Algorithmes exponentiels pour une généralisation de la domination

Mathieu Liedloff[3]
en collaboration avec
Fedor V. Fomin[1] Petr A. Golovach[1]
Jan Kratochvíl[2] Dieter Kratsch[3]

[1]Université de Bergen
Bergen, Norvège

[2]Université Charles
Prague, République Tchèque

[3]Université Paul Verlaine
Metz, France

Séminaire *Visualisation et Algorithmes de Graphes*, LIRMM, juin 2008

Plan de l'exposé

Historique

- En 1971, Cook démontre la NP-complétude de SAT [Coo71] ;
- En 1979, Garey et Johnson [GJ79] recensent plus de 300 problèmes NP-complets.
- Juin 2008 : on ne compte plus leur nombre ...
  ... ils sont présents dans de nombreux domaines.

Conséquence (actuelle) la plus importante :

Aucun problème NP-complet ne peut être résolu en temps polynomial, sauf si P=NP.

Question ouverte depuis 30 ans !

Historique

- En 1971, Cook démontre la NP-complétude de SAT [Coo71] ;
- En 1979, Garey et Johnson [GJ79] recensent plus de 300 problèmes NP-complets.
- Juin 2008 : on ne compte plus leur nombre ...
  ... ils sont présents dans de nombreux domaines.

Conséquence (actuelle) la plus importante :

Aucun problème NP-complet ne peut être résolu en temps polynomial, sauf si P=NP.

Question ouverte depuis 30 ans !

Attaques possibles

Différentes attaques pour résoudre un tel problème :

- imposer des restrictions sur l'entrée ;
- chercher un algorithme d'approximation ;
- chercher un algorithme randomisé ;
- chercher un algorithme à paramètre fixé ;
- utiliser des heuristiques ;
- construire un algorithme demandant un temps exponentiel.

Attaques possibles

Différentes attaques pour résoudre un tel problème :

- imposer des restrictions sur l'entrée ;
- chercher un algorithme d'approximation ;
- chercher un algorithme randomisé ;
- chercher un algorithme à paramètre fixé ;
- utiliser des heuristiques ;
- construire un algorithme demandant un temps exponentiel.

Attaques possibles

Différentes attaques pour résoudre un tel problème :

- imposer des restrictions sur l'entrée ;
- chercher un algorithme d'approximation ;
- chercher un algorithme randomisé ;
- chercher un algorithme à paramètre fixé ;
- utiliser des heuristiques ;
- construire un algorithme demandant un temps exponentiel.

Attaques possibles

Différentes attaques pour résoudre un tel problème :

- imposer des restrictions sur l'entrée ;
- chercher un algorithme d'approximation ;
- chercher un algorithme randomisé ;
- chercher un algorithme à paramètre fixé ;
- utiliser des heuristiques ;
- construire un algorithme demandant un temps exponentiel.

Attaques possibles

Différentes attaques pour résoudre un tel problème :

- imposer des restrictions sur l'entrée ;
- chercher un algorithme d'approximation ;
- chercher un algorithme randomisé ;
- chercher un algorithme à paramètre fixé ;
- utiliser des heuristiques ;
- construire un algorithme demandant un temps exponentiel.

Attaques possibles

Différentes attaques pour résoudre un tel problème :

- imposer des restrictions sur l'entrée ;
- chercher un algorithme d'approximation ;
- chercher un algorithme randomisé ;
- chercher un algorithme à paramètre fixé ;
- utiliser des heuristiques ;
- construire un algorithme demandant un temps exponentiel.

Attaques possibles

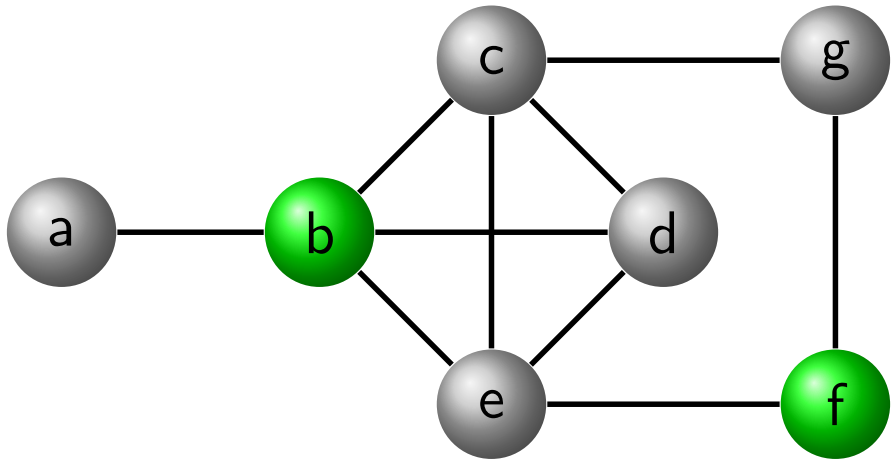
Objectif de cet exposé :

S'attaquer à des problèmes de type domination dans un graphe au moyen d'algorithmes exponentiels.

Domination

Entrée : un graphe $G = (V, E)$.
Question : trouver un plus petit ensemble dominant de G.

Un ensemble D est dominant si tout sommet de $V \setminus D$ a au moins un voisin dans D.



Motivations

L'utilisation des différentes attaques n'est pas toujours possible.

Pour le problème DOMINATION :

- pas d'algorithme calculant une solution approchée avec un rapport constant, sauf si P=NP ; [RS97, Fei98]
- pas d'algorithme à paramètre fixé, sauf si W[2]=FPT. [DF99]

Si on cherche une solution exacte, un algorithme exponentiel est utile.

Motivations

L'utilisation des différentes attaques n'est pas toujours possible.

Pour le problème DOMINATION :

- pas d'algorithme calculant une solution approchée avec un rapport constant, sauf si P=NP ; [RS97, Fei98]
- pas d'algorithme à paramètre fixé, sauf si W[2]=FPT. [DF99]

Si on cherche une solution exacte, un algorithme exponentiel est utile.

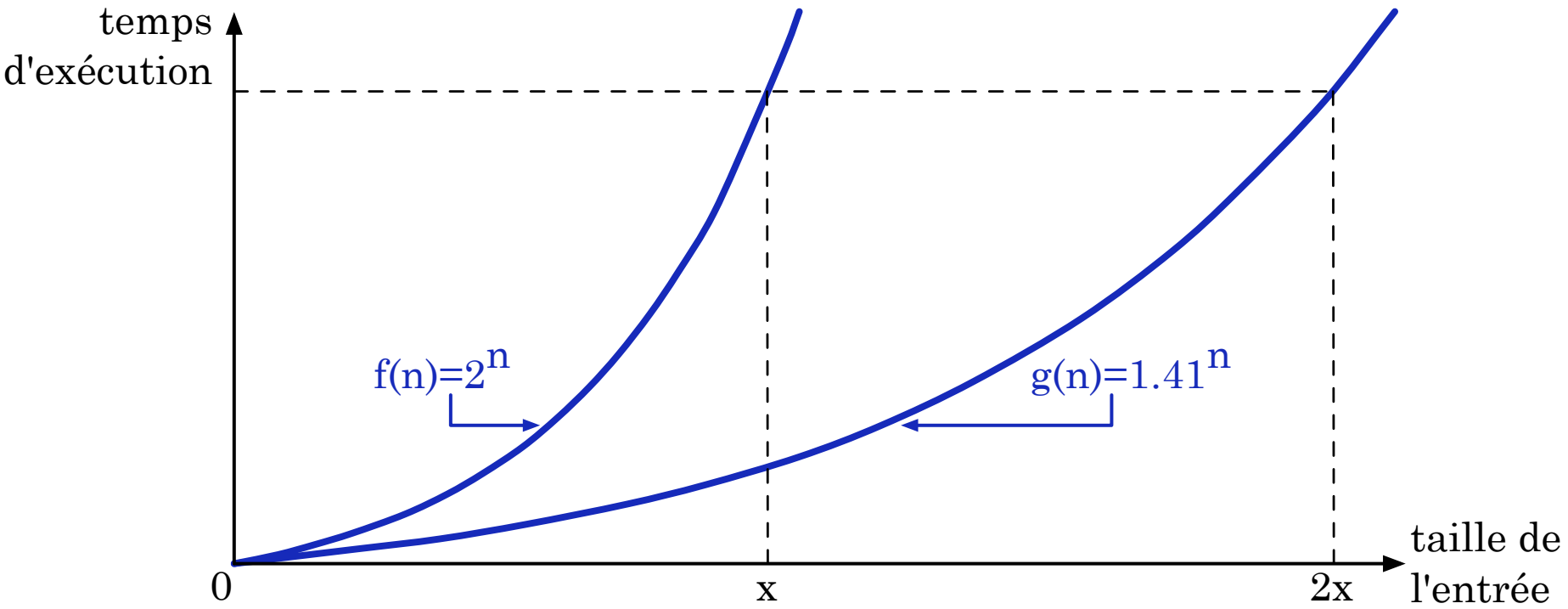
Algorithmes exponentiels

Si une entrée de taille T est traitée en une unité de temps

- par un algorithme polynomial en n^c
  $\Rightarrow$ une machine $100\times$ plus rapide traite $\sqrt[c]{100} \cdot T$
  $\longrightarrow$ gain : facteur multiplicatif

- par un algorithme exponentiel en c^n
  $\Rightarrow$ une machine $100\times$ plus rapide traite $\log_c 100 + T$
  $\longrightarrow$ gain : facteur additif

Algorithmes exponentiels



$2^{n/2} \approx 1.41^n$ $2^{n/3} \approx 1.26^n$ $2^{n/4} \approx 1.19^n$

Le problème Domination

Le problème Domination est

- NP-complet ;
- W[2]-complet ;
- non approximable avec un facteur constant (seulement $\log n$-approximable, sauf si P=NP).

Construire des algorithmes exponentiels rapides pour le résoudre ?

Domination peut être résolu en temps

$$O(\text{poly}(n) \cdot 2^n) = O^*(2^n).$$

Le problème Domination

Le problème Domination est

- NP-complet ;
- W[2]-complet ;
- non approximable avec un facteur constant (seulement $\log n$-approximable, sauf si P=NP).

Construire des algorithmes exponentiels rapides pour le résoudre ?

Domination peut être résolu en temps

$$O(\text{poly}(n) \cdot 2^n) = O^*(2^n).$$

Le problème DOMINATION

Le problème DOMINATION est

- NP-complet ;
- W[2]-complet ;
- non approximable avec un facteur constant (seulement log n-approximable, sauf si P=NP).

Construire des algorithmes exponentiels rapides pour le résoudre ?

DOMINATION peut être résolu en temps

$$O(\text{poly}(n) \cdot 2^n) = O^*(2^n).$$

Le problème Domination

Le problème Domination est

- NP-complet ;
- W[2]-complet ;
- non approximable avec un facteur constant (seulement $\log n$-approximable, sauf si P=NP).

Il est possible de faire mieux !

Résultats connus pour DOMINATION

- 2004 : $O(1.9386^n)$ [Fomin, Kratsch, Woeginger]
- 2004 : $O(1.8899^n)$ [Randerath, Schiermeyer]
- 2004 : $O(1.8025^n)$ [Grandoni]
- 2005 : [Fomin, Grandoni, Kratsch]
  - $O(1.5263^n)$ (espace polynomial)
  - $O(1.5137^n)$ (espace exponentiel)
- 2007 : $O(1.7088^n)$ [L.]
- 2008 : [van Rooij, Bodlaender]
  - $O(1.5134^n)$ (espace polynomial)
  - $O(1.5063^n)$ (espace exponentiel)

Résultats connus pour DOMINATION

- 2004 : $O(1.9386^n)$ [Fomin, Kratsch, Woeginger]
- 2004 : $O(1.8899^n)$ [Randerath, Schiermeyer]
- 2004 : $O(1.8025^n)$ [Grandoni]
- 2005 : [Fomin, Grandoni, Kratsch]
  - $O(1.5263^n)$ (espace polynomial)
  - $O(1.5137^n)$ (espace exponentiel)
- 2007 : $O(1.7088^n)$ [L.]
- 2008 : [van Rooij, Bodlaender]
  - $O(1.5134^n)$ (espace polynomial)
  - $O(1.5063^n)$ (espace exponentiel)

Résultats connus pour DOMINATION

- 2004 : $O(1.9386^n)$ [Fomin, Kratsch, Woeginger]
- 2004 : $O(1.8899^n)$ [Randerath, Schiermeyer]
- 2004 : $O(1.8025^n)$ [Grandoni]
- 2005 : [Fomin, Grandoni, Kratsch]
  - $O(1.5263^n)$ (espace polynomial)
  - $O(1.5137^n)$ (espace exponentiel)
- 2007 : $O(1.7088^n)$ [L.]
- 2008 : [van Rooij, Bodlaender]
  - $O(1.5134^n)$ (espace polynomial)
  - $O(1.5063^n)$ (espace exponentiel)

Résultats connus pour Domination

- 2004 : $O(1.9386^n)$ [Fomin, Kratsch, Woeginger]
- 2004 : $O(1.8899^n)$ [Randerath, Schiermeyer]
- 2004 : $O(1.8025^n)$ [Grandoni]
- 2005 : [Fomin, Grandoni, Kratsch]
  - $O(1.5263^n)$ (espace polynomial)
  - $O(1.5137^n)$ (espace exponentiel)
- 2007 : $O(1.7088^n)$ [L.]
- 2008 : [van Rooij, Bodlaender]
  - $O(1.5134^n)$ (espace polynomial)
  - $O(1.5063^n)$ (espace exponentiel)

Résultats connus pour DOMINATION

- 2004 : $O(1.9386^n)$ [Fomin, Kratsch, Woeginger]
- 2004 : $O(1.8899^n)$ [Randerath, Schiermeyer]
- 2004 : $O(1.8025^n)$ [Grandoni]
- 2005 : [Fomin, Grandoni, Kratsch]
  - $O(1.5263^n)$ (espace polynomial)
  - $O(1.5137^n)$ (espace exponentiel)
- 2007 : $O(1.7088^n)$ [L.]
- 2008 : [van Rooij, Bodlaender]
  - $O(1.5134^n)$ (espace polynomial)
  - $O(1.5063^n)$ (espace exponentiel)

Résultats connus pour DOMINATION

- 2004 : $O(1.9386^n)$ [Fomin, Kratsch, Woeginger]
- 2004 : $O(1.8899^n)$ [Randerath, Schiermeyer]
- 2004 : $O(1.8025^n)$ [Grandoni]
- 2005 : [Fomin, Grandoni, Kratsch]
  - $O(1.5263^n)$ (espace polynomial)
  - $O(1.5137^n)$ (espace exponentiel)
- 2007 : $O(1.7088^n)$ [L.]
- 2008 : [van Rooij, Bodlaender]
  - $O(1.5134^n)$ (espace polynomial)
  - $O(1.5063^n)$ (espace exponentiel)

Résultats connus pour DOMINATION

- 2004 : $O(1.9386^n)$ [Fomin, Kratsch, Woeginger]
- 2004 : $O(1.8899^n)$ [Randerath, Schiermeyer]
- 2004 : $O(1.8025^n)$ [Grandoni]
- 2005 : [Fomin, Grandoni, Kratsch]
  - $O(1.5263^n)$ (espace polynomial)
  - $O(1.5137^n)$ (espace exponentiel)
- 2007 : $O(1.7088^n)$ [L.]
- 2008 : [van Rooij, Bodlaender]
  - $O(1.5134^n)$ (espace polynomial)
  - $O(1.5063^n)$ (espace exponentiel)

Domination sur quelques classes de graphes

Domination reste NP-complet, même restreint aux graphes :

- planaires
- max degré 3
- c-denses
- cordaux
- 4-cordaux
- faiblement cordaux
- cercles
- bipartis
- splits

Domination sur quelques classes de graphes

Domination reste NP-complet, même restreint aux graphes :

planaires	$\longrightarrow$	$O(15.8895^{\sqrt{n}})$	[Dorn07]
max degré 3	$\longrightarrow$	$O(1.2010^n)$	[Fomin, Høie]
c-denses	$\longrightarrow$	$O(1.2303^{n(1+\sqrt{1-2c})})$	[GKLT]
cordaux	$\longrightarrow$	$O(1.4173^n)$	[GKLT]
4-cordaux	$\longrightarrow$	$O(1.4913^n)$	[GKLT]
faiblement cordaux	$\longrightarrow$	$O(1.4842^n)$	[GKLT]
cercles	$\longrightarrow$	$O(1.4956^n)$	[GKLT]
bipartis	$\longrightarrow$	$O(1.4143^n)$	[L.]
splits	$\longrightarrow$	$O(1.2303^n)$	[folklore]

Rappel [van Rooij, Bodlaender] : graphe quelconque en $O(1.5063^n)$

Variantes de la domination

Diverses variantes du problème DOMINATION ont été étudiées :

- Edge Dominating Set $\longrightarrow$ $O(1.3226^n)$ [RB08]
- Connected Dominating Set $\longrightarrow$ $O(1.9407^n)$ [FGK06]
- Independent Dominating Set $\longrightarrow$ $O(1.3575^n)$ [GL06]
- Min / Ex Dominating Clique $\longrightarrow$ $O(1.3387^n)$ [KL07]
- Partial Domination $\longrightarrow$ $O(1.6183^n)$ [L07]
- Roman Domination $\longrightarrow$ $O(1.6183^n)$ [L07]
- Broadcast Domination $\longrightarrow$ $O^*(2^n)$ [L07]

Conception et Analyse d'algorithmes exponentiels

L'obtention de ces temps d'exécution semble **magique !**

La plupart des algorithmes compétitifs pour DOMINATION utilisent

→ comme technique de conception :

Brancher & Réduire

→ comme technique d'analyse :

Mesurer pour Conquérir

Regardons deux algorithmes pour le problème ENSEMBLE STABLE.

