Algorithmique avancée : Introduction à la complexité paramétrée

Marin Bougeret
marin.bougeret@lirmm.fr

January 29, 2025

POLYTECH
MONTPELLIER

Outline

Contexte

- Certains problèmes combinatoires se résolvent efficacement par DP (Sac à Dos, Partition de nombres, ..)
- Pour les autres, que faire ?

Contexte

- Certains problèmes combinatoires se résolvent efficacement par DP (Sac à Dos, Partition de nombres, ..)
- Pour les autres, que faire ?

VC_{dec} (NP-complet)

- entrée : un graphe G, un entier k
- question : décider si $opt(G) \leq k$ (cad si il existe k sommets couvrant toutes les arêtes)

Contexte

- Certains problèmes combinatoires se résolvent efficacement par DP (Sac à Dos, Partition de nombres, ..)
- Pour les autres, que faire ?

Résolution de VC_{dec}

Si le paramètre k est petit, on peut avoir un algorithme pas si mauvais :

- Algorithme naïf (XP) en $\approx \mathcal{O}(n^k)$
- Algoirthme intelligent (FPT) en $\approx \mathcal{O}(2^k poly(n))$

Complexité paramétrée

Idée : si en pratique un des paramètres $p(I)$ de l'instance est petit :

- VC_{dec} dans un graphe de $n = 1000$ sommets je cherche un VC de taille au plus $k = 5$ (paramètre : k)
- SS_{dec} parmis $n = 1000$ objets et $C = 40000$, il n'y a en fait que 3 tailles d'objets (paramètre : #tailles)
- Bio informatique : Instances de reconstruction de chaînes d'ADN ont treewidth ≤ 11
- Robotique : le nombre de degrés de liberté dans des problèmes de planification de mouvements est ≤ 10.
- Compilateurs : Vérifier la compatibilité des déclarations de types est difficile, mais en pratique la profondeur des déclarations de types est ≤ 10

Contexte

Comment en tirer avantage ? On pourrait envisager :

- Un algorithme de complexité $\mathcal{O}(n^{p(I)})$ (ou plus généralement $\mathcal{O}(n^{f(p(I))})$ pour n'importe quelle fonction f). On parlera d'algorithme/problème XP.
- Encore mieux : un algorithme de complexité $\mathcal{O}(2^{p(I)}poly(n))$ (ou plus généralement $\mathcal{O}(f(p(I)poly(n)))$ pour n'importe quelle fonction f). On parlera d'algorithme/problème FPT.

Ces algorithmes ne sont pas polynomiaux en général (ex quand $p(I) = n/5$), mais le deviennent quand $p(I)$ est une constante.

Outline

Algorithme XP

Rappel

Le problème VC_{dec}/k est

- entrée : un graphe G, un entier k
- paramètre : k
- question : décider si $opt(G) \leq k$ (cad si il existe k sommets couvrant toutes les arêtes)

Algorithme XP

Rappel

Le problème VC_{dec}/k est

- entrée : un graphe G, un entier k
- paramètre : k
- question : décider si $opt(G) \leq k$ (cad si il existe k sommets couvrant toutes les arêtes)

But

Résoudre VC_{dec} en temps $\mathcal{O}(n^k poly(n))$

Algorithme XP

Un algorithme XP pour VC_{dec}

- Pour tout ensemble S de k sommets, vérfier (en temps poly, par exemple $\mathcal{O}(n^2)$) que S est un VC (couvre toutes les arêtes).
- Complexité $\mathcal{O}(n^k n^2)$

(nous discuterons plus tard l'implémentation de ce genre d'algorithme).

Et si on cherchait un algorithme plus rapide pour VC_{dec}/k :

But

Résoudre VC_{dec} en temps $\mathcal{O}(f(p(I))poly(|I|))$ pour une certaine fonction f

Algorithme XP

Un algorithme XP pour VC_{dec}

- Pour tout ensemble S de k sommets, vérfier (en temps poly, par exemple $\mathcal{O}(n^2)$) que S est un VC (couvre toutes les arêtes).
- Complexité $\mathcal{O}(n^k n^2)$

(nous discuterons plus tard l'implémentation de ce genre d'algorithme).

