable**, sauf dans le cas euclidien :

TSP

Le problème TSP n'a pas l'air méchant ; mais :

- il est NP-complet $\Rightarrow$ l'existence d'un algorithme polynomial est très peu probable ;
  - même dans la version "euclidienne" (les poids correspondent à des distances) ;
- il n'est **pas approximable**, sauf dans le cas euclidien :
  - il existe une 3/2-approximation [1] ;

TSP

Le problème TSP n'a pas l'air méchant ; mais :

- il est NP-complet $\Rightarrow$ l'existence d'un algorithme polynomial est très peu probable ;
  - même dans la version "euclidienne" (les poids correspondent à des distances) ;
- il n'est **pas approximable**, sauf dans le cas euclidien :
  - il existe une 3/2-approximation [1] ;
  - on ne peut pas faire mieux que $123/122 \approx 1.008\ldots$ [2] ;

Recherche exhaustive pour le VOYAGEUR DE COMMERCE

- Pour simplifier, supposons G complet (sinon : rajoutons les arêtes manquantes avec un poids $+\infty$) ;

Recherche exhaustive pour le VOYAGEUR DE COMMERCE

- Pour simplifier, supposons G complet (sinon : rajoutons les arêtes manquantes avec un poids $+\infty$) ;
- Une solution pour TSP est un ordonnancement des sommets de G ;

Recherche exhaustive pour le VOYAGEUR DE COMMERCE

- Pour simplifier, supposons G complet (sinon : rajoutons les arêtes manquantes avec un poids $+\infty$) ;
- Une solution pour TSP est un ordonnancement des sommets de G ;
- La recherche exhaustive examine donc toutes les permutations de V ;

Recherche exhaustive pour le VOYAGEUR DE COMMERCE

- Pour simplifier, supposons G complet (sinon : rajoutons les arêtes manquantes avec un poids $+\infty$) ;
- Une solution pour TSP est un ordonnancement des sommets de G ;
- La recherche exhaustive examine donc toutes les permutations de V ;
- La complexité est donc en $O(|V|!)$;

Recherche exhaustive pour le VOYAGEUR DE COMMERCE

- Pour simplifier, supposons G complet (sinon : rajoutons les arêtes manquantes avec un poids $+\infty$) ;
- Une solution pour TSP est un ordonnancement des sommets de G ;
- La recherche exhaustive examine donc toutes les permutations de V ;
- La complexité est donc en $O(|V|!)$;
- (si G est non orienté, on n'examine "que" la moitié des permutations, ce qui ne change rien à la complexité) ;

Minoration pour TSP

- On cherche un chemin de poids minimum couvrant tous les sommets . . .

Minoration pour TSP

- On cherche un chemin de poids minimum couvrant tous les sommets . . .
- . . . un chemin est un cas particulier d'arbre . . .

Minoration pour TSP

- On cherche un chemin de poids minimum couvrant tous les sommets ...
- ... un chemin est un cas particulier d'arbre ...
- ... et donc, le poids d'une solution optimale pour TSP est au moins celui d'un ACPM, qu'on est capable de trouver en temps polynomial ;

Majoration pour TSP

- Comme TSP n'est pas approximable, on ne trouvera pas de majoration comparable avec notre minoration basée sur les ACPM ;

Majoration pour TSP

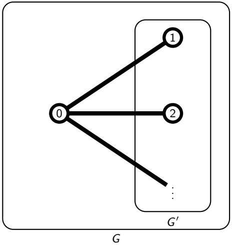
- Comme TSP n'est pas approximable, on ne trouvera pas de majoration comparable avec notre minoration basée sur les ACPM ;
- Cela ne nous empêche pas de trouver une solution initiale ;

Majoration pour TSP

- Comme TSP n'est pas approximable, on ne trouvera pas de majoration comparable avec notre minoration basée sur les ACPM ;
- Cela ne nous empêche pas de trouver une solution initiale ;
- Il suffit de prendre une permutation aléatoire des sommets, puisque le graphe est complet ;