La généralisation permet aussi de modéliser ce problème.

Conception et Analyse d'algorithmes exponentiels

L'obtention de ces temps d'exécution semble **magique !**

La plupart des algorithmes compétitifs pour DOMINATION utilisent

→ comme technique de conception :

Brancher & Réduire

→ comme technique d'analyse :

Mesurer pour Conquérir

Regardons deux algorithmes pour le problème ENSEMBLE STABLE.

La généralisation permet aussi de modéliser ce problème.

Conception et Analyse d'algorithmes exponentiels

L'obtention de ces temps d'exécution semble **magique !**

La plupart des algorithmes compétitifs pour DOMINATION utilisent

→ comme technique de conception :

Brancher & Réduire

→ comme technique d'analyse :

Mesurer pour Conquérir

Regardons deux algorithmes pour le problème ENSEMBLE STABLE.

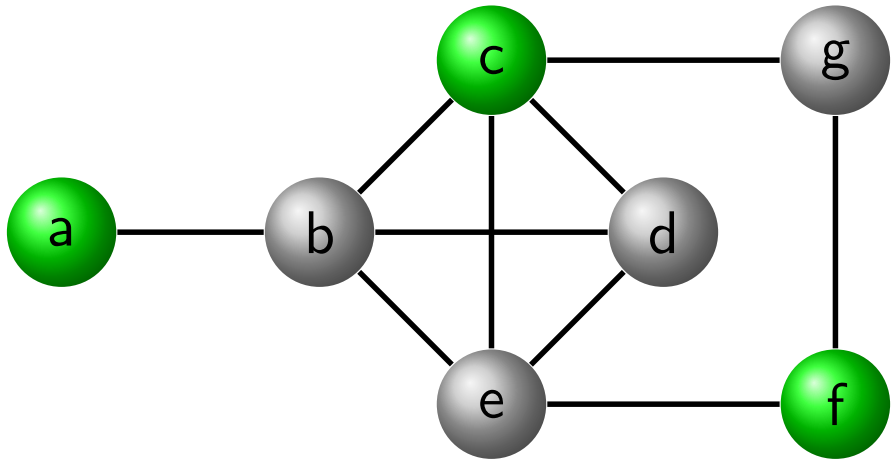
La généralisation permet aussi de modéliser ce problème.

Le problème Ensemble Stable

Ensemble Stable

Entrée : un graphe $G = (V, E)$.
Question : trouver un plus grand ensemble stable de G.

Un ensemble S est stable si les sommets de S sont deux à deux non adjacents.



Ensemble Stable - Algorithme I

Un premier algorithme :

énumérer tous les ensembles stables maximaux (par l'inclusion)

- Prendre un sommet v de degré minimum dans le graphe
  - $\longrightarrow$ soit v appartient à l'ensemble
  - $\longrightarrow$ soit (par maximalité) un voisin de v appartient à l'ensemble

Algorithme `EnumEnsStable`$(G,\ S)$
Entrées: Un graphe G et un ensemble stable S.
Sortie: L'union de S avec tous les ens. stables de G.

si G est vide alors
  retourner l'ensemble stable S

Choisir un sommet v de degré minimum

`EnumEnsStable`$(G - N[v],\ S \cup \{v\})$

pour tous les voisins w de v faire
  `EnumEnsStable`$(G - N[w],\ S \cup \{w\})$

Ensemble Stable - Algorithme I (analyse)

Un premier algorithme :

énumérer tous les ensembles stables maximaux (par l'inclusion)

- Prendre un sommet v de degré minimum dans le graphe
  - $\rightarrow$ soit v appartient à l'ensemble
  - $\rightarrow$ soit (par maximalité) un voisin de v appartient à l'ensemble

Analyse : Notons par $T(n)$ le temps d'exécution de l'algorithme sur un graphe à n sommet. Soit $d(x)$ le degré d'un sommet x.

$$T(n) \leq T(n-1-d(v)) + \sum_{w \in N(v)} T(n-1-d(w))$$

Comme $d(w) \geq d(v), \forall w \in N(v)$:

$$T(n) \leq T(n-1-d(v)) + d(v)T(n-1-d(v))$$

$\longrightarrow$ La solution de cette récurrence est maximum pour $d(v) = 2$.
On obtient alors $T(n) = O^*(3^{n/3})$.

Ensemble Stable - Algorithme I (conséquence)

Un premier algorithme :

énumérer tous les ensembles stables maximaux (par l'inclusion)

- Prendre un sommet v de degré minimum dans le graphe
  - $\rightarrow$ soit v appartient à l'ensemble
  - $\rightarrow$ soit (par maximalité) un voisin de v appartient à l'ensemble

Conséquence combinatoire :

Puisque l'algorithme énumère tous les ensembles stables maximaux d'un graphe, on en déduit que leur nombre est au plus $O^*(3^{n/3})$.

De plus, il existe vraiment des graphes ayant $3^{n/3}$ ensembles stables maximaux.

Autrement dit, le temps d'exécution de cet algorithme est optimal (à un facteur polynomial près).

Ensemble Stable - Algorithme II

Trouver un ensemble stable de taille maximum
sans énumérer tous les ensembles stables ?

Ensemble Stable - Algorithme II

Trouver un ensemble stable de taille maximum
sans énumérer tous les ensembles stables ?

Oui !

Ensemble Stable - Algorithme II

Un algorithme plus rapide pour calculer un ens. stable maximum :

- Prendre un sommet v de degré maximum dans le graphe

  $\longrightarrow$ soit v appartient à la solution
  $\longrightarrow$ soit v n'y appartient pas

Rappel : énumération en $O^*(3^{n/3}) = O(1.4423^n)$

Ensemble Stable - Algorithme II

Un algorithme plus rapide pour calculer un ens. stable maximum :

- Prendre un sommet v de degré maximum dans le graphe

  $\rightarrow$ soit v appartient à la solution
  $\rightarrow$ soit v n'y appartient pas

Analyse :

$$T(n) \leq T(n-1-d(v)) + T(n-1)$$

$\longrightarrow$ La solution de cette récurrence est maximum pour $d(v) = 0$.
On obtient alors $T(n) = O^*(2^n)$.

Rappel : énumération en $O^*(3^{n/3}) = O(1.4423^n)$

Ensemble Stable - Algorithme II

Un algorithme plus rapide pour calculer un ens. stable maximum :

- S'il existe un sommet v t.q. $d(v) = 0$ ou 1, ajouter v à la solution et supprimer $N[v]$.
- Prendre un sommet v de degré maximum dans le graphe

$\longrightarrow$ soit v appartient à la solution
$\longrightarrow$ soit v n'y appartient pas

Analyse :

$$T(n) \leq T(n-1-d(v)) + T(n-1)$$

Rappel : énumération en $O^*(3^{n/3}) = O(1.4423^n)$

Ensemble Stable - Algorithme II

Un algorithme plus rapide pour calculer un ens. stable maximum :

- S'il existe un sommet v t.q. $d(v) = 0$ ou 1, ajouter v à la solution et supprimer $N[v]$.
- Prendre un sommet v de degré maximum dans le graphe

$\rightarrow$ soit v appartient à la solution
$\rightarrow$ soit v n'y appartient pas

Analyse :

$$T(n) \leq T(n-1-d(v)) + T(n-1)$$

$\longrightarrow$ La solution de cette récurrence est maximum pour $d(v) = 2$.
On obtient alors $T(n) = O^*(1.4656^n)$.

Rappel : énumération en $O^*(3^{n/3}) = O(1.4423^n)$

Ensemble Stable - Algorithme II

Un algorithme plus rapide pour calculer un ens. stable maximum :

- S'il existe un sommet v t.q. $d(v) = 0$ ou 1, ajouter v à la solution et supprimer $N[v]$.
- Prendre un sommet v de degré maximum dans le graphe
  - Si $d(v) \geq 3$
    - $\rightarrow$ soit v appartient à la solution
    - $\rightarrow$ soit v n'y appartient pas
  - Sinon, on résout le problème en temps polynomial

Analyse :

$$T(n) \leq T(n-1-d(v)) + T(n-1)$$

Rappel : énumération en $O^*(3^{n/3}) = O(1.4423^n)$

Ensemble Stable - Algorithme II

Un algorithme plus rapide pour calculer un ens. stable maximum :

- S'il existe un sommet v t.q. $d(v) = 0$ ou 1, ajouter v à la solution et supprimer $N[v]$.
- Prendre un sommet v de degré maximum dans le graphe
  - Si $d(v) \geq 3$
    - $\rightarrow$ soit v appartient à la solution
    - $\rightarrow$ soit v n'y appartient pas
  - Sinon, on résout le problème en temps polynomial

Analyse :

$$T(n) \leq T(n-1-d(v)) + T(n-1)$$

$\longrightarrow$ La solution de cette récurrence est maximum pour $d(v) = 3$.
On obtient alors $T(n) = O^*(1.3803^n)$.

Rappel : énumération en $O^*(3^{n/3}) = O(1.4423^n)$

Ensemble Stable - Algorithme II

Un algorithme plus rapide pour calculer un ens. stable maximum :

- S'il existe un sommet v t.q. $d(v) = 0$ ou 1, ajouter v à la solution et supprimer $N[v]$.
- Prendre un sommet v de degré maximum dans le graphe
  - Si $d(v) \geq 3$
    - $\rightarrow$ soit v appartient à la solution
    - $\rightarrow$ soit v n'y appartient pas
  - Sinon, on résout le problème en temps polynomial

Cette analyse est-elle *optimale* ou peut-on améliorer le temps d'exécution ?

On obtient alors $T(n) = O^*(1.3803^n)$.

Rappel : énumération en $O^*(3^{n/3}) = O(1.4423^n)$

Ensemble Stable - Algorithme II (analyse)

- S'il existe un sommet v t.q. $d(v) = 0$ ou 1, ajouter v à la solution et supprimer $N[v]$.
- Prendre un sommet v de degré maximum dans le graphe
  - Si $d(v) \geq 3$
    - $\longrightarrow$ soit v appartient à la solution
    - $\longrightarrow$ soit v n'y appartient pas
  - Sinon, on résout le problème en temps polynomial

Soit N_i $(i \in \{2, 3, 4\})$ le nombre de sommets de degré i.
Soit $w_i \in [0, 1]$ $(i \in \{2, 3, 4\})$, des réels.

Soit la mesure $\mu(G) = w_2 N_2 + w_3 N_3 + w_{\geq 4} N_{\geq 4} \quad \leq n$

Refaisons l'analyse en notant
$T(\mu)$ le temps d'exécution de l'algo sur un graphe de taille $\mu(G)$.

Ensemble Stable - Algorithme II (analyse)

- S'il existe un sommet v t.q. $d(v) = 0$ ou 1, ajouter v à la solution et supprimer $N[v]$.
- Prendre un sommet v de degré maximum dans le graphe
  - Si $d(v) \geq 3$
    - $\rightarrow$ soit v appartient à la solution
    - $\rightarrow$ soit v n'y appartient pas
  - Sinon, on résout le problème en temps polynomial

$$\mu(G) = w_2 N_2 + w_3 N_3 + w_{\geq 4} N_{\geq 4} \quad \leq n$$

Soit v le sommet choisi par l'algorithme.
Notons d_2, d_3, d_4, $d_{\geq 5}$ son nombre de voisins de degré 2, 3, 4 et ≥ 5.

- Si $d(v) = 3$:

$$\begin{aligned} T(\mu) \quad \leq \quad & T\big(\mu - w_3 - d_2 w_2 - d_3 w_3\big) + \\ & T\big(\mu - w_3 - d_2 w_2 - d_3 (w_3 - w_2)\big) \end{aligned}$$

Ensemble Stable - Algorithme II (analyse)

- S'il existe un sommet v t.q. $d(v) = 0$ ou 1, ajouter v à la solution et supprimer $N[v]$.
- Prendre un sommet v de degré maximum dans le graphe
  - Si $d(v) \geq 3$
    - $\rightarrow$ soit v appartient à la solution
    - $\rightarrow$ soit v n'y appartient pas
  - Sinon, on résout le problème en temps polynomial

$$\mu(G) = w_2 N_2 + w_3 N_3 + w_{\geq 4} N_{\geq 4} \quad \leq n$$

Soit v le sommet choisi par l'algorithme.
Notons d_2, d_3, d_4, $d_{\geq 5}$ son nombre de voisins de degré 2, 3, 4 et ≥ 5.

- Si $d(v) = 4$:

$$\begin{aligned} T(\mu) \quad \leq \quad & T\left(\mu - w_{\geq 4} - d_2 w_2 - d_3 w_3 - d_4 w_{\geq 4}\right) \quad + \\ & T\left(\mu - w_{\geq 4} - d_2 w_2 - d_3 (w_3 - w_2) - d_4 (w_{\geq 4} - w_3)\right) \end{aligned}$$

Ensemble Stable - Algorithme II (analyse)

- S'il existe un sommet v t.q. $d(v) = 0$ ou 1, ajouter v à la solution et supprimer $N[v]$.
- Prendre un sommet v de degré maximum dans le graphe
  - Si $d(v) \geq 3$
    - $\rightarrow$ soit v appartient à la solution
    - $\rightarrow$ soit v n'y appartient pas
  - Sinon, on résout le problème en temps polynomial

$$\mu(G) = w_2 N_2 + w_3 N_3 + w_{\geq 4} N_{\geq 4} \quad \leq n$$

Soit v le sommet choisi par l'algorithme.
Notons d_2, d_3, d_4, $d_{\geq 5}$ son nombre de voisins de degré 2, 3, 4 et ≥ 5.

- Si $d(v) \geq 5$:

$$\begin{aligned} T(\mu) \quad \leq \quad & T\big(\mu - w_{\geq 4} - d_2 w_2 - d_3 w_3 - (d_4 + d_{\geq 5}) w_{\geq 4}\big) \quad + \\ & T\big(\mu - w_{\geq 4} - d_2 w_2 - d_3 (w_3 - w_2) - d_4 (w_{\geq 4} - w_3)\big) \end{aligned}$$

Ensemble Stable - Algorithme II (analyse)

- S'il existe un sommet v t.q. $d(v) = 0$ ou 1, ajouter v à la solution et supprimer $N[v]$.
- Prendre un sommet v de degré maximum dans le graphe
  - Si $d(v) \geq 3$
    - $\longrightarrow$ soit v appartient à la solution
    - $\longrightarrow$ soit v n'y appartient pas
  - Sinon, on résout le problème en temps polynomial

$$\mu(G) = w_2 N_2 + w_3 N_3 + w_{\geq 4} N_{\geq 4} \quad \leq n$$

$\longrightarrow$ calculer ces récurrences pour toutes les valeurs $0 \leq d_2,\ d_3,\ d_4,\ d_{\geq 5} \leq 5$.

$\longrightarrow$ chercher les valeurs des $w_i \in [0, 1]$ qui minimisent la plus grande solution

Ensemble Stable - Algorithme II (analyse)

- S'il existe un sommet v t.q. $d(v) = 0$ ou 1, ajouter v à la solution et supprimer $N[v]$.
- Prendre un sommet v de degré maximum dans le graphe
  - Si $d(v) \geq 3$
    - $\rightarrow$ soit v appartient à la solution
    - $\rightarrow$ soit v n'y appartient pas
  - Sinon, on résout le problème en temps polynomial

$$\mu(G) = w_2 N_2 + w_3 N_3 + w_{\geq 4} N_{\geq 4} \quad \leq n$$

Théorème

L'algorithme détermine un ensemble stable maximum en temps $O(1.2905^n)$.
($w2 = 0.5967$, $w_3 = 0.9287$ et $w_{\geq 4} = 1$)

Rappel : analyse précédente $O(1.3803^n)$

Analyse du temps d'exécution

- Pour établir cette borne supérieure, nous avons utilisé une mesure non standard.
- Une autre mesure pourrait donner une « meilleure borne ».

La borne supérieure est-elle éloignée de la meilleure borne qu'on pourrait obtenir ?

$\longrightarrow$ Une borne inférieure sur le temps d'exécution de l'algorithme donne une estimation sur la précision de l'analyse.

Analyse du temps d'exécution

- Pour établir cette borne supérieure, nous avons utilisé une mesure non standard.
- Une autre mesure pourrait donner une « meilleure borne ».

La borne supérieure est-elle éloignée de la meilleure borne qu'on pourrait obtenir ?

→ Une borne inférieure sur le temps d'exécution de l'algorithme donne une estimation sur la précision de l'analyse.