Et si on cherchait un algorithme plus rapide pour VC_{dec}/k :

But

Résoudre VC_{dec} en temps $\mathcal{O}(f(k)poly(n))$ pour une certaine fonction f

Algorithme XP

Un algorithme XP pour VC_{dec}

- Pour tout ensemble S de k sommets, vérfier (en temps poly, par exemple $\mathcal{O}(n^2)$) que S est un VC (couvre toutes les arêtes).
- Complexité $\mathcal{O}(n^k n^2)$

(nous discuterons plus tard l'implémentation de ce genre d'algorithme).

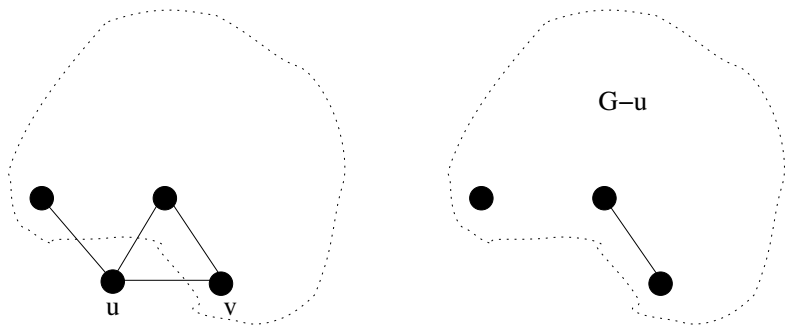
Et si on cherchait un algorithme plus rapide pour VC_{dec}/k :

But

Résoudre VC_{dec} en temps $\mathcal{O}(2^k poly(n))$

Algorithme FPT

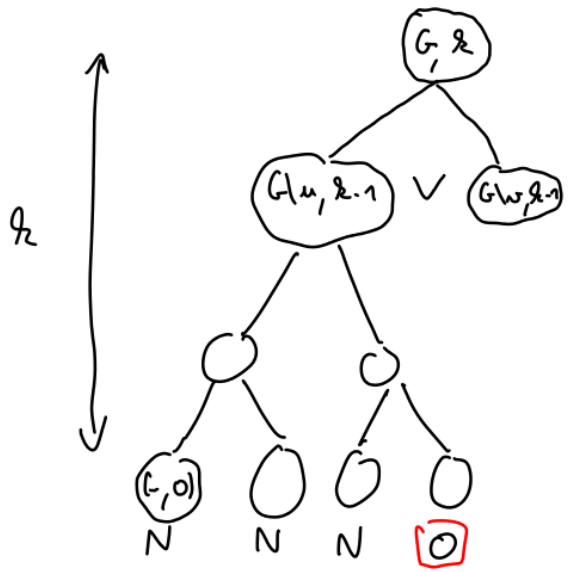
But : On va écrire un algo. $A(G, k)$ qui décide si il existe un VC de taille au plus k.
Observation :Soit $e = (u, v)$ une arête de G, alors dans toute solution S, u OU v doit être dans S
Algorithme :Trouver une arête $e = (u, v)$, et donc
$A(G, k)$ retourne $A(G - u, k - 1)$ OU $A(G - v, k - 1)$



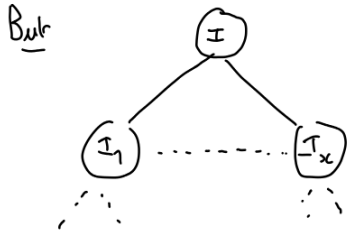
Ecriture de l'algo et analyse :

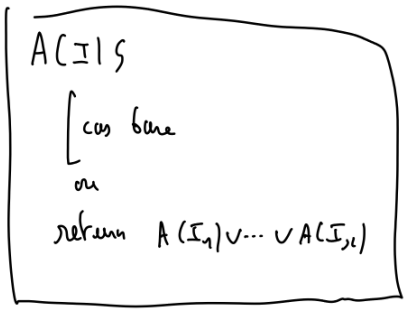
Un algorithme plus rapide pour VC_{dec}.

```
bool A(G,k){// k >= 0
  //retourne vrai ssi existe un VC de taille
      <= k
  if(E(G)=vide)// G n'a pas d'arete
    return true
  if(k=0)
    return false
  soit uv une arete de G
  return A(G-u,k-1) || A(G-v,k-1)
}
```