Recherche récursive d'un chemin optimal

- Construisons un chemin optimal de manière récursive ;
- Partons du sommet 0 : si l'on obtient un chemin hamiltonien P' de u à v dans le graphe $G' = G \setminus \{0\}$, alors il suffit de relier une extrémité de P' à 0 pour obtenir un chemin hamiltonien pour G ;



- Comme on veut minimiser le poids de P, notre critère sera :

$$\min_{v \in N(0)} \left(w(\{0, v\}) + OPT(G \setminus \{0\}, v) \right)$$

Approche branch and bound pour TSP

- La minoration nous permet de savoir si G' vaut la peine d'être exploré ;
- Si le poids d'un ACPM sur G' que l'on connecte ensuite au sommet de départ u est trop élevé, alors on n'examine pas G' ;
- Plus formellement, si S est notre meilleure solution actuelle au départ de u et qu'on veut chercher une solution optimale dans G' au départ de v, on ne le fait que si

$$w(\{u, v\}) + |ACPM(G')| < w(S)$$

Branch and bound pour TSP

En pseudocode, cela donne :

Algorithme 3 : TSPBranchAndBound(G, départ)

Entrées : un graphe pondéré G, un sommet de départ.
Sortie : un chemin hamiltonien de poids minimum pour G démarrant au sommet de départ.

si $G.nombre\_sommets() \leq 1$ **alors renvoyer** G.sommets();
minoration $\leftarrow w(ACPM(G))$;
$S \leftarrow$ chemin hamiltonien aléatoire sur G démarrant en départ;
$G' \leftarrow G \setminus \{$départ$\}$;
$m \leftarrow w(ACPM(G'))$;
pour chaque $v \in G.voisins(départ)$ **faire**
    si $w(S) = minoration$ **alors renvoyer** S ;
    si $w(\{départ, v\}) + m < w(S)$ **alors**
        $S' \leftarrow$ (départ, v) + TSPBranchAndBound(G', v);
        si $w(S') < w(S)$ **alors** $S \leftarrow S'$;
renvoyer S;

Conception d'un algorithme branch and bound

On en tire les leçons suivantes sur la conception d'un algorithme de type branch and bound :

- il nous faut un "découpage" des solutions, qui permette :

Conception d'un algorithme branch and bound

On en tire les leçons suivantes sur la conception d'un algorithme de type branch and bound :

- il nous faut un "découpage" des solutions, qui permette :
  1. d'évaluer la qualité d'une solution partielle ;

Conception d'un algorithme branch and bound

On en tire les leçons suivantes sur la conception d'un algorithme de type branch and bound :

- il nous faut un "découpage" des solutions, qui permette :
  1. d'évaluer la qualité d'une solution partielle ;
  2. d'éviter le plus tôt possible de prolonger sa construction si elle n'est pas prometteuse ;

Conception d'un algorithme branch and bound

On en tire les leçons suivantes sur la conception d'un algorithme de type branch and bound :

- il nous faut un "découpage" des solutions, qui permette :
  1. d'évaluer la qualité d'une solution partielle ;
  2. d'éviter le plus tôt possible de prolonger sa construction si elle n'est pas prometteuse ;
- il nous faut également des bornes les plus "serrées" possible :

Conception d'un algorithme branch and bound

On en tire les leçons suivantes sur la conception d'un algorithme de type branch and bound :

- il nous faut un "découpage" des solutions, qui permette :
  1. d'évaluer la qualité d'une solution partielle ;
  2. d'éviter le plus tôt possible de prolonger sa construction si elle n'est pas prometteuse ;
- il nous faut également des bornes les plus "serrées" possible :
  1. une minoration, pour éliminer au plus vite les candidats non prometteurs **et** vérifier si notre solution actuelle est optimale ;

Conception d'un algorithme branch and bound

On en tire les leçons suivantes sur la conception d'un algorithme de type branch and bound :

- il nous faut un "découpage" des solutions, qui permette :
  1. d'évaluer la qualité d'une solution partielle ;
  2. d'éviter le plus tôt possible de prolonger sa construction si elle n'est pas prometteuse ;
- il nous faut également des bornes les plus "serrées" possible :
  1. une minoration, pour éliminer au plus vite les candidats non prometteurs **et** vérifier si notre solution actuelle est optimale ;
  2. une majoration, pour démarrer l'exploration avec une solution de qualité raisonnable ;

Peut-on faire mieux ?