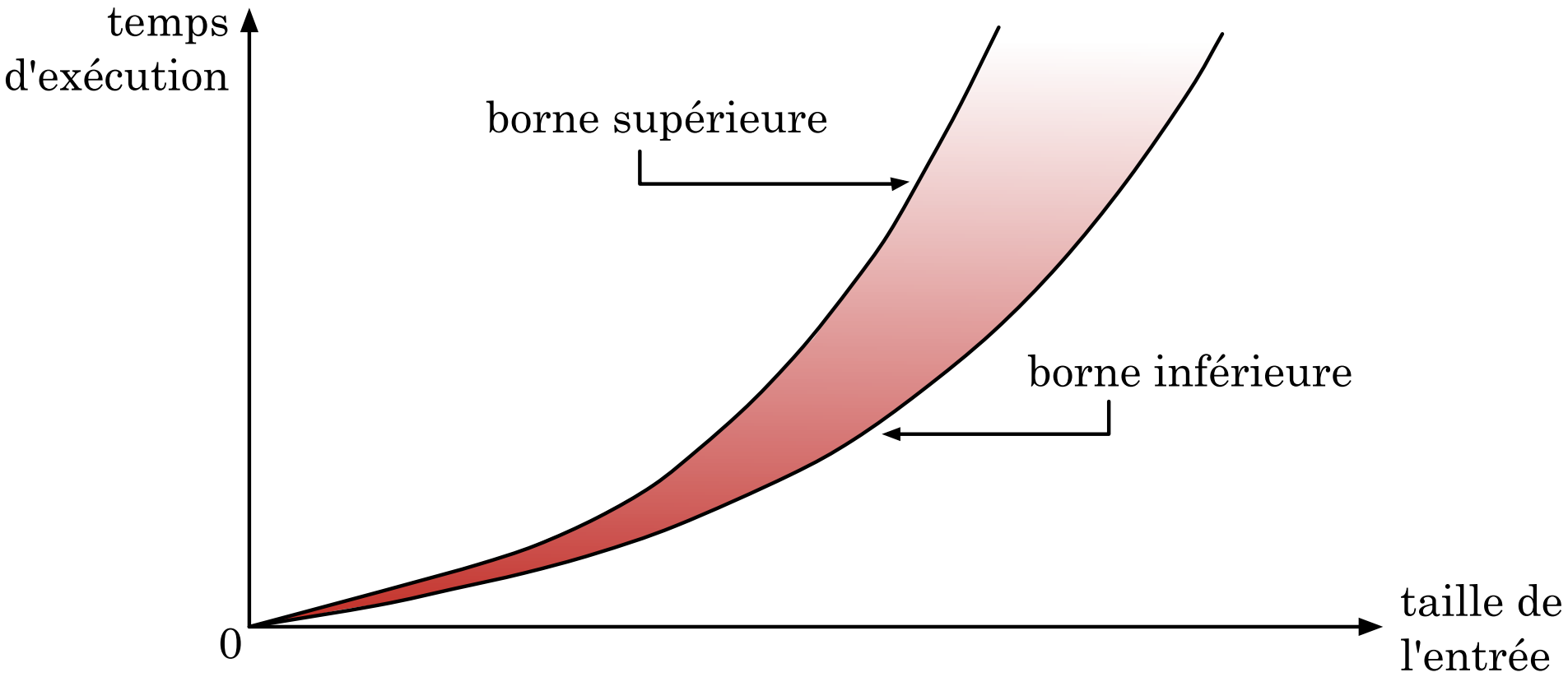
Analyse du temps d'exécution

- Pour établir cette borne supérieure, nous avons utilisé une mesure non standard.
- Une autre mesure pourrait donner une « meilleure borne ».

La borne supérieure est-elle éloignée de la meilleure borne qu'on pourrait obtenir ?

→ Une borne inférieure sur le temps d'exécution de l'algorithme donne une estimation sur la précision de l'analyse.

Borne inférieure sur le temps d'exécution de l'algorithme

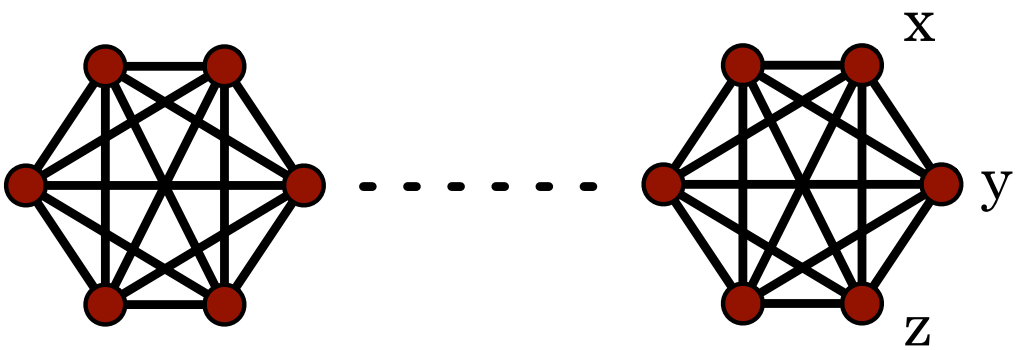


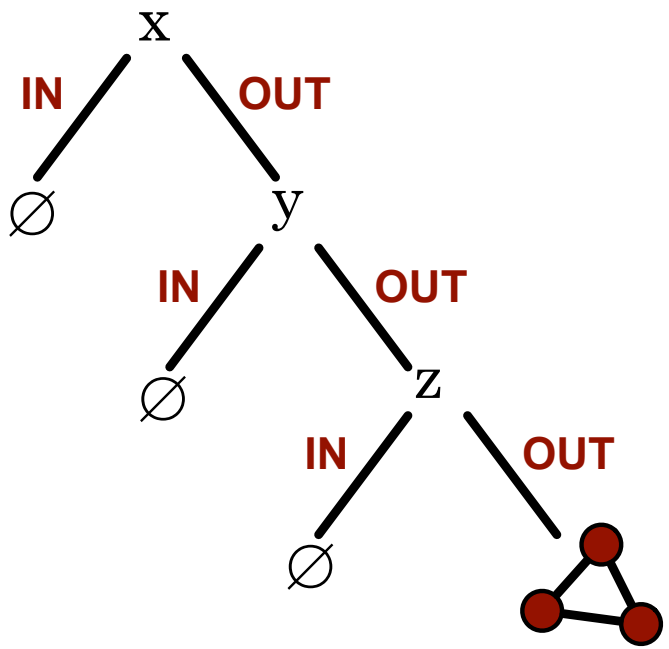
Borne inférieure sur le temps d'exécution de l'algorithme

Théorème (borne inférieure) :

L'algorithme résout le problème Ensemble Stable maximum en temps $\Omega(4^{n/6}) = \Omega(1.2599^n)$.

$\longrightarrow$ Collection de K_6.





Résultats connus pour Ensemble Stable

- 1977 : $O(1.2600^n)$ [Tarjan Trojanowski 77]
- 1986 : $O(1.2346^n)$ [Jian 86]
- 1986 : [Robson 86]
  - $O(1.2278^n)$ (espace polynomial)
  - $O(1.2108^n)$ (espace exponentiel)
- 1990 : $O(1.2737^n)$ [Shindo Tomita 90]
- 1990 : $O(1.2227^n)$ [Beigel99]
- 2001 : [Robson 01]
  - $O(1.2025^n)$ (espace polynomial)
  - $O(1.1889^n)$ (espace exponentiel)
- 2006 : $O(1.2210^n)$ [Fomin Grandoni Kratsch 06]

Regardons une généralisation des problèmes Domination et Ensemble Stable :

- problème d'énumération
- analyse « non classique » pour battre $O^*(2^n)$
- conséquences combinatoires
- étude de bornes inférieures

(σ, ϱ)-domination

(σ, ϱ)-domination

Définition

- problème introduit par J.A. Telle en 1994 ;
- généralise beaucoup de problèmes de type domination.

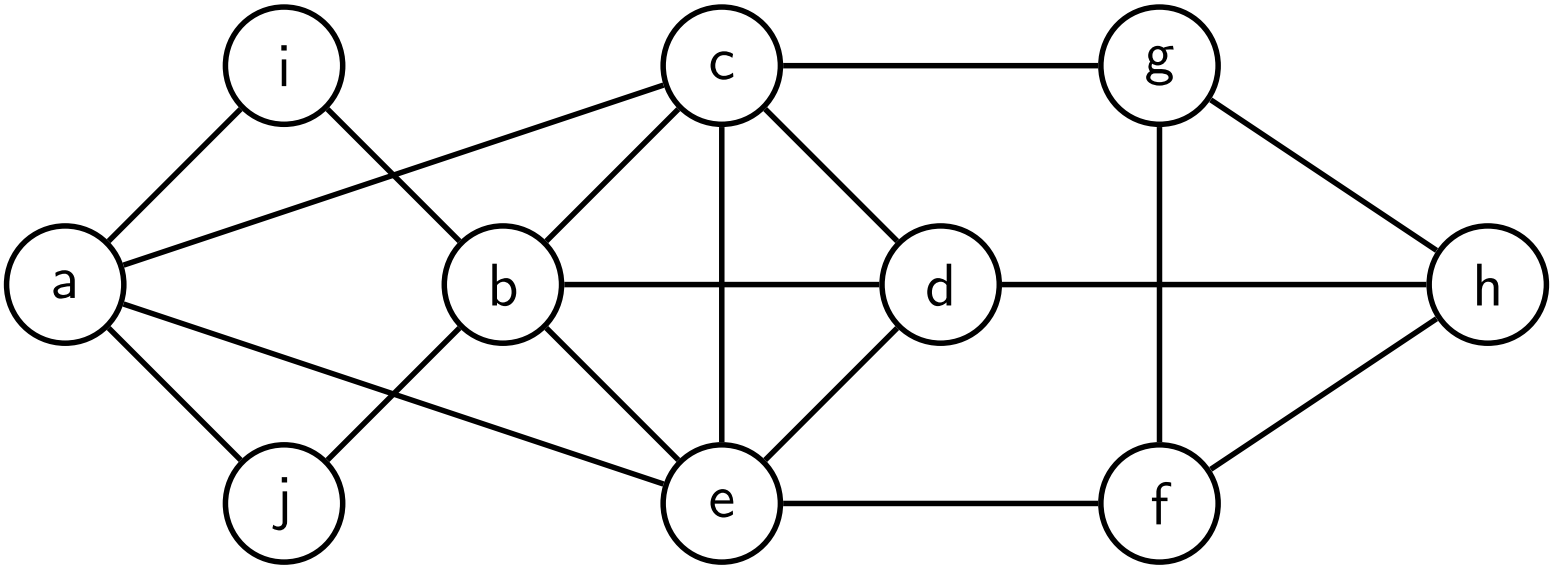
(σ, ϱ)-Domination

Étant donné un graphe $G = (V, E)$, un ensemble $S \subseteq V$ est (σ, ϱ)-Dominant ssi

- pour tout $v \in S$, $\quad |N(v) \cap S| \in \sigma$;
- pour tout $v \notin S$, $\quad |N(v) \cap S| \in \varrho$.

Mnémonique : σ pour **S**électionné, ϱ pour **R**ejeté

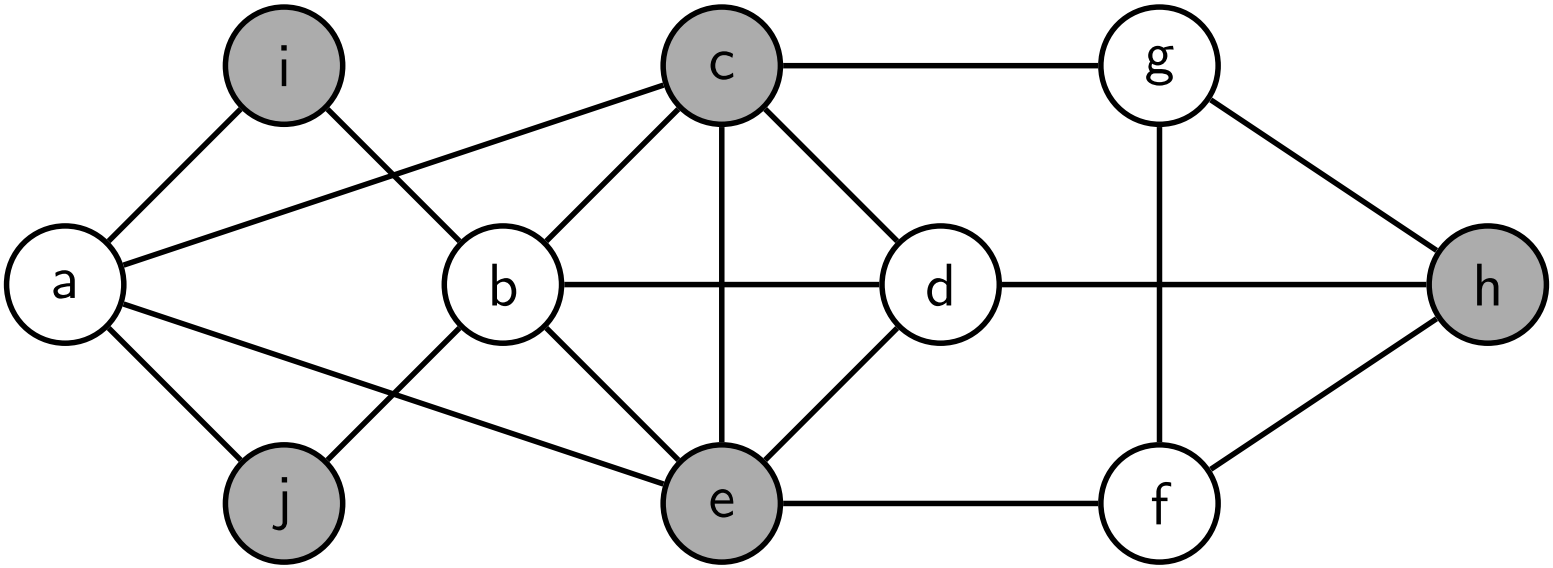
Exemple



Soit $\sigma = \{0, 1\}$ et $\varrho = \{2, 3, 4\}$.

Mnémonique : σ pour **S**électionné, ϱ pour **R**ejeté

Exemple



Soit $\sigma = \{0, 1\}$ et $\varrho = \{2, 3, 4\}$.

Mnémonique : σ pour **S**électionné, ϱ pour **R**ejeté

Beaucoup de problèmes ...

De nombreux problèmes bien connus sont modélisables :

σ	ϱ	(σ, ϱ)-Domination
$\mathbb{N}$	$\mathbb{N}^* = \mathbb{N} \setminus \{0\}$	ensemble dominant
$\mathbb{N}$	$\{r, r+1, \ldots\}$	ensemble r-dominant
$\{0\}$	$\mathbb{N}$	ensemble stable
$\mathbb{N}$	$\{1\}$	ensemble dominant parfait
$\{0\}$	$\{1\}$	code parfait
$\{0\}$	$\{0, 1\}$	ensemble stable fort
$\{0\}$	$\mathbb{N}^*$	ensemble stable dominant
$\mathbb{N}^*$	$\mathbb{N}^*$	ensemble dominant total
$\{1\}$	$\{1\}$	ensemble dominant total parfait
$\{1\}$	$\mathbb{N}$	couplage induit
$\{0, \ldots, r\}$	$\mathbb{N}$	sous-graphe induit de degré borné
$\{r\}$	$\mathbb{N}$	sous-graphe induit r-régulier

Étant donné un graphe G, on s'intéresse à

∃ ensemble (σ, ϱ)-dominant

Est-ce que G a un ensemble (σ, ϱ)-dominant ?

Enum ensembles (σ, ϱ)-dominants

Lister tous les ensembles (σ, ϱ)-dominants de G.

ensembles (σ, ϱ)-dominants

Déterminer le nombre d'ensembles (σ, ϱ)-dominants de G.

Max ensemble (σ, ϱ)-dominant

Trouver un ensemble (σ, ϱ)-dominant de G de taille maximum.

Min ensemble (σ, ϱ)-dominant

Trouver un ensemble (σ, ϱ)-dominant de G de taille minimum.

Étant donné un graphe G, on s'intéresse à

∃ ensemble (σ, ϱ)-dominant

Est-ce que G a un ensemble (σ, ϱ)-dominant ?

Enum ensembles (σ, ϱ)-dominants

Lister tous les ensembles (σ, ϱ)-dominants de G.

ensembles (σ, ϱ)-dominants

Déterminer le nombre d'ensembles (σ, ϱ)-dominants de G.

Max ensemble (σ, ϱ)-dominant

Trouver un ensemble (σ, ϱ)-dominant de G de taille maximum.

Min ensemble (σ, ϱ)-dominant

Trouver un ensemble (σ, ϱ)-dominant de G de taille minimum.

Résultats connus

Théorème : [Telle 94]

Les problèmes $\exists(\sigma = \{0\}, \varrho = \{q, q+1, q+2, \ldots\})$-DOMINANT sont NP-complets pour tout $q \geq 2$.

Le problème $\exists(\sigma = \{1\}, \varrho = \mathbb{N}^*)$-DOMINANT est NP-complet (couplage induit dominant).

Théorème : [Telle 94]

Soit $\sigma = \{0, 1\}$ and $\varrho = \mathbb{N}^*$.
MAX-(σ, ϱ)-DOMINANT et MIN-(σ, ϱ)-DOMINANT sont NP-difficiles, alors que $\exists(\sigma, \varrho)$-DOMINANT est polynomial.