Preuve



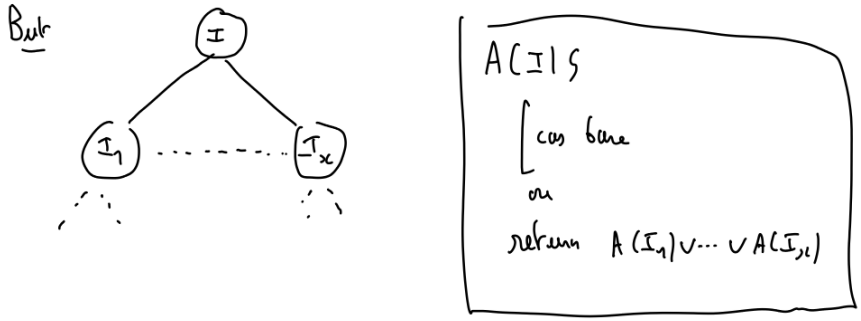


Branchement correct

Etant donné un algorithme récursif $A(I)$ dit "de branchement" (qui soit répond avec un cas de base, soit répond $A(I_1)$ OU .. OU $A(I_x)$), donc dit que le branchement est correct si

- les cas de base sont corrects (donnent la bonne réponse)
- I est une OUI instance $\Leftrightarrow \exists i$ tq I_i est une OUI instance

Preuve



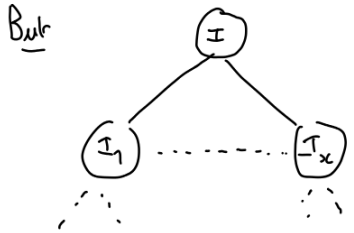
Propriété

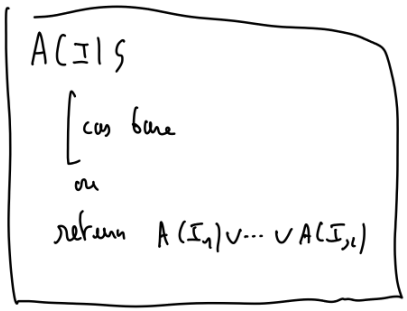
Si le branchement est correct, alors l'algorithme est correct (cad $A(I)$ répond OUI ssi I oui instance)

Preuve par récurrence sur profondeur arbre

- si I traitée par un cas de base, ok
- sinon
  - si I OUI instance, alors $\exists i$ tq I_i est une OUI instance, par récurrence $A(I_i)$ répondra OUI, et donc $A(I)$ OUI
  - si I NON instance, alors $\forall i$, I_i est une NON instance, par récurrence tous les $A(I_i)$ répondront NON, et donc $A(I)$ NON

Preuve





Conclusion

Pour un algorithme de branchement, on se contentera donc de démontrer que le branchement est correct.

Un algorithme plus rapide pour VC_{dec}.

```
bool A(G,k){// k >= 0
  if(E(G)=vide)// G n'a pas d'arete
    return true
  if(k=0)
    return false
  soit uv une arete de G
  return A(G-u,k-1) || A(G-v,k-1)
}
```

Branchement correct ?

Cas de base corrects : ok

Un algorithme plus rapide pour VC_{dec}.

```
bool A(G,k){// k >= 0
  if(E(G)=vide)// G n'a pas d'arete
    return true
  if(k=0)
    return false
  soit uv une arete de G
  return A(G-u,k-1) || A(G-v,k-1)
}
```

Branchement correct ?

I OUI instance $\Rightarrow \exists i$ tq I_i est une OUI instance ?

- Supposons $I = (G, k)$ OUI instance. Donc G admet un VC S de taille au plus k.
- Soit uv une arête de G. Alors $u \in S$ ou $v \in S$.
- Si $u \in S$, alors $S \setminus u$ doit être un VC de $G - u$. Donc $G - u$ admet un VC de taille au plus $k - 1$: $(G - u, k - 1)$ est une OUI instance.
- Si $v \in S$, même raisonnement.

Un algorithme plus rapide pour VC_{dec}.

```
bool A(G,k){// k >= 0
  if(E(G)=vide)// G n'a pas d'arete
    return true
  if(k=0)
    return false
  soit uv une arete de G
  return A(G-u,k-1) || A(G-v,k-1)
}
```

Branchement correct ?

I OUI instance $\Leftarrow \exists i$ tq I_i est une OUI instance ?

- Supposons par exemple que $(G - u, k - 1)$ OUI instance. Donc $G - u$ admet un VC S de taille au plus $k - 1$.
- Or, $S \cup \{u\}$ est bien un VC de G de taille au plus k, donc (G, k) est une OUI instance.

Un algorithme plus rapide pour VC_{dec}.

