Sur les ensembles stables et les bicliques

Serge Gaspers[1] Dieter Kratsch[2]
Mathieu Liedloff[3]

[1]Département d'informatique,
Université de Bergen,
Bergen, Norvège

[2]Laboratoire d'Informatique Théorique et Appliquée
Université Paul Verlaine - Metz
Metz, France

[3]Laboratoire d'Informatique Fondamentale d'Orléans
Université d'Orléans
Orléans, France

Journée Informatique en Région Centre 2008
Orléans, 9 octobre 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.
- Octobre 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.
- Octobre 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.

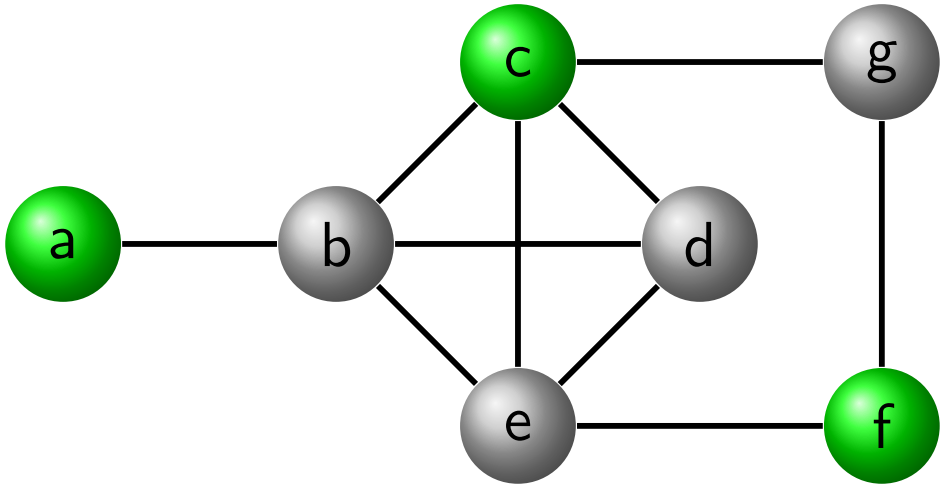
Attaques possibles

Objectif de cet exposé :

Développer des algorithmes exponentiels pour les problèmes de graphes « Ensemble Stable » et « Biclique ».

Max-Ensemble Stable (Max-IS)

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



Motivations

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

Le problème Max-Ensemble Stable :

- pas d'algorithme calculant une solution approchée avec un rapport constant, sauf si P=NP ; [BS92]
- pas d'algorithme à paramètre fixé, sauf si W[1]=FPT. [DF95]

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

Motivations

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

Le problème Max-Ensemble Stable :

- pas d'algorithme calculant une solution approchée avec un rapport constant, sauf si P=NP ; [BS92]
- pas d'algorithme à paramètre fixé, sauf si W[1]=FPT. [DF95]