Résultats connus

Théorème : [Telle 94]

Les problèmes $\exists(\sigma = \{0\}, \varrho = \{q, q+1, q+2, \ldots\})$-DOMINANT sont NP-complets pour tout $q \geq 2$.

Le problème $\exists(\sigma = \{1\}, \varrho = \mathbb{N}^*)$-DOMINANT est NP-complet (couplage induit dominant).

Théorème : [Telle 94]

Soit $\sigma = \{0, 1\}$ and $\varrho = \mathbb{N}^*$.
MAX-(σ, ϱ)-DOMINANT et MIN-(σ, ϱ)-DOMINANT sont NP-difficiles, alors que $\exists(\sigma, \varrho)$-DOMINANT est polynomial.

Résultats connus

Soit σ et ϱ finis tels que $0 \notin \varrho$.

Théorème : [Telle 94]

Le problème $\exists(\sigma, \varrho)$-DOMINANT est NP-complet.

Théorème : [Kratochvíl Manuel Miller 95]

Le problème $\exists(\sigma, \varrho)$-DOMINANT est polynomial sur les graphes d'intervalles.

Résultats connus

Soit σ et ϱ finis tels que $0 \notin \varrho$.

Théorème : [Telle 94]

Le problème $\exists(\sigma, \varrho)$-DOMINANT est NP-complet.

Théorème : [Kratochvíl Manuel Miller 95]

Le problème $\exists(\sigma, \varrho)$-DOMINANT est polynomial sur les graphes d'intervalles.

Résultats connus

Théorème : [Golovach Kratochvíl 07]

Pour σ, ϱ finis, $\exists(\sigma, \varrho)$-DOMINANT est polynomial sur les graphes cordaux si tous les graphes cordaux ont au plus un ensemble (σ, ϱ)-Dominant, sinon il est NP-complet.

Pour une paire (σ, ϱ) d'ensembles finis, il peut être décidé en temps fini s'il existe un graphe cordal avec deux ensembles (σ, ϱ)-Dominants.

Théorème : [Golovach Kratochvíl 07]

Soit σ et ϱ finis tels que $\varrho \neq \{0\}$.
Le problème MAX-(σ, ϱ)-DOMINANT est NP-difficile sur les graphes cordaux.

Résultats connus

Théorème : [Golovach Kratochvíl 07]

Pour σ, ϱ finis, $\exists(\sigma, \varrho)$-DOMINANT est polynomial sur les graphes cordaux si tous les graphes cordaux ont au plus un ensemble (σ, ϱ)-Dominant, sinon il est NP-complet.

Pour une paire (σ, ϱ) d'ensembles finis, il peut être décidé en temps fini s'il existe un graphe cordal avec deux ensembles (σ, ϱ)-Dominants.

Théorème : [Golovach Kratochvíl 07]

Soit σ et ϱ finis tels que $\varrho \neq \{0\}$.
Le problème MAX-(σ, ϱ)-DOMINANT est NP-difficile sur les graphes cordaux.

Nos résultats

Nous avons résolu :

ENUM ENSEMBLES (σ, ϱ)-DOMINANTS en temps $O^*(c^n)$, avec $c < 2$, si σ est sans successeur (ne contient pas deux entiers consécutifs), et

- σ et ϱ sont finis ; ou
- σ ou ϱ est fini et $\sigma \cap \varrho = \emptyset$.

les problèmes $\exists$, $\#$, MAX, MIN ENSEMBLE (σ, ϱ)-DOMINANT

- en temps $O^*(2^{n/2})$ si $|\sigma| + |\varrho| = 2$;
- en temps $O^*(3^{n/2})$ si $|\sigma| + |\varrho| = 3$.

Nos résultats

Nous avons résolu :

ENUM ENSEMBLES (σ, ϱ)-DOMINANTS en temps $O^*(c^n)$, avec $c < 2$, si σ est sans successeur (ne contient pas deux entiers consécutifs), et

- σ et ϱ sont finis ; ou
- σ ou ϱ est fini et $\sigma \cap \varrho = \emptyset$.

les problèmes $\exists$, $\#$, MAX, MIN ENSEMBLE (σ, ϱ)-DOMINANT

- en temps $O^*(2^{n/2})$ si $|\sigma| + |\varrho| = 2$;
- en temps $O^*(3^{n/2})$ si $|\sigma| + |\varrho| = 3$.

Nos résultats

Nous avons résolu :

en utilisant un mécanisme de rechargement

« algorithme Brancher & Recharger »

les problèmes $\exists$, $\#$, Max, Min ensemble (σ, ϱ)-dominant

- en temps $O^*(2^{n/2})$ si $|\sigma| + |\varrho| = 2$;
- en temps $O^*(3^{n/2})$ si $|\sigma| + |\varrho| = 3$.

Nos résultats

Nous avons résolu :

en utilisant un mécanisme de rechargement

« algorithme Brancher & Recharger »

en triant puis en cherchant

« algorithme Trier & Chercher »

Nos résultats - conditions nécessaires

La condition σ sans successeur est essentielle pour l'algorithme, mais pas suffisante pour énumérer plus vite qu'en $O^*(2^n)$.

- Si σ et ϱ sont infinis et sans successeur :
- Si σ est sans successeur et fini mais $\sigma \cap \varrho \neq \emptyset$:

Nos résultats - conditions nécessaires

La condition σ sans successeur est essentielle pour l'algorithme, mais pas suffisante pour énumérer plus vite qu'en $O^*(2^n)$.

- Si σ et ϱ sont infinis et sans successeur :

  Soit σ l'ensemble des entiers pairs et
  soit ϱ l'ensemble des entiers impairs.

  Alors $G = K_n$ a 2^{n-1} ensembles (σ, ϱ)-dominants

- Si σ est sans successeur et fini mais $\sigma \cap \varrho \neq \emptyset$:

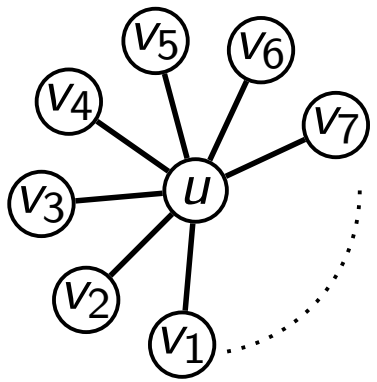
Nos résultats - conditions nécessaires

La condition σ sans successeur est essentielle pour l'algorithme, mais pas suffisante pour énumérer plus vite qu'en $O^*(2^n)$.

- Si σ et ϱ sont infinis et sans successeur :
- Si σ est sans successeur et fini mais $\sigma \cap \varrho \neq \emptyset$:

  Soit $\sigma = \{0\}$ et soit $\varrho = \mathbb{N}$.

  Alors $G = K_{1,n-1}$ a $2^{n-1} + 1$ ensembles (σ, ϱ)-dominants.



Nos résultats - conditions nécessaires

La condition σ sans successeur est essentielle pour l'algorithme, mais pas suffisante pour énumérer plus vite qu'en $O^*(2^n)$.

- Si σ et ϱ sont infinis et sans successeur :
- Si σ est sans successeur et fini mais $\sigma \cap \varrho \neq \emptyset$:

C'est pourquoi on impose des restrictions supplémentaires :

- σ et ϱ sont finis ; ou
- σ ou ϱ est fini et $\sigma \cap \varrho = \emptyset$.

Nos résultats - conditions nécessaires

Condition σ sans successeur et

- σ et ϱ sont finis ; ou
- σ ou ϱ est fini et $\sigma \cap \varrho = \emptyset$.

Problèmes bien connus vérifiant ces conditions :

σ	ϱ	(σ, ϱ)-Domination
$\{0\}$	$\{1\}$	code parfait
$\{0\}$	$\{0, 1\}$	ensemble stable fort
$\{0\}$	$\mathbb{N}^*$	ensemble stable dominant
$\{1\}$	$\{1\}$	ensemble dominant total parfait

L'algorithme Brancher & Recharger

Énumération de tous les ensembles (σ, ϱ)-dominants en utilisant :

- règles de réduction *(réduire ou simplifier l'instance)* ;
- une simple règle de branchement
  (résoudre récursivement le problème) ;
- un mécanisme de rechargement
  (principalement pour établir le temps d'exécution).

L'algorithme Brancher & Recharger

Énumération de tous les ensembles (σ, ϱ)-dominants en utilisant :

- règles de réduction *(réduire ou simplifier l'instance)* ;
- une simple règle de branchement *(résoudre récursivement le problème)* ;
- un mécanisme de rechargement *(principalement pour établir le temps d'exécution)*.

L'algorithme Brancher & Recharger

Énumération de tous les ensembles (σ, ϱ)-dominants en utilisant :

- règles de réduction *(réduire ou simplifier l'instance)* ;
- une simple règle de branchement
  (résoudre récursivement le problème) ;
- un mécanisme de rechargement
  (principalement pour établir le temps d'exécution).

L'algorithme Brancher & Recharger

Énumération de tous les ensembles (σ, ϱ)-dominants en utilisant :

- règles de réduction *(réduire ou simplifier l'instance)* ;
- une simple règle de branchement
  (résoudre récursivement le problème) ;
- un mécanisme de rechargement
  (principalement pour établir le temps d'exécution).

La règle de branchement

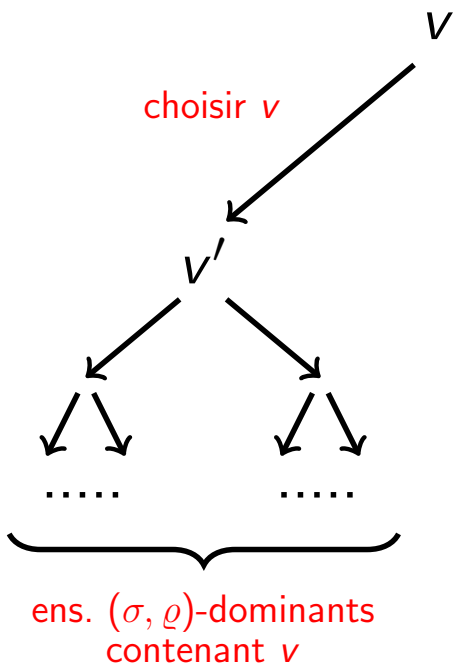
La règle de branchement est très simple.
Étant donné un sommet v sur lequel on veut brancher :

v

La règle de branchement

La règle de branchement est très simple.
Étant donné un sommet v sur lequel on veut brancher :

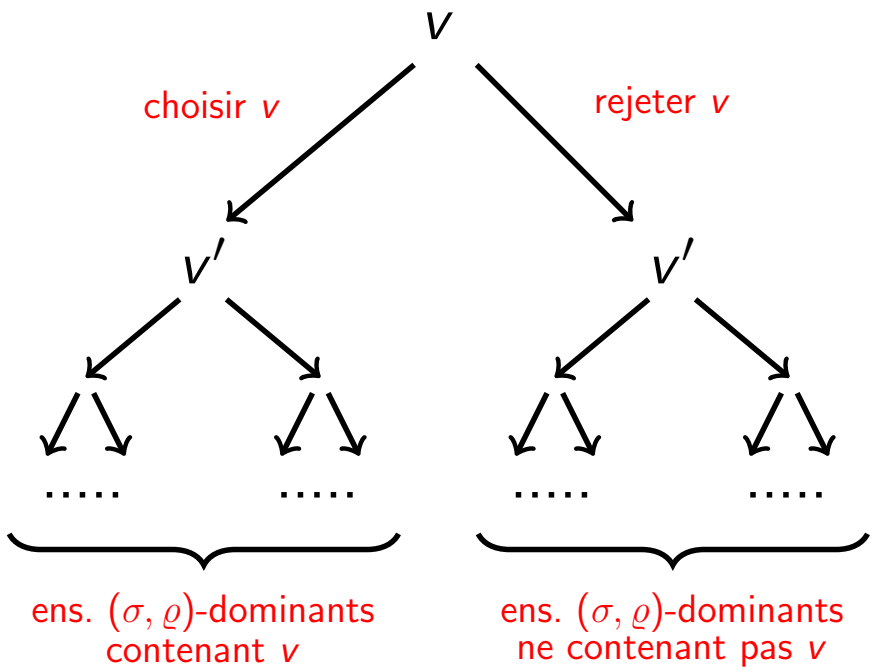
- « choisir v » et construire récursivement les ensembles (σ, ϱ)-dominants contenant v ;



La règle de branchement

La règle de branchement est très simple.
Étant donné un sommet v sur lequel on veut brancher :

- « choisir v » et construire récursivement les ensembles (σ, ϱ)-dominants contenant v ;
- « rejeter v » et construire récursivement les ensembles (σ, ϱ)-dominants ne contenant pas v.



La règle de branchement

On définit le poids d'un graphe G par $w(G) = \sum_{v \in V} w(v)$.

- Initialement, tous les sommets ont un poids de 1.
- Une fois qu'on a branché sur sommet v, $w(v) = 0$.

Le but est que, lorsqu'on branche sur un sommet v,

- soit v est rejeté et $w(G)$ décroît d'au moins 1 ;
- soit v est choisi et $w(G)$ décroît d'au moins $1 + \epsilon$.

Idée : lorsqu'on branche sur v, et qu'il est choisi, on prend ϵ d'un voisin de v sur lequel on n'a pas encore branché.

La règle de branchement

On définit le poids d'un graphe G par $w(G) = \sum_{v \in V} w(v)$.

- Initialement, tous les sommets ont un poids de 1.
- Une fois qu'on a branché sur sommet v, $w(v) = 0$.

Le but est que, lorsqu'on branche sur un sommet v,

- soit v est rejeté et $w(G)$ décroît d'au moins 1 ;
- soit v est choisi et $w(G)$ décroît d'au moins $1 + \epsilon$.

Idée : lorsqu'on branche sur v, et qu'il est choisi, on prend ϵ d'un voisin de v sur lequel on n'a pas encore branché.

La règle de branchement

On définit le poids d'un graphe G par $w(G) = \sum_{v \in V} w(v)$.

- Initialement, tous les sommets ont un poids de 1.
- Une fois qu'on a branché sur sommet v, $w(v) = 0$.

Le but est que, lorsqu'on branche sur un sommet v,

- soit v est rejeté et $w(G)$ décroît d'au moins 1 ;
- soit v est choisi et $w(G)$ décroît d'au moins $1 + \epsilon$.

Idée : lorsqu'on branche sur v, et qu'il est choisi, on prend ϵ d'un voisin de v sur lequel on n'a pas encore branché.

La règle de branchement

On définit le poids d'un graphe G par $w(G) = \sum_{v \in V} w(v)$.

- Initialement, tous les sommets ont un poids de 1.
- Une fois qu'on a branché sur sommet v, $w(v) = 0$.

Le but est que, lorsqu'on branche sur un sommet v,

- soit v est rejeté et $w(G)$ décroît d'au moins 1 ;
- soit v est choisi et $w(G)$ décroît d'au moins $1 + \epsilon$.

Idée : lorsqu'on branche sur v, et qu'il est choisi, on prend ϵ d'un voisin de v sur lequel on n'a pas encore branché.

La règle de branchement

On définit le poids d'un graphe G par $w(G) = \sum_{v \in V} w(v)$.

- Initialement, tous les sommets ont un poids de 1.
- Une fois qu'on a branché sur sommet v, $w(v) = 0$.

Le but est que, lorsqu'on branche sur un sommet v,

- soit v est rejeté et $w(G)$ décroît d'au moins 1 ;
- soit v est choisi et $w(G)$ décroît d'au moins $1 + \epsilon$.

Idée : lorsqu'on branche sur v, et qu'il est choisi, on prend ϵ d'un voisin de v sur lequel on n'a pas encore branché.

La valeur de ϵ

Soit $p = \max \sigma$ et soit $q = \max \varrho$.

La valeur de ϵ est donnée par :

- Si σ et ϱ sont finis

$$\epsilon = \frac{1}{1 + \max(p, q)}$$

- Si σ ou ϱ est fini et $\sigma \cap \varrho = \emptyset$.

$$\epsilon = \frac{1}{1 + \min(p, q)}$$

Le rechargement

Les règles de réduction se chargent des problèmes techniques :

- existence d'un voisin auquel on peut prendre ϵ ;
- pas de sommets de poids négatif ou nul ;
- ...