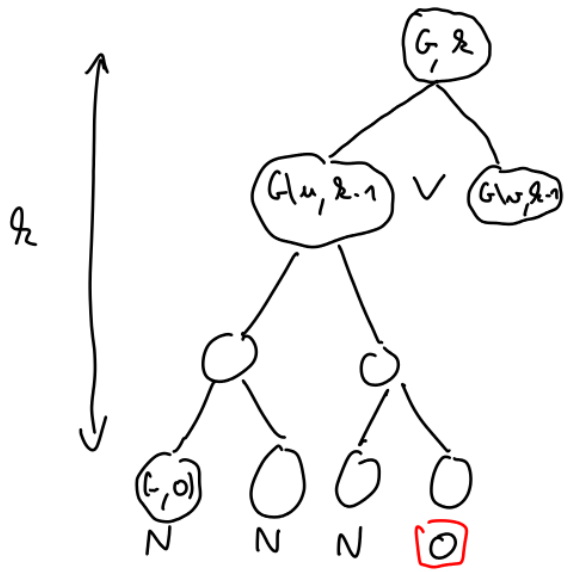
```
bool A(G,k){// k >= 0
  if(E(G)=vide)// G n'a pas d'arete
    return true
  if(k=0)
    return false
  soit uv une arete de G
  return A(G-u,k-1) || A(G-v,k-1)
}
```

Conclusion

L'algorithme A est correct.

Temps d'éxecution ?

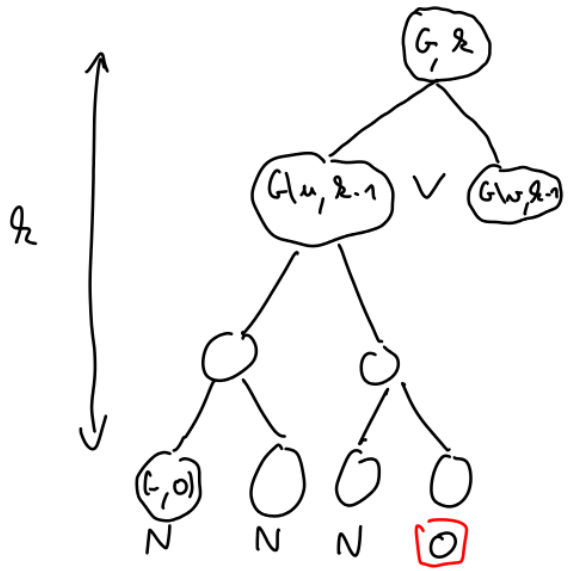
Un algorithme plus rapide pour VC_{dec}.



Un méthode pour calculer la complexité

- se faire une idée en se disant
  complexité = $\mathcal{O}$(nbnoeuds * t) avec t= complexité des calculs sur un noeud
- faire la preuve par récurrence

Un algorithme plus rapide pour VC_{dec}.



- nbnoeuds $= 1 + 2 + \cdots + 2^k \leq 2^{k+1}$
- temps $= \mathcal{O}(n^2)$ (analyse naïve) avec $n = |V(G)|$ car
  - il faut vérifier $E(G) = \emptyset$ et construire $G - u$ et $G - v$
  - on majore beaucoup ici car dans les appels rec, le graphe à moins de n sommets
- complexité $= \mathcal{O}(2^k n^2)$

Un algorithme plus rapide pour VC_{dec}.

- Soit $f(n,k)$ nb d'opérations faites dans le pire cas par $A(G,k)$ (avec G à n sommets).
- Soit $p(n)$ complexité des opérations faites sur un noeud (donc toutes les opérations, saufs celles dans appels rec), avec $p(n) = \mathcal{O}(n^2)$.

Prouvons par rec sur k que $f(n,k) \leq (2^{k+1}-1)p(n)$.

- $k = 0$, ok
- k quelconque,

$$\begin{aligned} f(n,k) &\leq p(n) + 2f(n-1,k-1) \\ &\leq p(n) + 2 \times (2^k - 1) \times p(n-1) \\ &\leq p(n) + 2 \times (2^k - 1) \times p(n) \\ &\leq p(n)(1 + 2^{k+1} - 2) \\ &= p(n)(2^{k+1} - 1) \end{aligned}$$

Un algorithme plus rapide pour VC_{dec}.