- On peut souvent améliorer les performances de l'algorithme en pratique ; certaines instances intermédiaires peuvent être :

Peut-on faire mieux ?

- On peut souvent améliorer les performances de l'algorithme en pratique ; certaines instances intermédiaires peuvent être :
  - "faciles" $\Rightarrow$ résolvons-les directement ;

Peut-on faire mieux ?

- On peut souvent améliorer les performances de l'algorithme en pratique ; certaines instances intermédiaires peuvent être :
  - "faciles" $\Rightarrow$ résolvons-les directement ;
  - décomposables $\Rightarrow$ résolution en parallèle ;

Peut-on faire mieux ?

- On peut souvent améliorer les performances de l'algorithme en pratique ; certaines instances intermédiaires peuvent être :
  - "faciles" $\Rightarrow$ résolvons-les directement ;
  - décomposables $\Rightarrow$ résolution en parallèle ;
- Dans le cas de VERTEX COVER :

Peut-on faire mieux ?

- On peut souvent améliorer les performances de l'algorithme en pratique ; certaines instances intermédiaires peuvent être :
  - "faciles" $\Rightarrow$ résolvons-les directement ;
  - décomposables $\Rightarrow$ résolution en parallèle ;
- Dans le cas de VERTEX COVER :
  - le problème est facile à résoudre si G est biparti, ou complet ;

Peut-on faire mieux ?

- On peut souvent améliorer les performances de l'algorithme en pratique ; certaines instances intermédiaires peuvent être :
  - "faciles" $\Rightarrow$ résolvons-les directement ;
  - décomposables $\Rightarrow$ résolution en parallèle ;
- Dans le cas de VERTEX COVER :
  - le problème est facile à résoudre si G est biparti, ou complet ;
  - chaque composante connexe peut se résoudre en parallèle ;

Peut-on faire mieux ?

- On peut souvent améliorer les performances de l'algorithme en pratique ; certaines instances intermédiaires peuvent être :
  - "faciles" $\Rightarrow$ résolvons-les directement ;
  - décomposables $\Rightarrow$ résolution en parallèle ;
- Dans le cas de VERTEX COVER :
  - le problème est facile à résoudre si G est biparti, ou complet ;
  - chaque composante connexe peut se résoudre en parallèle ;
- Même si le graphe de départ n'est pas un "cas facile", de nombreuses sous-instances peuvent l'être ;

Complexité et performances en pratique

- Attention, la complexité reste exponentielle au pire cas ;
- L'algorithme sera d'autant plus rapide qu'on arrivera à évacuer rapidement des options non prometteuses ;
- Attention à la complexité des vérifications ;

Bibliographie

[1] Nicos Christofides.
Worst-case analysis of a new heuristic for the travelling salesman problem.
Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, February 1976.

[2] Marek Karpinski, Michael Lampis, and Richard Schmied.
New inapproximability bounds for tsp.
Journal of Computer and System Sciences, 81(8) :1665–1677, 2015.

[3] A. H. Land and A. G. Doig.
An automatic method of solving discrete programming problems.
Econometrica, 28(3) :497–520, 1960.

[4] Silvio Micali and Vijay V. Vazirani.
An $O(\sqrt{|V|}|E|)$ algorithm for finding maximum matching in general graphs.
In *21st Annual Symposium on Foundations of Computer Science, Syracuse, New York, USA, 13-15 October 1980*, pages 17–27. IEEE Computer Society, 1980.