Procédure `Forcer-a`$(v, S, \overline{S}, w, H)$
  si $\exists x \in S$ *tel que* v *est son unique voisin libre* **alors**
    cas où
      $|N(x) \cap S| \in \sigma$ **alors** $\overline{S} := \overline{S} \cup \{v\}, w(v) := 0$;
      $|N(x) \cap S| + 1 \in \sigma$ **alors** $S := S \cup \{v\}, w(v) := 0$;
      $\{|N(x) \cap S|, |N(x) \cap S| + 1\} \cap \sigma = \emptyset$ **alors Stop;**
  si $\exists x$ *tel que* $|N(x) \cap S| > \max\{p, q\}$ **alors Stop;**

Procédure `Forcer-b`$(v, S, \overline{S}, w, H)$
  tant que ($\exists x$ *tel que* x *est libre et* $|N(x) \cap S| > \min\{p, q\}$) **ou**
  ($\exists y \in S$ *avec un unique voisin libre* z) **ou** ($\exists$ *un sommet libre* u *sans voisin libre*) **faire**
    soit x **ou** y, z **ou** u **de tels sommets;**
    cas où
      $|N(x) \cap S| > \max\{p, q\}$ **alors Stop;**
      $|N(x) \cap S| > p$ **alors** $\overline{S} := \overline{S} \cup \{x\}; w(x) := 0$;
      $|N(x) \cap S| > q$ **alors** $S := S \cup \{x\}; w(x) := 0$;
      $|N(y) \cap S| \in \sigma$ **alors** $\overline{S} := \overline{S} \cup \{z\}; w(z) := 0$;
      $|N(y) \cap S| + 1 \in \sigma$ **alors** $S := S \cup \{z\}; w(z) := 0$;
      $\{|N(y) \cap S|, |N(y) \cap S| + 1\} \cap \sigma = \emptyset$ **alors Stop;**
      $|N(u) \cap S| \in \sigma$ **alors** $S := S \cup \{u\}; w(u) := 0$;
      $|N(u) \cap S| \in \varrho$ **alors** $\overline{S} := \overline{S} \cup \{u\}; w(u) := 0$;
      $|N(u) \cap S| \notin \sigma \cup \varrho$ **alors Stop;**

Le rechargement

Mais, lorsqu'on branche sur un sommet v, il est possible que des voisins de v aient pris des ϵ de $w(v)$, et donc $w(v) < 1$.

➤ Un ϵ a été pris de $w(v)$ uniquement par un voisin maintenant dans l'ensemble (σ, ϱ)-dominant.

➤ Les règles de réduction garantissent que chaque voisin de v dans l'ensemble (σ, ϱ)-dominant, a un autre voisin sur lequel on n'a pas encore branché.

Le rechargement

Mais, lorsqu'on branche sur un sommet v, il est possible que des voisins de v aient pris des ϵ de $w(v)$, et donc $w(v) < 1$.

On doit « recharger » v !

➤ Un ϵ a été pris de $w(v)$ uniquement par un voisin maintenant dans l'ensemble (σ, ϱ)-dominant.

➤ Les règles de réduction garantissent que chaque voisin de v dans l'ensemble (σ, ϱ)-dominant, a un autre voisin sur lequel on n'a pas encore branché.

Le rechargement

Mais, lorsqu'on branche sur un sommet v, il est possible que des voisins de v aient pris des ϵ de $w(v)$, et donc $w(v) < 1$.

On doit « recharger » v !

➤ Un ϵ a été pris de $w(v)$ uniquement par un voisin maintenant dans l'ensemble (σ, ϱ)-dominant.

➤ Les règles de réduction garantissent que chaque voisin de v dans l'ensemble (σ, ϱ)-dominant, a un autre voisin sur lequel on n'a pas encore branché.

Le rechargement

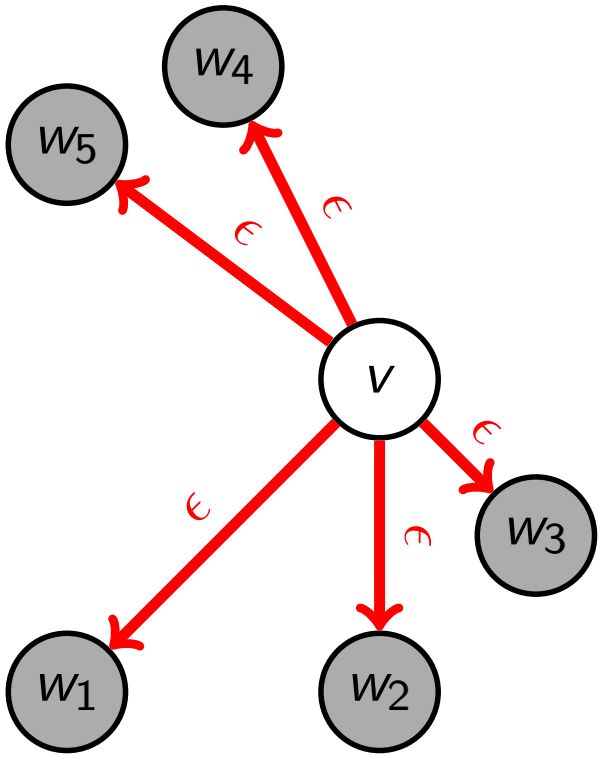
Mais, lorsqu'on branche sur un sommet v, il est possible que des voisins de v aient pris des ϵ de $w(v)$, et donc $w(v) < 1$.

On doit « recharger » v !

➤ Un ϵ a été pris de $w(v)$ uniquement par un voisin maintenant dans l'ensemble (σ, ϱ)-dominant.

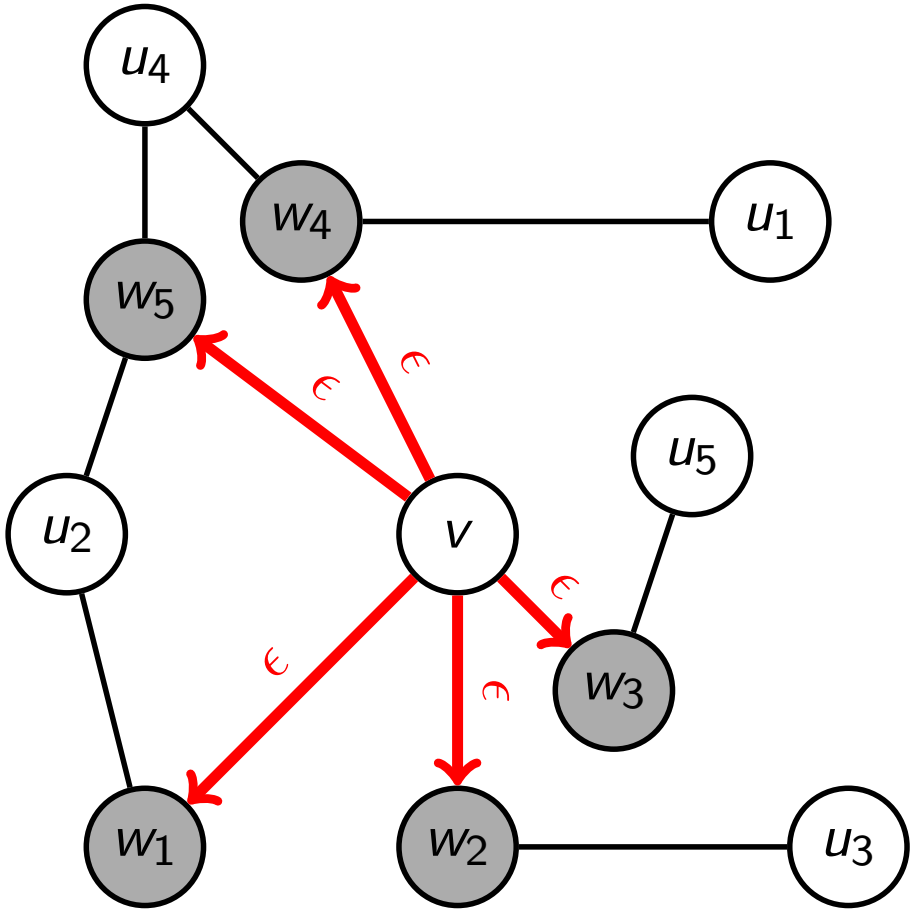
➤ Les règles de réduction garantissent que chaque voisin de v dans l'ensemble (σ, ϱ)-dominant, a un autre voisin sur lequel on n'a pas encore branché.

Le rechargement



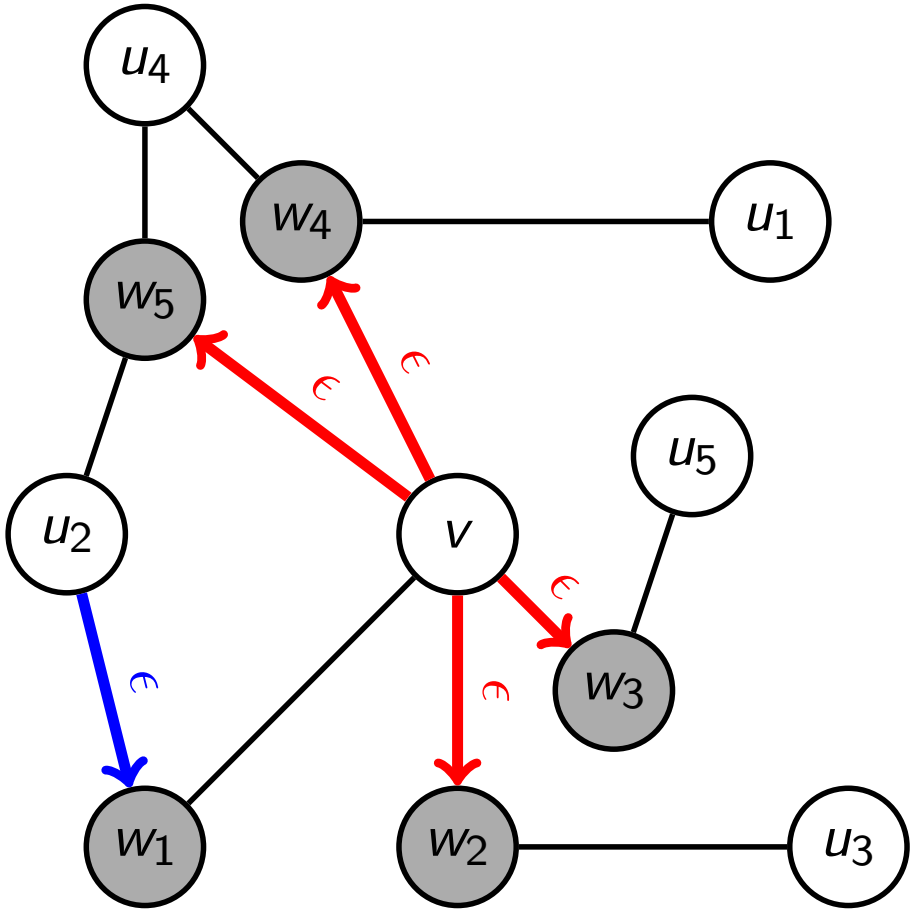
Les sommets en gris sont dans l'ensemble (σ, ϱ)-dominant ; on n'a pas encore branché sur les autres sommets.

Le rechargement



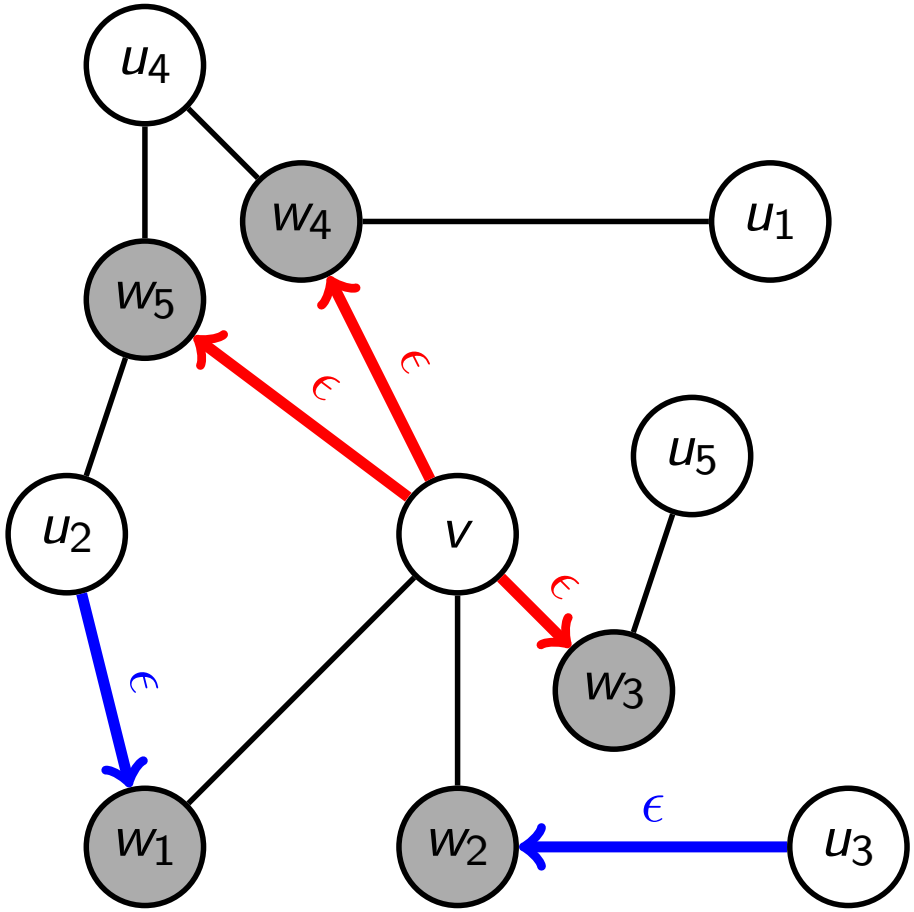
Les sommets en gris sont dans l'ensemble (σ, ϱ)-dominant ; on n'a pas encore branché sur les autres sommets.

Le rechargement



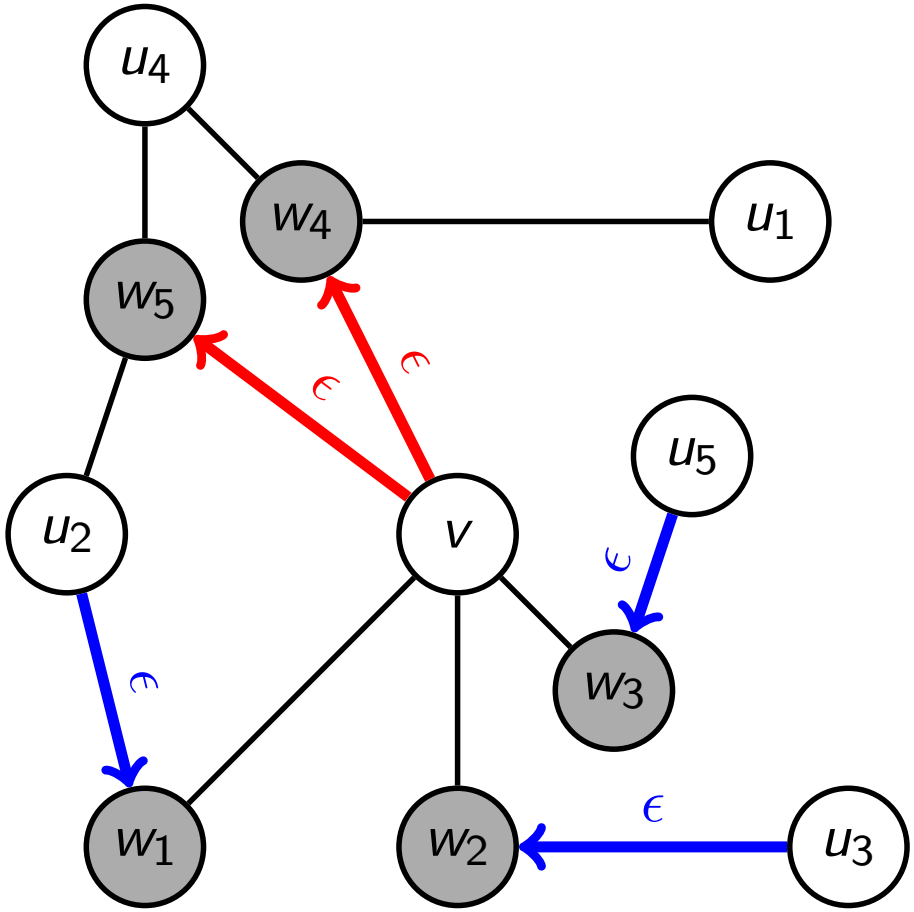
Les sommets en gris sont dans l'ensemble (σ, ϱ)-dominant ; on n'a pas encore branché sur les autres sommets.

Le rechargement



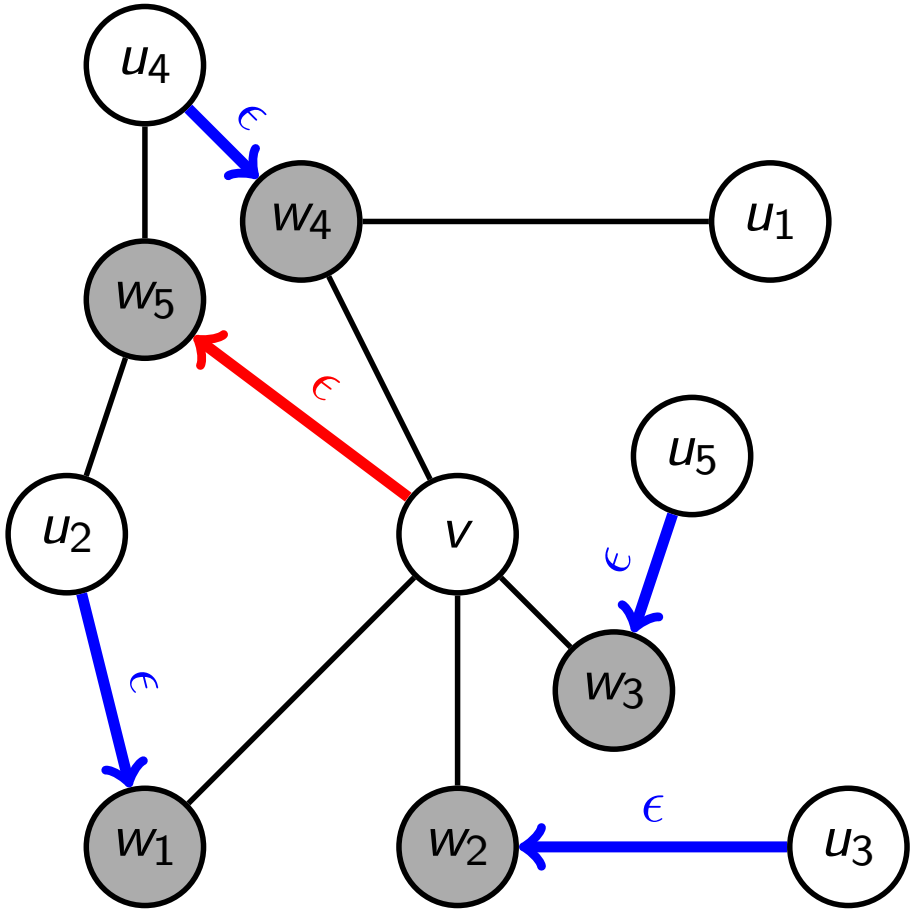
Les sommets en gris sont dans l'ensemble (σ, ϱ)-dominant ; on n'a pas encore branché sur les autres sommets.