- Soit $f(n,k)$ nb d'opérations faites dans le pire cas par $A(G,k)$ (avec G à n sommets).
- Soit $p(n)$ complexité des opérations faites sur un noeud (donc toutes les opérations, saufs celles dans appels rec), avec $p(n) = \mathcal{O}(n^2)$.

On obtient donc :

Proposition

On a donc $f(n,k) \leq (2^{k+1}-1)p(n) = \mathcal{O}(2^k p(n)) = \mathcal{O}(2^k n^2)$.

Un algorithme plus rapide pour VC_{dec}.

- Soit $f(n,k)$ nb d'opérations faites dans le pire cas par $A(G,k)$ (avec G à n sommets).
- Soit $p(n)$ complexité des opérations faites sur un noeud (donc toutes les opérations, saufs celles dans appels rec), avec $p(n) = \mathcal{O}(n^2)$.

Pour les fois suivantes, on pourra utiliser le résultat suivant.

Proposition

Un algorithme de branchement qui

- fait au plus Δ appels récursifs (degré max Δ vers le bas dans l'arbre)
- sur une profondeur $\leq k$
- avec une complexité de $p(n)$ sur chaque noeud

A une complexité en $\mathcal{O}(\Delta^k p(n))$

Conclusion pour Vertex Cover

- On sait résoudre VC_{dec} en temps $\mathcal{O}(2^k n^2)$.
- VC_{dec}/k est donc FPT.

Outline

Algorithme XP

Rappel

Le problème IS_{dec}/k est

- entrée : un graphe G, un entier k
- paramètre : k
- question : décider si $opt(G) \geq k$ (cad si il existe un stable de taille k)

But

Résoudre IS_{dec} en temps $\mathcal{O}(|I|^{f(p(I))} poly(|I|))$ pour une certaine fonction f

Algorithme XP

Rappel

Le problème IS_{dec}/k est

- entrée : un graphe G, un entier k
- paramètre : k
- question : décider si $opt(G) \geq k$ (cad si il existe un stable de taille k)

But

Résoudre IS_{dec} en temps $\mathcal{O}(n^{f(k)}poly(n))$ pour une certaine fonction f

Algorithme XP

Rappel

Le problème IS_{dec}/k est

- entrée : un graphe G, un entier k
- paramètre : k
- question : décider si $opt(G) \geq k$ (cad si il existe un stable de taille k)

But

Résoudre IS_{dec} en temps $\mathcal{O}(n^k poly(n))$

Algorithme XP

Un algorithme pour IS_{dec}

- Pour tout ensemble S de k sommets, vérfier (en temps poly, par exemple $\mathcal{O}(n^2)$) que S est un stable (ne contient pas d'arêtes).
- Complexité $\mathcal{O}(n^k n^2)$

Et si on cherchait un algorithme plus rapide pour IS_{dec}/k :

But

Résoudre IS_{dec} en temps $\mathcal{O}(f(p(I))poly(|I|))$ pour une certaine fonction f

Algorithme XP

Un algorithme pour IS_{dec}

- Pour tout ensemble S de k sommets, vérfier (en temps poly, par exemple $\mathcal{O}(n^2)$) que S est un stable (ne contient pas d'arêtes).
- Complexité $\mathcal{O}(n^k n^2)$

Et si on cherchait un algorithme plus rapide pour IS_{dec}/k :

But

Résoudre IS_{dec} en temps $\mathcal{O}(f(k)poly(n))$ pour une certaine fonction f

Algorithme XP

Un algorithme pour IS_{dec}

- Pour tout ensemble S de k sommets, vérfier (en temps poly, par exemple $\mathcal{O}(n^2)$) que S est un stable (ne contient pas d'arêtes).
- Complexité $\mathcal{O}(n^k n^2)$

Et si on cherchait un algorithme plus rapide pour IS_{dec}/k :

But

Résoudre IS_{dec} en temps $\mathcal{O}(2^k poly(n))$

Algorithme XP

Un algorithme pour IS_{dec}

- Pour tout ensemble S de k sommets, vérfier (en temps poly, par exemple $\mathcal{O}(n^2)$) que S est un stable (ne contient pas d'arêtes).
- Complexité $\mathcal{O}(n^k n^2)$

Et si on cherchait un algorithme plus rapide pour IS_{dec}/k :

But

Résoudre IS_{dec} en temps $\mathcal{O}(2^k poly(n))$: personne n'a jamais trouvé

Algorithme XP

Un algorithme pour IS_{dec}

- Pour tout ensemble S de k sommets, vérfier (en temps poly, par exemple $\mathcal{O}(n^2)$) que S est un stable (ne contient pas d'arêtes).
- Complexité $\mathcal{O}(n^k n^2)$

Et si on cherchait un algorithme plus rapide pour IS_{dec}/k :

But

Résoudre IS_{dec} en temps $\mathcal{O}(f(k)poly(n))$ pour une certaine fonction f : personne n'a jamais trouvé (pour aucune fonction f)

Algorithme XP

Un algorithme pour IS_{dec}

- Pour tout ensemble S de k sommets, vérfier (en temps poly, par exemple $\mathcal{O}(n^2)$) que S est un stable (ne contient pas d'arêtes).
- Complexité $\mathcal{O}(n^k n^2)$

Et si on cherchait un algorithme plus rapide pour IS_{dec}/k :

But

IS_{dec}/k FPT: personne n'a jamais trouvé

Algorithme XP

Un algorithme pour IS_{dec}

- Pour tout ensemble S de k sommets, vérfier (en temps poly, par exemple $\mathcal{O}(n^2)$) que S est un stable (ne contient pas d'arêtes).
- Complexité $\mathcal{O}(n^k n^2)$

Et si on cherchait un algorithme plus rapide pour IS_{dec}/k :

IS_{dec}/k est dit W1-hard

Conclusion

La complexité paramétrée permet un regard plus fin :

- VC_{dec} et IS_{dec} sont tous les deux NP-complet.
- Cependant, VC_{dec}/k est FPT, alors que IS_{dec}/k est W1-hard.

Outline

MAX-SAT

Le problème MAX-SAT$_{dec}/k$ est

- entrée : un ensemble C, un entier k
- paramètre : $p(C, k) = k$
- question : décider si on peut satisfaire au moins k clauses

MAX-SAT$_{dec}/k$ est FPT, plus précisément on peut le résoudre en $\mathcal{O}(2^k poly(n))$.

Preuve :

- si il existe une variable n'apparaissant que positivement :
- soit k_1 le nombre de clauses satisfaites en mettant x_i à vraie, et I_1 l'instance obtenue en enlevant ces k_1 clauses
- retourner $A(I_1, k - k_1)$

MAX-SAT

Le problème MAX-SAT$_{dec}/k$ est

- entrée : un ensemble C, un entier k
- paramètre : $p(C, k) = k$
- question : décider si on peut satisfaire au moins k clauses

MAX-SAT$_{dec}/k$ est FPT, plus précisément on peut le résoudre en $\mathcal{O}(2^k poly(n))$.

Preuve :

- si il existe une variable n'apparaissant que négativement :
- même idée

MAX-SAT

Le problème MAX-SAT$_{dec}/k$ est

- entrée : un ensemble C, un entier k
- paramètre : $p(C, k) = k$
- question : décider si on peut satisfaire au moins k clauses

MAX-SAT$_{dec}/k$ est FPT, plus précisément on peut le résoudre en $\mathcal{O}(2^k poly(n))$.

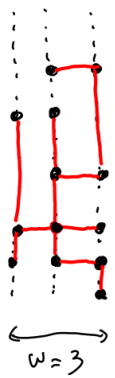
Preuve :

- sinon, prendre une variable x_i (apparaissant pos et neg)
- soit k_1 le nombre de clauses satisfaites en mettant x_i à vraie, et I_1 l'instance obtenue en enlevant ces k_1 clauses, et en enlevant x_i des clauses restantes
- soit k_2, I_2, .. pareil avec x_i à faux. On a $k_i > 0$.
- $A(I, k)$ va retourner $A(I_1, k - k_1)$ OU $A(I_2, k - k_2)$

Le paramètre n'est pas toujours la valeur qu'on cherche à atteindre.
On peut considérer des paramètres dits "structuraux" :

Le problème IS_{dec}/width est

- entrée : un graphe G planaire représentable sur une grille, un entier k
- paramètre : $p(G, k) = w$, la largeur de la grille
- question : décider si il existe un stable de taille au moins k



IS_{dec}/width est FPT, plus plus précisément on peut le résoudre en $\mathcal{O}(2^w poly(n))$.