Le rechargement



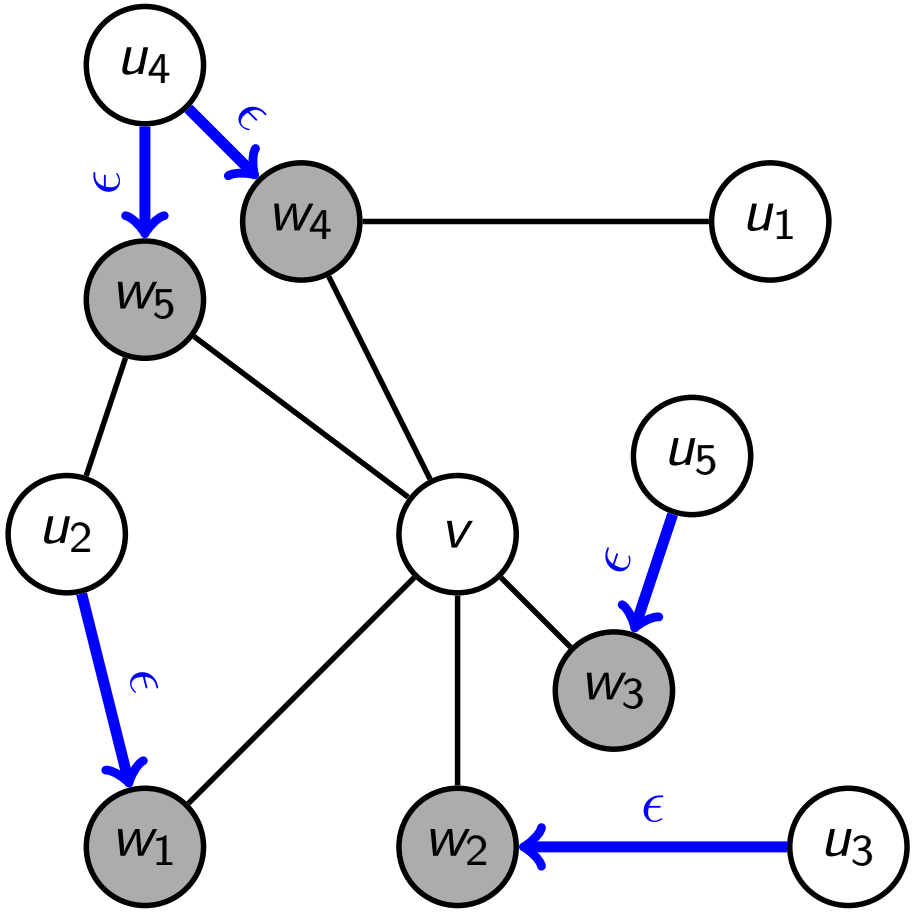
Les sommets en gris sont dans l'ensemble (σ, ϱ)-dominant ; on n'a pas encore branché sur les autres sommets.

Le rechargement



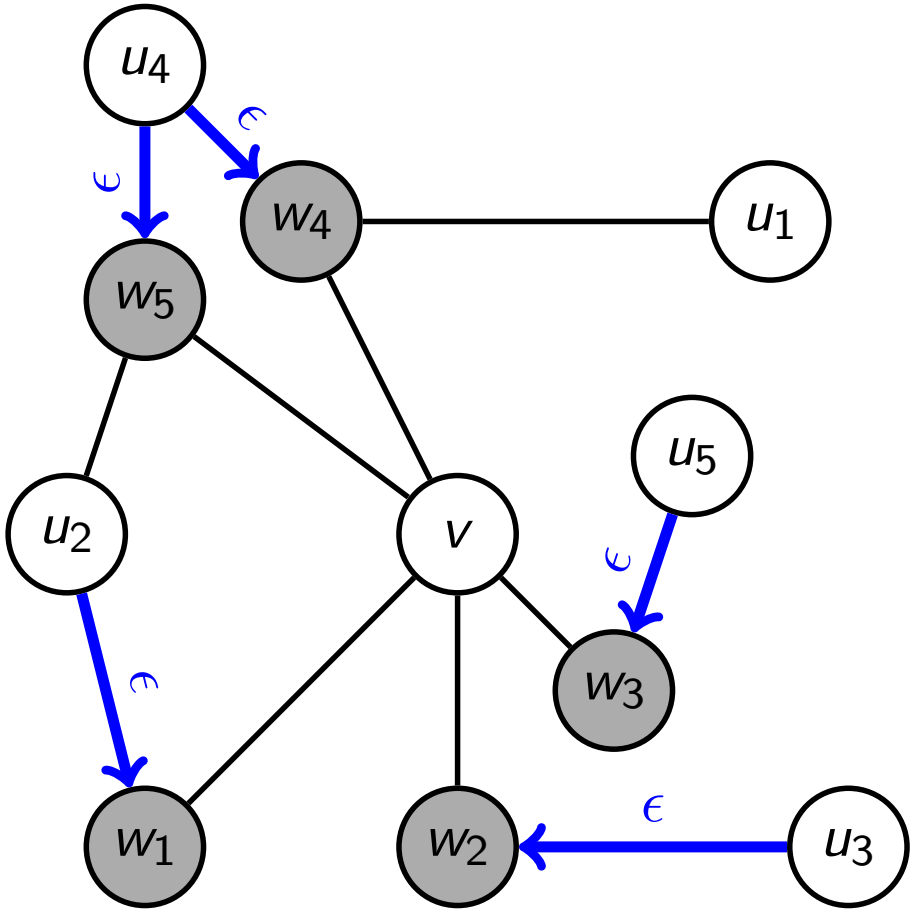
Les sommets en gris sont dans l'ensemble (σ, ϱ)-dominant ; on n'a pas encore branché sur les autres sommets.

Le rechargement



Les sommets en gris sont dans l'ensemble (σ, ϱ)-dominant ; on n'a pas encore branché sur les autres sommets.

Le rechargement



Les sommets en gris sont dans l'ensemble (σ, ϱ)-dominant ;
on n'a pas encore branché sur les autres sommets.

$$w(v) = 1$$

Le temps d'exécution

Soit $T\big(w(G)\big)$ le temps d'exécution de l'algorithme sur un graphe G de poids $w(G)$.

Seule la règle de branchement contribue à la croissance exponentielle de $T\big(w(G)\big)$ et la récurrence correspondante est :

$$T\big(w(G)\big) = T\big(\underbrace{w(G) - (1+\epsilon)}_{\text{choisir un sommet}}\big) + T\big(\underbrace{w(G) - 1}_{\text{rejeter un sommet}}\big).$$

La valeur de ϵ, et donc du temps d'exécution, dépend de σ et de ϱ.

Le temps d'exécution

Soit $T\big(w(G)\big)$ le temps d'exécution de l'algorithme sur un graphe G de poids $w(G)$.

Seule la règle de branchement contribue à la croissance exponentielle de $T\big(w(G)\big)$ et la récurrence correspondante est :

$$T\big(w(G)\big) = T\big(\underbrace{w(G) - (1+\epsilon)}_{\text{choisir un sommet}}\big) + T\big(\underbrace{w(G) - 1}_{\text{rejeter un sommet}}\big).$$

La valeur de ϵ, et donc du temps d'exécution, dépend de σ et de ϱ.

Le temps d'exécution

Théorème :

Sous les hypothèses précédentes sur σ et ϱ, l'algorithme énumère tous les ensembles (σ, ϱ)-dominants en temps $O^*(c^n)$, avec $c < 2$.

(Formellement, c est la plus grande racine de $x^{1+\epsilon} - x^{\epsilon} - 1 = 0$.)

φ	0	1	2	3	4	5	6	100
c	1.6181	1.7549	1.8192	1.8567	1.8813	1.8987	1.9116	1.9932

où

- $\varphi = \max(\max \sigma, \max \varrho)$, si σ et ϱ sont finis ;
- $\varphi = \min(\max \sigma, \max \varrho)$, si σ ou ϱ est fini et $\sigma \cap \varrho = \emptyset$.

Conséquence combinatoire

Une conséquence combinatoire de l'algorithme et de son analyse est que, sous les hypothèses précédentes sur σ et ϱ, tout graphe contient au plus c^n ensembles (σ, ϱ)-dominants, avec $c < 2$.

Conséquence combinatoire

Une conséquence combinatoire de l'algorithme et de son analyse est que, sous les hypothèses précédentes sur σ et ϱ, tout graphe contient au plus c^n ensembles (σ, ϱ)-dominants, avec $c < 2$.

Mais ces bornes ne sont très probablement pas atteintes !

Conséquence combinatoire

Une conséquence combinatoire de l'algorithme et de son analyse est que, sous les hypothèses précédentes sur σ et ϱ, tout graphe contient au plus c^n ensembles (σ, ϱ)-dominants, avec $c < 2$.

Mais ces bornes ne sont très probablement pas atteintes !

$\longrightarrow$ Il est donc intéressant de chercher des bornes inférieures.

Bornes combinatoires inférieures

Soit $r \geq 2$ un entier positif.
Soit σ l'ensemble des entiers pairs de $[0, r-1]$.
Soit ϱ l'ensemble des entiers impairs de $[0, r-1]$.

Considérons $G = s\ K_r$ (s copies de K_r).

Bornes combinatoires inférieures

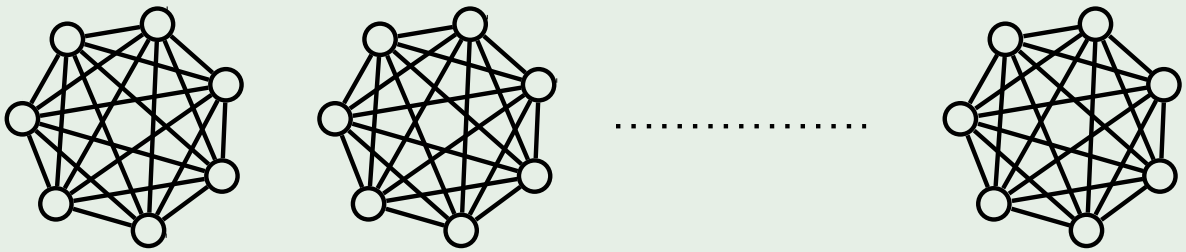
Soit $r \geq 2$ un entier positif.
Soit σ l'ensemble des entiers pairs de $[0, r-1]$.
Soit ϱ l'ensemble des entiers impairs de $[0, r-1]$.

Considérons $G = s\ K_r$ (s copies de K_r).
$\longrightarrow$ G contient $2^{(r-1)s} = 2^{\frac{r-1}{r}n}$ ensembles (σ, ϱ)-Dominants.

Bornes combinatoires inférieures

Soit $r \geq 2$ un entier positif.
Soit σ l'ensemble des entiers pairs de $[0, r-1]$.
Soit ϱ l'ensemble des entiers impairs de $[0, r-1]$.

Considérons $G = s\ K_r$ (s copies de K_r).
$\longrightarrow$ G contient $2^{(r-1)s} = 2^{\frac{r-1}{r}n}$ ensembles (σ, ϱ)-Dominants.



Un graphe $G = s\ K_7$ pour lequel chaque ensemble de cardinalité impaire d'un K_7 est un ensemble (σ, ϱ)-Dominant.

$(\sigma = \{0, 2, 4, 6\}$ et $\varrho = \{1, 3, 5\})$

Bornes combinatoires inférieures

Rappel :

$\sigma = \{\text{entiers pairs de [0,r-1]}\}$ $\varrho = \{\text{entiers impairs de [0,r-1]}\}$

Puisque pour un $r \geq 2$ fixé, σ et ϱ sont finis et $\sigma \cap \varrho = \emptyset$, alors les deux variantes de l'algorithmes peuvent être utilisées, c-à-d pour

$$\epsilon = \epsilon_{\max} = \frac{1}{1 + \max(p, q)} \quad \text{ou} \quad \epsilon = \epsilon_{\min} = \frac{1}{1 + \min(p, q)}.$$

r	2	3	4	...	8	9	...	101
avec $\epsilon_{\max}$	1.7549	1.8192	1.8567	...	1.9216	1.9296	...	1.9932
avec $\epsilon_{\min}$	1.6181	1.7549	1.8192	...	1.9116	1.9216	...	1.9932
$2^{\frac{r-1}{r}}$	1.4142	1.5874	1.6817	...	1.8340	1.8517	...	1.9863

Bornes combinatoires inférieures

Rappel :

$\sigma = \{\text{entiers pairs de [0,r-1]}\}$ $\varrho = \{\text{entiers impairs de [0,r-1]}\}$

Puisque pour un $r \geq 2$ fixé, σ et ϱ sont finis et $\sigma \cap \varrho = \emptyset$, alors les deux variantes de l'algorithmes peuvent être utilisées, c-à-d pour

$$\epsilon = \epsilon_{\max} = \frac{1}{1 + \max(p, q)} \quad \text{ou} \quad \epsilon = \epsilon_{\min} = \frac{1}{1 + \min(p, q)}.$$

r	2	3	4	...	8	9	...	101
avec $\epsilon_{\max}$	1.7549	1.8192	1.8567	...	1.9216	1.9296	...	1.9932
avec $\epsilon_{\min}$	1.6181	1.7549	1.8192	...	1.9116	1.9216	...	1.9932
$2^{\frac{r-1}{r}}$	1.4142	1.5874	1.6817	...	1.8340	1.8517	...	1.9863

L'approche « Trier et Chercher »

En utilisant la technique « Trier et Chercher »,
nous pouvons résoudre :

les problèmes $\exists$, $\#$, Max, Min ensemble (σ, ϱ)-dominant

- en temps $O^*(2^{n/2})$ si $|\sigma| + |\varrho| = 2$;
- en temps $O^*(3^{n/2})$ si $|\sigma| + |\varrho| = 3$.

Dans la suite, regardons $|\sigma| + |\varrho| = 2$, avec $\sigma = \{p\}$ et $\varrho = \{q\}$.

L'approche « Trier et Chercher »

- technique peu connue, très peu utilisée
- 1974, Horowitz et Sahni ;
  1981, Schroeppel et Shamir

  $\longrightarrow$ algorithme en $O^*(2^{n/2})$ pour Sac à Dos Discret
- 2003, Woeginger

  $\longrightarrow$ algorithme en $O^*(2^{n/2})$ pour Subset Sum

L'approche « Trier et Chercher »

- technique peu connue, très peu utilisée
- 1974, Horowitz et Sahni ;
  1981, Schroeppel et Shamir

  $\longrightarrow$ algorithme en $O^*(2^{n/2})$ pour Sac à Dos Discret
- 2003, Woeginger

  $\longrightarrow$ algorithme en $O^*(2^{n/2})$ pour Subset Sum

L'approche « Trier et Chercher »

- technique peu connue, très peu utilisée
- 1974, Horowitz et Sahni ;
  1981, Schroeppel et Shamir

  $\longrightarrow$ algorithme en $O^*(2^{n/2})$ pour Sac à Dos Discret
- 2003, Woeginger

  $\longrightarrow$ algorithme en $O^*(2^{n/2})$ pour Subset Sum

L'approche « Trier et Chercher »

Description de la technique sur un problème « simple » :

SUBSET SUM

Entrée : un ensemble S de n entiers et un entier K.
Question : trouver un sous-ensemble $S' \subseteq S$ t.q. $\sum_{x \in S'} = K$.

L'approche « Trier et Chercher »

Description de la technique sur un problème « simple » :

SUBSET SUM

Entrée : un ensemble S de n entiers et un entier K.
Question : trouver un sous-ensemble $S' \subseteq S$ t.q. $\sum_{x \in S'} = K$.

Idée :

1. partitionner S en S_1 et S_2, $|S_1| \approx |S_2|$;
2. pour chaque ensemble $S' \subseteq S_1$, calculer $sum(S') \leftarrow \sum_{x \in S'}$;
3. trier les S' par valeur $sum(S')$ croissante (stocker dans une table de taille $2^{n/2}$) ;
4. pour chaque ensemble $S' \subseteq S_2$, calculer $sum(S_2) \leftarrow \sum_{x \in S'}$ et vérifier si la valeur $K - sum(S_2)$ est présente dans la table.