Outline

Améliorer l'implémentation naïve

Revenons à VC et à l'algorithme précédent.

```
bool A(G,k){// k >= 0
  if(E(G)=vide)// G n'a pas d'arete
    return true
  if(k=0)
    return false
  soit uv une arete de G
  return A(G-u,k-1) || A(G-v,k-1)
}
```

- On garde l'idée de brancher sur "quel sommet d'une arête uv" on doit prendre.
- Idée : il y a parfois certains petits "morceaux d'instance" qu'on peut simplifier

Améliorer l'implémentation naïve

Idée

Ajouter des règles de réduction. Par exemple du type

- "si (...) alors je sais que je peux simplifier un peu l'instance (sans brancher, et en gardant une instance équivalente)"

Application pour VC

- Si il existe un sommet u de degré 1, et de voisin v, alors on a toujours intérêt à prendre v
- Autrement dit $(G, k) \Leftrightarrow (G - v, k - 1)$

On peut pousser plus loin cette idée de "règles de réduction" (avec la notion de "kernel").

Règle de réduction

Etre capable de déterminer si une règle de réduction est correcte!

Règle correcte : preuve

- Si il existe un sommet u de degré 1, et de voisin v, alors on a toujours intérêt à prendre v
- Autrement dit $(G, k) \Leftrightarrow (G - v, k - 1)$

Preuve $\Leftarrow$:

- Si $(G - v, k - 1)$ est une oui-instance, soit S' un VC de G avec $|S'| \leq k - 1$
- Soit $S = S \cup \{v\}$
- S est un VC de G de taille k, donc (G, k) est une oui-instance

Règle de réduction

Etre capable de déterminer si une règle de réduction est correcte!

Règle correcte : preuve

- Si il existe un sommet u de degré 1, et de voisin v, alors on a toujours intérêt à prendre v
- Autrement dit $(G, k) \Leftrightarrow (G - v, k - 1)$

Preuve $\Rightarrow$:

- Si (G, k) est une oui-instance, soit S un VC de G avec $|S| \leq k$
- Modifions S pour avoir une autre solution S' qui contient v :
- Si $v \in S$, alors $S' = S$
- Sinon, forcément $u \in S$, et alors $S' = (S \setminus u) \cup \{v\}$ est toujours un VC de G avec $|S'| \leq k$
- Maintenant $S' - v$ est un VC de $G - v$ de taille $k - 1$, donc $(G - v, k - 1)$ est une oui-instance

Règle de réduction

Etre capable de déterminer si une règle de réduction est correcte!

Règle incorrecte : contre exemple

- Si il existe un sommet u de degré 2, et de voisins v_1, v_2, alors on a toujours intérêt à prendre les v_i
- Autrement dit $(G, k) \Leftrightarrow (G - \{v_1, v_2\}, k - 2)$

L'implication $\Rightarrow$ est fausse.

- Soit G composé d'un chemin (v_1, v, v_2) et d'une arête séparée $\{u, u'\}$ et $k = 2$
- (G, k) est une oui-instance (prendre $S = \{v, u\}$)
- $(G - \{v_1, v_2\}, k - 2)$ n'est pas une oui-instance

Améliorer l'implémentation naïve

```
bool A(G,k)// k >= 0
  if(existe u de degre 1)//v est voisin de u
    return A(G-v,k-1);
  if(E(G)=vide)// G n'a pas d'arete
     return true
  if(k=0)
    return false
  soit uv une arete de G
  return A(G-u,k-1) || A(G-v,k-1)
```

Algorithme FPT pour problème d'optimisation

Considérons le cas d'un problème d'optimisation Π. Ce qui nous intéresse étant donné I, c'est calculer $opt(I)$. Si l'on a A, un algorithme FPTpour la paramétrisation standard, il suffit d'exécuter $A(I, k)$ pour k partant de 0.

Pour un problème de minimisation :

- $A(I, 0)$ $(opt(I) \leq 0?)$: false
- $A(I, 1)$ $(opt(I) \leq 1?)$: ..
- $A(I, k^* - 1)$ $(opt(I) \leq k^* - 1?)$: false
- $A(I, k^*)$ $(opt(I) \leq k^*?)$: true // on s'arrête à $k^* =$ plus petit k tq $A(I, k)$ répond true

- On a donc une solution optimale car $k^* = opt(I)$.
- Complexité $f(I) \leq k^* f(I, k^*)$.

Ex pour VC, on obtient $f(I) = \mathcal{O}(k^* 2^{k^*} n^2) = \mathcal{O}(2^{k^*} n^3)$.