L'approche « Trier et Chercher »

Description de la technique sur un problème « simple » :

SUBSET SUM

Entrée : un ensemble S de n entiers et un entier K.
Question : trouver un sous-ensemble $S' \subseteq S$ t.q. $\sum_{x \in S'} = K$.

Idée :

1. partitionner S en S_1 et S_2, $|S_1| \approx |S_2|$;
2. pour chaque ensemble $S' \subseteq S_1$, calculer $sum(S') \leftarrow \sum_{x \in S'}$;
3. trier les S' par valeur $sum(S')$ croissante (stocker dans une table de taille $2^{n/2}$) ;
4. pour chaque ensemble $S' \subseteq S_2$, calculer $sum(S_2) \leftarrow \sum_{x \in S'}$ et vérifier si la valeur $K - sum(S_2)$ est présente dans la table.

L'approche « Trier et Chercher »

Description de la technique sur un problème « simple » :

SUBSET SUM

Entrée : un ensemble S de n entiers et un entier K.
Question : trouver un sous-ensemble $S' \subseteq S$ t.q. $\sum_{x \in S'} = K$.

Idée :

1. partitionner S en S_1 et S_2, $|S_1| \approx |S_2|$;
2. pour chaque ensemble $S' \subseteq S_1$, calculer $sum(S') \leftarrow \sum_{x \in S'}$;
3. trier les S' par valeur $sum(S')$ croissante (stocker dans une table de taille $2^{n/2}$) ;
4. pour chaque ensemble $S' \subseteq S_2$, calculer $sum(S_2) \leftarrow \sum_{x \in S'}$ et vérifier si la valeur $K - sum(S_2)$ est présente dans la table.

L'approche « Trier et Chercher »

Description de la technique sur un problème « simple » :

SUBSET SUM

Entrée : un ensemble S de n entiers et un entier K.
Question : trouver un sous-ensemble $S' \subseteq S$ t.q. $\sum_{x \in S'} = K$.

Idée :

1. partitionner S en S_1 et S_2, $|S_1| \approx |S_2|$;
2. pour chaque ensemble $S' \subseteq S_1$, calculer $sum(S') \leftarrow \sum_{x \in S'}$;
3. trier les S' par valeur $sum(S')$ croissante (stocker dans une table de taille $2^{n/2}$) ;
4. pour chaque ensemble $S' \subseteq S_2$, calculer $sum(S_2) \leftarrow \sum_{x \in S'}$ et vérifier si la valeur $K - sum(S_2)$ est présente dans la table.

L'approche « Trier et Chercher »

Description de la technique sur un problème « simple » :

SUBSET SUM

Entrée : un ensemble S de n entiers et un entier K.
Question : trouver un sous-ensemble $S' \subseteq S$ t.q. $\sum_{x \in S'} = K$.

Idée :

1. partitionner S en S_1 et S_2, $|S_1| \approx |S_2|$;
2. pour chaque ensemble $S' \subseteq S_1$, calculer $sum(S') \leftarrow \sum_{x \in S'}$;
3. trier les S' par valeur $sum(S')$ croissante (stocker dans une table de taille $2^{n/2}$) ;
4. pour chaque ensemble $S' \subseteq S_2$, calculer $sum(S_2) \leftarrow \sum_{x \in S'}$ et vérifier si la valeur $K - sum(S_2)$ est présente dans la table.

L'approche « Trier et Chercher »

Description de la technique sur un problème « simple » :

SUBSET SUM

Entrée : un ensemble S de n entiers et un entier K.
Question : trouver un sous-ensemble $S' \subseteq S$ t.q. $\sum_{x \in S'} = K$.

Idée :

1. partitionner S en S_1 et S_2, $|S_1| \approx |S_2|$;
2. pour chaque ensemble $S' \subseteq S_1$, calculer $sum(S') \leftarrow \sum_{x \in S'}$;
3. trier les S' par valeur $sum(S')$ croissante (stocker dans une table de taille $2^{n/2}$) ;
4. pour chaque ensemble $S' \subseteq S_2$, calculer $sum(S_2) \leftarrow \sum_{x \in S'}$ et vérifier si la valeur $K - sum(S_2)$ est présente dans la table.

$\longrightarrow$ temps d'exécution : $O^*(2^{n/2})$

L'approche « Trier et Chercher »

Généralisation de la technique :

- partitionner l'entrée en deux ensembles $\mathcal{I}_1$, $\mathcal{I}_2$
- pour chaque sous-ensemble de $\mathcal{I}_1$, calculer le *vecteur correspondant* [1] $(O^*(2^{|\mathcal{I}_1|}))$
- pour chaque sous-ensemble de $\mathcal{I}_2$, calculer le *vecteur correspondant* $(O^*(2^{|\mathcal{I}_2|}))$
- trier les vecteurs de $\mathcal{I}_2$ en ordre lexicographique $(O^*(2^{|\mathcal{I}_2|} \log 2^{|\mathcal{I}_2|}) = O^*(2^{|\mathcal{I}_2|}))$
- pour chaque vecteur de $\mathcal{I}_1$, chercher s'il existe un vecteur *adéquat* [2] dans $\mathcal{I}_2$ en utilisant une recherche dichotomique $(O^*(n2^{|\mathcal{I}_1|} \log 2^{|\mathcal{I}_2|}) = O^*(2^{|\mathcal{I}_1|}))$

[1] la définition et le calcul du vecteur dépendent du problème

[2] étant donné un vecteur, le vecteur adéquat dépend du problème

L'approche « Trier et Chercher »

Généralisation de la technique :

- partitionner l'entrée en deux ensembles $\mathcal{I}_1$, $\mathcal{I}_2$
- pour chaque sous-ensemble de $\mathcal{I}_1$, calculer le *vecteur correspondant* [1] $(O^*(2^{|\mathcal{I}_1|}))$
- pour chaque sous-ensemble de $\mathcal{I}_2$, calculer le *vecteur correspondant* $(O^*(2^{|\mathcal{I}_2|}))$
- trier les vecteurs de $\mathcal{I}_2$ en ordre lexicographique $(O^*(2^{|\mathcal{I}_2|} \log 2^{|\mathcal{I}_2|}) = O^*(2^{|\mathcal{I}_2|}))$
- pour chaque vecteur de $\mathcal{I}_1$, chercher s'il existe un vecteur *adéquat* [2] dans $\mathcal{I}_2$ en utilisant une recherche dichotomique $(O^*(n2^{|\mathcal{I}_1|} \log 2^{|\mathcal{I}_2|}) = O^*(2^{|\mathcal{I}_1|}))$

[1] la définition et le calcul du vecteur dépendent du problème

[2] étant donné un vecteur, le vecteur adéquat dépend du problème

L'approche « Trier et Chercher »

Généralisation de la technique :

- partitionner l'entrée en deux ensembles $\mathcal{I}_1$, $\mathcal{I}_2$
- pour chaque sous-ensemble de $\mathcal{I}_1$, calculer le *vecteur correspondant* [1] $(O^*(2^{|\mathcal{I}_1|}))$
- pour chaque sous-ensemble de $\mathcal{I}_2$, calculer le *vecteur correspondant* $(O^*(2^{|\mathcal{I}_2|}))$
- trier les vecteurs de $\mathcal{I}_2$ en ordre lexicographique $(O^*(2^{|\mathcal{I}_2|} \log 2^{|\mathcal{I}_2|}) = O^*(2^{|\mathcal{I}_2|}))$
- pour chaque vecteur de $\mathcal{I}_1$, chercher s'il existe un vecteur *adéquat* [2] dans $\mathcal{I}_2$ en utilisant une recherche dichotomique $(O^*(n2^{|\mathcal{I}_1|} \log 2^{|\mathcal{I}_2|}) = O^*(2^{|\mathcal{I}_1|}))$

[1] la définition et le calcul du vecteur dépendent du problème

[2] étant donné un vecteur, le vecteur adéquat dépend du problème

L'approche « Trier et Chercher »

Généralisation de la technique :

- partitionner l'entrée en deux ensembles $\mathcal{I}_1$, $\mathcal{I}_2$
- pour chaque sous-ensemble de $\mathcal{I}_1$, calculer le *vecteur correspondant* [1] $(O^*(2^{|\mathcal{I}_1|}))$
- pour chaque sous-ensemble de $\mathcal{I}_2$, calculer le *vecteur correspondant* $(O^*(2^{|\mathcal{I}_2|}))$
- trier les vecteurs de $\mathcal{I}_2$ en ordre lexicographique $(O^*(2^{|\mathcal{I}_2|} \log 2^{|\mathcal{I}_2|}) = O^*(2^{|\mathcal{I}_2|}))$
- pour chaque vecteur de $\mathcal{I}_1$, chercher s'il existe un vecteur *adéquat* [2] dans $\mathcal{I}_2$ en utilisant une recherche dichotomique $(O^*(n2^{|\mathcal{I}_1|} \log 2^{|\mathcal{I}_2|}) = O^*(2^{|\mathcal{I}_1|}))$

[1] la définition et le calcul du vecteur dépendent du problème

[2] étant donné un vecteur, le vecteur adéquat dépend du problème

L'approche « Trier et Chercher »

Généralisation de la technique :

- partitionner l'entrée en deux ensembles $\mathcal{I}_1$, $\mathcal{I}_2$
- pour chaque sous-ensemble de $\mathcal{I}_1$, calculer le *vecteur correspondant* [1] $(O^*(2^{|\mathcal{I}_1|}))$
- pour chaque sous-ensemble de $\mathcal{I}_2$, calculer le *vecteur correspondant* $(O^*(2^{|\mathcal{I}_2|}))$
- trier les vecteurs de $\mathcal{I}_2$ en ordre lexicographique $(O^*(2^{|\mathcal{I}_2|} \log 2^{|\mathcal{I}_2|}) = O^*(2^{|\mathcal{I}_2|}))$
- pour chaque vecteur de $\mathcal{I}_1$, chercher s'il existe un vecteur *adéquat* [2] dans $\mathcal{I}_2$ en utilisant une recherche dichotomique $(O^*(n2^{|\mathcal{I}_1|} \log 2^{|\mathcal{I}_2|}) = O^*(2^{|\mathcal{I}_1|}))$

[1] la définition et le calcul du vecteur dépendent du problème

[2] étant donné un vecteur, le vecteur adéquat dépend du problème

L'approche « Trier et Chercher »

Généralisation de la technique :

- partitionner l'entrée en deux ensembles $\mathcal{I}_1$, $\mathcal{I}_2$
- pour chaque sous-ensemble de $\mathcal{I}_1$, calculer le *vecteur correspondant* [1] $(O^*(2^{|\mathcal{I}_1|}))$
- pour chaque sous-ensemble de $\mathcal{I}_2$, calculer le *vecteur correspondant* $(O^*(2^{|\mathcal{I}_2|}))$
- trier les vecteurs de $\mathcal{I}_2$ en ordre lexicographique $(O^*(2^{|\mathcal{I}_2|} \log 2^{|\mathcal{I}_2|}) = O^*(2^{|\mathcal{I}_2|}))$
- pour chaque vecteur de $\mathcal{I}_1$, chercher s'il existe un vecteur *adéquat* [2] dans $\mathcal{I}_2$ en utilisant une recherche dichotomique $(O^*(n2^{|\mathcal{I}_1|} \log 2^{|\mathcal{I}_2|}) = O^*(2^{|\mathcal{I}_1|}))$

[1] la définition et le calcul du vecteur dépendent du problème

[2] étant donné un vecteur, le vecteur adéquat dépend du problème

L'approche « Trier et Chercher »

Généralisation de la technique :

- partitionner l'entrée en deux ensembles $\mathcal{I}_1$, $\mathcal{I}_2$
- pour chaque sous-ensemble de $\mathcal{I}_1$, calculer le *vecteur correspondant* [1] $(O^*(2^{|\mathcal{I}_1|}))$
- pour chaque sous-ensemble de $\mathcal{I}_2$, calculer le *vecteur correspondant* $(O^*(2^{|\mathcal{I}_2|}))$
- trier les vecteurs de $\mathcal{I}_2$ en ordre lexicographique $(O^*(2^{|\mathcal{I}_2|} \log 2^{|\mathcal{I}_2|}) = O^*(2^{|\mathcal{I}_2|}))$
- pour chaque vecteur de $\mathcal{I}_1$, chercher s'il existe un vecteur *adéquat* [2] dans $\mathcal{I}_2$ en utilisant une recherche dichotomique $(O^*(n2^{|\mathcal{I}_1|} \log 2^{|\mathcal{I}_2|}) = O^*(2^{|\mathcal{I}_1|}))$

Temps d'exécution : si $|\mathcal{I}_1| = |\mathcal{I}_2| = n/2$, alors $O^*(2^{n/2})$

[1] la définition et le calcul du vecteur dépendent du problème

[2] étant donné un vecteur, le vecteur adéquat dépend du problème

L'approche « Trier et Chercher »

Résoudre $\exists(\sigma = \{p\}, \varrho = \{q\})$-DOMINANT en $O^*(2^{n/2})$:

Soit $p, q \in \mathbb{N}$ et un graphe $G = (V, E)$. Soit $k = \lfloor n/2 \rfloor$.

- Partitionnons l'ensemble des sommets en

$$V_1 = \{v_1, v_2, \ldots, v_k\}$$

$$V_2 = \{v_{k+1}, v_{k+2}, \ldots, v_n\}$$

L'approche « Trier et Chercher »

Résoudre $\exists(\sigma = \{p\}, \varrho = \{q\})$-DOMINANT en $O^*(2^{n/2})$:

Soit $p, q \in \mathbb{N}$ et un graphe $G = (V, E)$. Soit $k = \lfloor n/2 \rfloor$.

- Partitionnons l'ensemble des sommets en

$$V_1 = \{v_1, v_2, \ldots, v_k\}$$

$$V_2 = \{v_{k+1}, v_{k+2}, \ldots, v_n\}$$

L'approche « Trier et Chercher »

- Pour chaque sous-ensemble $S_1 \subseteq V_1$, calculons le vecteur

$$\vec{s_1} = (x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$$

où

$$x_i = \begin{cases} p - |N(v_i) \cap S_1| & \text{si } 1 \leq i \leq k \text{ et } v_i \in S_1 \\ q - |N(v_i) \cap S_1| & \text{si } 1 \leq i \leq k \text{ et } v_i \notin S_1 \\ |N(v_i) \cap S_1| & \text{si } k+1 \leq i \leq n \end{cases}$$

- Pour chaque sous-ensemble $S_2 \subseteq V_2$, calculons le vecteur

$$\vec{s_2} = (x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$$

où

$$x_i = \begin{cases} |N(v_i) \cap S_2| & \text{si } 1 \leq i \leq k \\ p - |N(v_i) \cap S_2| & \text{si } k+1 \leq i \leq n \text{ et } v_i \in S_2 \\ q - |N(v_i) \cap S_2| & \text{si } k+1 \leq i \leq n \text{ et } v_i \notin S_2. \end{cases}$$

L'approche « Trier et Chercher »

- Pour chaque sous-ensemble $S_1 \subseteq V_1$, calculons le vecteur

$$\vec{s_1} = (x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$$

où

$$x_i = \begin{cases} p - |N(v_i) \cap S_1| & \text{si } 1 \leq i \leq k \text{ et } v_i \in S_1 \\ q - |N(v_i) \cap S_1| & \text{si } 1 \leq i \leq k \text{ et } v_i \notin S_1 \\ |N(v_i) \cap S_1| & \text{si } k+1 \leq i \leq n \end{cases}$$

- Pour chaque sous-ensemble $S_2 \subseteq V_2$, calculons le vecteur

$$\vec{s_2} = (x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$$

où

$$x_i = \begin{cases} |N(v_i) \cap S_2| & \text{si } 1 \leq i \leq k \\ p - |N(v_i) \cap S_2| & \text{si } k+1 \leq i \leq n \text{ et } v_i \in S_2 \\ q - |N(v_i) \cap S_2| & \text{si } k+1 \leq i \leq n \text{ et } v_i \notin S_2. \end{cases}$$

L'approche « Trier et Chercher »

- Pour chaque sous-ensemble $S_1 \subseteq V_1$, calculons le vecteur

$$\vec{s_1} = (x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$$

où

$$x_i = \begin{cases} p - |N(v_i) \cap S_1| & \text{si } 1 \leq i \leq k \text{ et } v_i \in S_1 \\ q - |N(v_i) \cap S_1| & \text{si } 1 \leq i \leq k \text{ et } v_i \notin S_1 \\ |N(v_i) \cap S_1| & \text{si } k+1 \leq i \leq n \end{cases}$$

- Pour chaque sous-ensemble $S_2 \subseteq V_2$, calculons le vecteur

$$\vec{s_2} = (x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$$

où

$$x_i = \begin{cases} |N(v_i) \cap S_2| & \text{si } 1 \leq i \leq k \\ p - |N(v_i) \cap S_2| & \text{si } k+1 \leq i \leq n \text{ et } v_i \in S_2 \\ q - |N(v_i) \cap S_2| & \text{si } k+1 \leq i \leq n \text{ et } v_i \notin S_2. \end{cases}$$

L'approche « Trier et Chercher »

- Trions les vecteurs correspondants à V_2 en ordre lexicogaphique (tri par base $O^*(n2^{n/2})$, tri fusion $O^*(2^{n/2} \log 2^{n/2})$)
- Pour chaque vecteur $\vec{s_1}$ (correspondant à un $S_1 \subseteq V_1$), vérifions s'il existe un $\vec{s_2}$ (correspondant à un $S_2 \subseteq V_2$), tel que $\vec{s_2} = \vec{s_1}$ (recherche dichotomique $O^*(n \log 2^{n/2})$)
- Si un tel couple $(\vec{s_1}, \vec{s_2})$ de vecteurs existe, alors $S_1 \cup S_2$ est un ensemble $(\{p\}, \{q\})$-Dominant

L'approche « Trier et Chercher »

- Trions les vecteurs correspondants à V_2 en ordre lexicogaphique (tri par base $O^*(n2^{n/2})$, tri fusion $O^*(2^{n/2} \log 2^{n/2})$)

- Pour chaque vecteur $\vec{s_1}$ (correspondant à un $S_1 \subseteq V_1$), vérifions s'il existe un $\vec{s_2}$ (correspondant à un $S_2 \subseteq V_2$), tel que $\vec{s_2} = \vec{s_1}$ (recherche dichotomique $O^*(n \log 2^{n/2})$)

- Si un tel couple $(\vec{s_1}, \vec{s_2})$ de vecteurs existe, alors $S_1 \cup S_2$ est un ensemble $(\{p\}, \{q\})$-Dominant

L'approche « Trier et Chercher »

- Trions les vecteurs correspondants à V_2 en ordre lexicogaphique (tri par base $O^*(n2^{n/2})$, tri fusion $O^*(2^{n/2} \log 2^{n/2})$)
- Pour chaque vecteur $\vec{s_1}$ (correspondant à un $S_1 \subseteq V_1$), vérifions s'il existe un $\vec{s_2}$ (correspondant à un $S_2 \subseteq V_2$), tel que $\vec{s_2} = \vec{s_1}$ (recherche dichotomique $O^*(n \log 2^{n/2})$)
- Si un tel couple $(\vec{s_1}, \vec{s_2})$ de vecteurs existe, alors $S_1 \cup S_2$ est un ensemble $(\{p\}, \{q\})$-Dominant

L'approche « Trier et Chercher »

- Trions les vecteurs correspondants à V_2 en ordre lexicogaphique (tri par base $O^*(n2^{n/2})$, tri fusion $O^*(2^{n/2} \log 2^{n/2})$)
- Pour chaque vecteur $\vec{s_1}$ (correspondant à un $S_1 \subseteq V_1$), vérifions s'il existe un $\vec{s_2}$ (correspondant à un $S_2 \subseteq V_2$), tel que $\vec{s_2} = \vec{s_1}$ (recherche dichotomique $O^*(n \log 2^{n/2})$)
- Si un tel couple $(\vec{s_1}, \vec{s_2})$ de vecteurs existe, alors $S_1 \cup S_2$ est un ensemble $(\{p\}, \{q\})$-Dominant

L'approche « Trier et Chercher »

- Trions les vecteurs correspondants à V_2 en ordre lexicogaphique (tri par base $O^*(n2^{n/2})$, tri fusion $O^*(2^{n/2} \log 2^{n/2})$)
- Pour chaque vecteur $\vec{s_1}$ (correspondant à un $S_1 \subseteq V_1$), vérifions s'il existe un $\vec{s_2}$ (correspondant à un $S_2 \subseteq V_2$), tel que $\vec{s_2} = \vec{s_1}$ (recherche dichotomique $O^*(n \log 2^{n/2})$)
- Si un tel couple $(\vec{s_1}, \vec{s_2})$ de vecteurs existe, alors $S_1 \cup S_2$ est un ensemble $(\{p\}, \{q\})$-Dominant

Temps d'exécution total : $O^*(2^{n/2})$

L'approche « Trier et Chercher »

$\longrightarrow$ l'algorithme peut être adpaté pour résoudre :

- # Ensembles (σ, ϱ)-Dominants

  Au lieu de seulement trier les vecteurs correspondant à V_2, les copies multiples sont supprimées et chaque vecteur est stocké avec son nombre d'occurences

- Max Ensemble (σ, ϱ)-Dominant
- Min Ensemble (σ, ϱ)-Dominant

  Au lieu de trier les vecteurs correspondant à V_2, les copies multiples sont supprimées et chaque vecteur est stocké avec un $S_2 \subseteq V_2$ de cardinalité maximum / minimum produisant ce vecteur

en temps $O^*(2^{n/2})$, avec $\sigma = \{p\}$ et $\varrho = \{q\}$

L'approche « Trier et Chercher »

$\longrightarrow$ l'algorithme peut être adpaté pour résoudre :

- # Ensembles (σ, ϱ)-Dominants

  Au lieu de seulement trier les vecteurs correspondant à V_2, les copies multiples sont supprimées et chaque vecteur est stocké avec son nombre d'occurences

- Max Ensemble (σ, ϱ)-Dominant
- Min Ensemble (σ, ϱ)-Dominant

  Au lieu de trier les vecteurs correspondant à V_2, les copies multiples sont supprimées et chaque vecteur est stocké avec un $S_2 \subseteq V_2$ de cardinalité maximum / minimum produisant ce vecteur

en temps $O^*(2^{n/2})$, avec $\sigma = \{p\}$ et $\varrho = \{q\}$

L'approche « Trier et Chercher »

$\longrightarrow$ l'algorithme peut être adpaté pour résoudre :

- # Ensembles (σ, ϱ)-Dominants

  Au lieu de seulement trier les vecteurs correspondant à V_2, les copies multiples sont supprimées et chaque vecteur est stocké avec son nombre d'occurences

- Max Ensemble (σ, ϱ)-Dominant
- Min Ensemble (σ, ϱ)-Dominant

  Au lieu de trier les vecteurs correspondant à V_2, les copies multiples sont supprimées et chaque vecteur est stocké avec un $S_2 \subseteq V_2$ de cardinalité maximum / minimum produisant ce vecteur

en temps $O^*(2^{n/2})$, avec $\sigma = \{p\}$ et $\varrho = \{q\}$

L'approche « Trier et Chercher »

$\longrightarrow$ l'algorithme peut être adpaté pour résoudre :

- # Ensembles (σ, ϱ)-Dominants

  Au lieu de seulement trier les vecteurs correspondant à V_2, les copies multiples sont supprimées et chaque vecteur est stocké avec son nombre d'occurences

- Max Ensemble (σ, ϱ)-Dominant
- Min Ensemble (σ, ϱ)-Dominant

  Au lieu de trier les vecteurs correspondant à V_2, les copies multiples sont supprimées et chaque vecteur est stocké avec un $S_2 \subseteq V_2$ de cardinalité maximum / minimum produisant ce vecteur

en temps $O^*(2^{n/2})$, avec $\sigma = \{p\}$ et $\varrho = \{q\}$

L'approche « Trier et Chercher »

$\longrightarrow$ l'algorithme peut être adpaté pour résoudre :

- # Ensembles (σ, ϱ)-Dominants

  Au lieu de seulement trier les vecteurs correspondant à V_2, les copies multiples sont supprimées et chaque vecteur est stocké avec son nombre d'occurences

- Max Ensemble (σ, ϱ)-Dominant
- Min Ensemble (σ, ϱ)-Dominant

  Au lieu de trier les vecteurs correspondant à V_2, les copies multiples sont supprimées et chaque vecteur est stocké avec un $S_2 \subseteq V_2$ de cardinalité maximum / minimum produisant ce vecteur

en temps $O^*(2^{n/2})$, avec $\sigma = \{p\}$ et $\varrho = \{q\}$

L'approche « Trier et Chercher »

Remarque : Enum Ensembles $(\{p\}, \{q\})$-Dominants ne peut être résolu en $O^*(2^{n/2})$.

Considérons le problème Code Parfait (i.e. $\sigma = \{0\}$, $\varrho = \{1\}$) et s copies de K_3.

Ce graphe a $3^s = 3^{n/3} > 2^{n/2}$ $(\{0\}, \{1\})$-DS.

Un graphe $G = s\ K_3$ pour lequel chaque ensemble contenant précisément un sommet de chaque K_3 est un Code Parfait.

L'approche « Trier et Chercher »

Remarque : Enum Ensembles $(\{p\}, \{q\})$-Dominants ne peut être résolu en $O^*(2^{n/2})$.

Considérons le problème Code Parfait (i.e. $\sigma = \{0\}$, $\varrho = \{1\}$) et s copies de K_3.

Ce graphe a $3^s = 3^{n/3} > 2^{n/2}$ $(\{0\}, \{1\})$-DS.

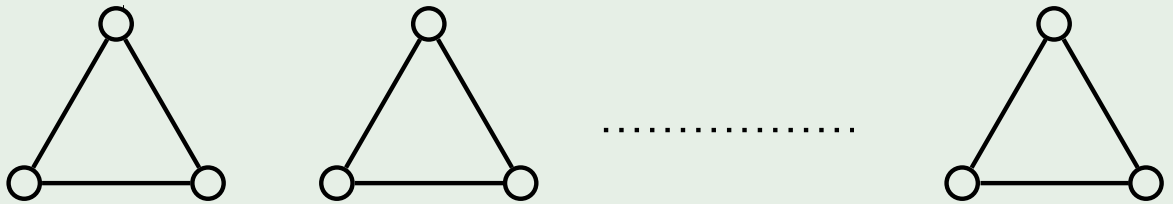
Un graphe $G = s\ K_3$ pour lequel chaque ensemble contenant précisément un sommet de chaque K_3 est un Code Parfait.

L'approche « Trier et Chercher »

Remarque : Enum Ensembles $(\{p\}, \{q\})$-Dominants ne peut être résolu en $O^*(2^{n/2})$.

Considérons le problème Code Parfait (i.e. $\sigma = \{0\}$, $\varrho = \{1\}$) et s copies de K_3.

Ce graphe a $3^s = 3^{n/3} > 2^{n/2}$ $(\{0\}, \{1\})$-DS.



Un graphe $G = s\ K_3$ pour lequel chaque ensemble contenant précisément un sommet de chaque K_3 est un Code Parfait.

Un algorithme « Trier et Chercher »

Dernière remarque :

L'approche peut être étendue pour résoudre

- $\sigma = \{p \mod m\}$ et $\varrho = \{q \mod m\}$;
- $|\sigma| + |\varrho| = 3$.

Conclusion

Conclusion - Questions

- Brancher & Réduire est une technique très puissante, mais les techniques d'analyse manquent encore de précision
- les bornes inférieures renseignent sur la précision de l'analyse
- L'analyse de ces algorithmes d'énumération implique des bornes combinatoires difficiles à obtenir par une analyse classique

Le problème (σ, ϱ)-Domination pourrait être étudié pour d'autres ensembles σ et ϱ ; en particulier :

- σ fini et $\varrho = \mathbb{N}^*$
- σ et ϱ co-finis

Ces deux restrictions modélisent de nombreux problèmes (Ensemble Stable, Couplage Induit, ...) ou (Domination, Domination Totale, ...).

Conclusion - Questions

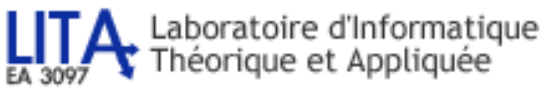
- Brancher & Réduire est une technique très puissante, mais les techniques d'analyse manquent encore de précision
- les bornes inférieures renseignent sur la précision de l'analyse
- L'analyse de ces algorithmes d'énumération implique des bornes combinatoires difficiles à obtenir par une analyse classique

Le problème (σ, ϱ)-Domination pourrait être étudié pour d'autres ensembles σ et ϱ ; en particulier :

- σ fini et $\varrho = \mathbb{N}^*$
- σ et ϱ co-finis

*Ces deux restrictions modélisent de nombreux problèmes (*Ensemble Stable, Couplage Induit, ...*) ou (*Domination, Domination Totale, ...*).*

Merci pour votre attention.



Bibliographie I


Beigel, R.,
Finding maximum independent sets in sparse and general graphs,
Proc. of SODA 1999, pp. 856–857.


Bodlaender, H. L., J. R. Gilbert, H. Hafsteinsson, T. Kloks, Approximating treewidth, pathwidth, frontsize and shortest elimination tree, *Journal of Algorithms* **18** (1995), p. 238–255.


Cook, S.,
The Complexity of Theorem Proving Procedures,
Proc. of the third annual ACM symposium on Theory of computing, 1971, pp. 151–158.


Dorn, F., Designing Subexponential Algorithms : Problems, Techniques & Structures, thèse de doctorat, Université de Bergen, Norvège, Juillet 2007.


Downey, R. G., M. R. Fellows, *Parameterized complexity*, Springer-Verlag, New York, 1999.


Feige,U., A threshold of ln n for approximation set cover, *Journal of ACM* **45** (1998), p. 634–652.


Fomin, F.V., K. Høie, Pathwidth of cubic graphs and exact algorithms, *Information Processing Letters* **97**, (2006), p. 191–196

Bibliographie II


Fomin, F.V., D. Kratsch, G.J. Woeginger, Exact (exponential) algorithms for the dominating set problem, *Proceedings of WG 2004*, *LNCS* **3353**, (2004), p. 245–256.


Fomin, F.V., F. Grandoni, D. Kratsch, Measure and conquer : Domination - A case study, *Proceedings of ICALP 2005*, *LNCS* **3380**, (2006), p. 192–203.


Fomin, F.V., F. Grandoni, D. Kratsch,
Measure and conquer : A simple $O(2^{0.288n})$ independent set algorithm, *Proc. of SODA 2006*, (2006), pp. 18–25.


Fomin, F.V., F. Grandoni, D. Kratsch, Connected Dominating Set faster than 2^n, *Proceedings of FSTTCS 2006*, *LNCS* **4337**, (2006), p. 152–163.


Garey, M.R., D.S. Johnson,
Computers and Intractability : A Guide to the Theory of NP-Completeness, *Freeman*, 1979.


Gaspers, S., D. Kratsch, M. Liedloff, I. Todinca, Exponential Time Algorithms for the Minimum Dominating Set Problem on Some Graph Classes, article sousmis à *ACM Transactions on Algorithms*.

Bibliographie III


Gaspers, S., M. Liedloff, A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs, *Proceedings of WG 2006*, *LNCS* **4271**, (2006), p. 78–89.


Golovach, P. A., J. Kratochvíl,
Computational complexity of generalized domination : A complete dichotomy for chordal graphs,
Proceedings of WG 2007, LNCS 4769, pp. 1–11, (2007).


Grandoni, F., A note on the complexity of minimum dominating set, *Journal of Discrete Algorithms* **4** (2006), p. 209–214.


Heggernes, P., D. Lokshtanov, Optimal broadcast domination in polynomial time, *Discrete Mathematics* **306** (2006), p. 3267–3280.


Horowitz, E., S. Sahni,
Computing Partitions with Applications to the Knapsack Problem,
Journal of the ACM **21** (1974), pp. 277–292.


Jian, T.,
An $O(2^{0.304n})$ algorithm for solving maximum independent set problem,
IEEE Trans. Computers, **35**, (1986), pp. 847–851.

Bibliographie IV


Kloks, T., Treewidth of Circle Graphs, *International Journal of Foundations of Computer Science* **7** (1996), p. 111–120.


Kratochvíl, J., P.D. Manuel, M. Miller,
Computational complexity of generalized domination : Generalized Domination in Chordal Graphs,
Nordic Journal of Computing **2** (1995), pp. 41–50.


Kratsch, D., M. Liedloff, An Exact Algorithm for the Minimum Dominating Clique Problem, *Theoretical Computer Science*, 2007.


Liedloff, M., Dominating Set on Bipartite Graphs, submit to IPL, Algorithmes exacts et exponentiels pour les problèmes NP-difficiles : domination, variantes et généralisations, PhD Thesis, University of Metz, 2007.


Randerath, B., I. Schiermeyer, Exact algorithms for Minimum Dominating Set, Technical Report zaik-469, Zentrum für Angewandte Informatik, Köln, Germany, April 2004.


Raz, R., S. Safra, A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP, *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, (1997), p. 475–484.

Bibliographie V


Robson, J. M.,
Algorithms for maximum independent sets,
J. Algorithms, **7**, (1986), pp. 425–440.


Robson, J. M.,
Finding a maximum independent set in time $O(2^{n/4})$,
Tech. Rep. 1251-01, LaBRI, Université Bordeaux I, 2001.


Schroeppel, R., A. Shamir,
A $T = O(2^{n/2})$, $S = O(2^{n/2})$ algorithm for certain NP-complete problems,
SIAM Journal on Computing **3** (1981), pp. 456–464.


Shindo, M.,E. Tomita,
Simple algorithms for finding a maximum clique and its worst case time complexity,
Syst. Comput. Jpn, **21**, (1990), pp. 1–13.


R.E. Tarjan, J. M.,A.E. Trojanowski,
Finding a maximum independent set,
SIAM J. Comput., **6**, (1977), pp. 537–546.

Bibliographie VI


Telle, J. A.,
Complexity of domination-type problems in graphs,
Nordic Journal of Computing **1**, (1994), pp. 157–171.


van Rooij, J. M. M., H. L. Bodlaender,
Exact Algorithms for Edge Domination,
Proceedings of IWPEC 2008, à paraître.