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

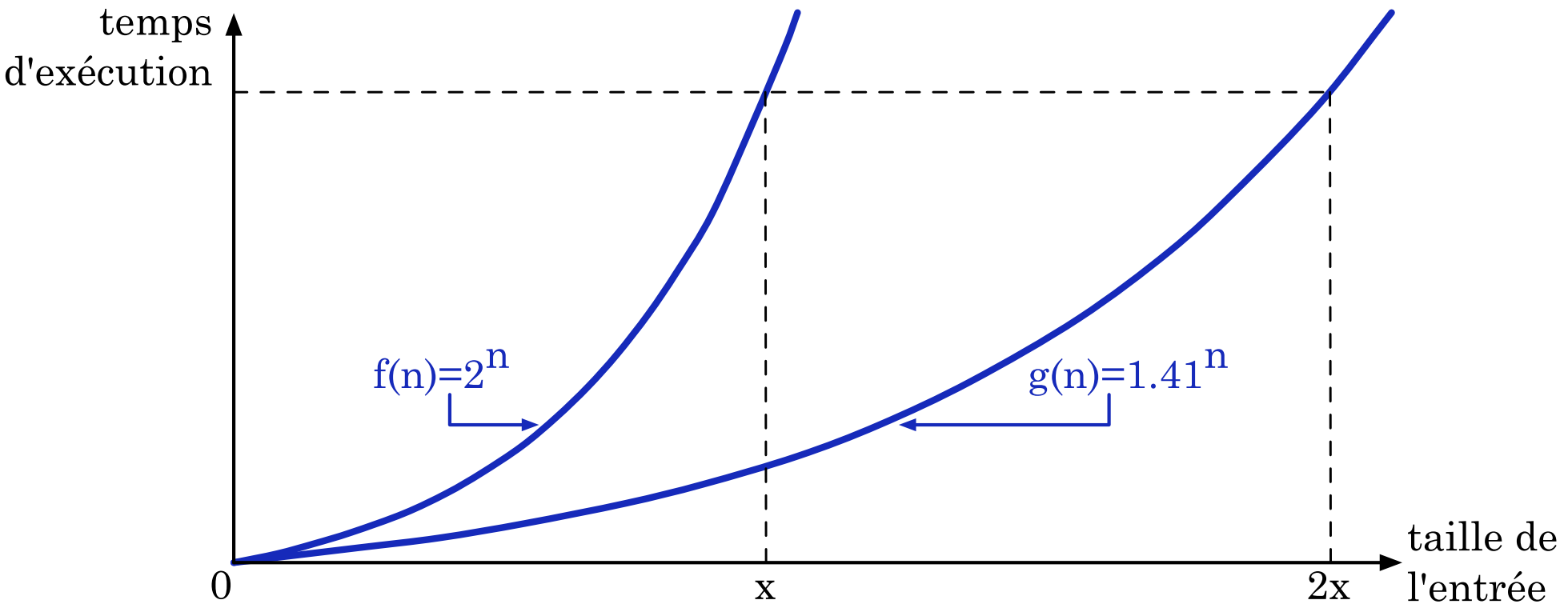
Algorithmes exponentiels

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 Max-Ensemble Stable

Le problème Max-Ensemble Stable est

- NP-complet ;
- W[1]-complet ;
- non approximable avec un facteur constant (seulement log n-approximable, sauf si P=NP).

Construire des algorithmes exponentiels rapides pour le résoudre ?

Max-Ensemble Stable peut être résolu en temps

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

Le problème Max-Ensemble Stable

Le problème Max-Ensemble Stable est

- NP-complet ;
- W[1]-complet ;
- non approximable avec un facteur constant (seulement log n-approximable, sauf si P=NP).

Construire des algorithmes exponentiels rapides pour le résoudre ?

Max-Ensemble Stable peut être résolu en temps

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

Le problème Max-Ensemble Stable

Le problème Max-Ensemble Stable est

- NP-complet ;
- W[1]-complet ;
- non approximable avec un facteur constant (seulement log n-approximable, sauf si P=NP).

Construire des algorithmes exponentiels rapides pour le résoudre ?

Max-Ensemble Stable peut être résolu en temps

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

Le problème Max-Ensemble Stable

Le problème Max-Ensemble Stable est

- NP-complet ;
- W[1]-complet ;
- non approximable avec un facteur constant (seulement log n-approximable, sauf si P=NP).

Il est possible de faire mieux !

Résultats connus pour Max-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] (rapport technique)
  - $O(1.2025^n)$ (espace polynomial)
  - $O(1.1889^n)$ (espace exponentiel)
- 2006 : $O(1.2210^n)$ [Fomin Grandoni Kratsch 06]

Autour du problème Ensemble Stable

2 autres problèmes intéressants :

Enum-Ensemble Stable (Enum-IS)

Lister tous les ensembles stables maximaux d'un graphe.

#-Ensemble Stable (#-IS)

Dénombrer les ensembles stables maximaux d'un graphe.

Résultats connus pour Max-Ensemble Stable

Ensemble Stable

Résultats connus pour Max-Ensemble Stable

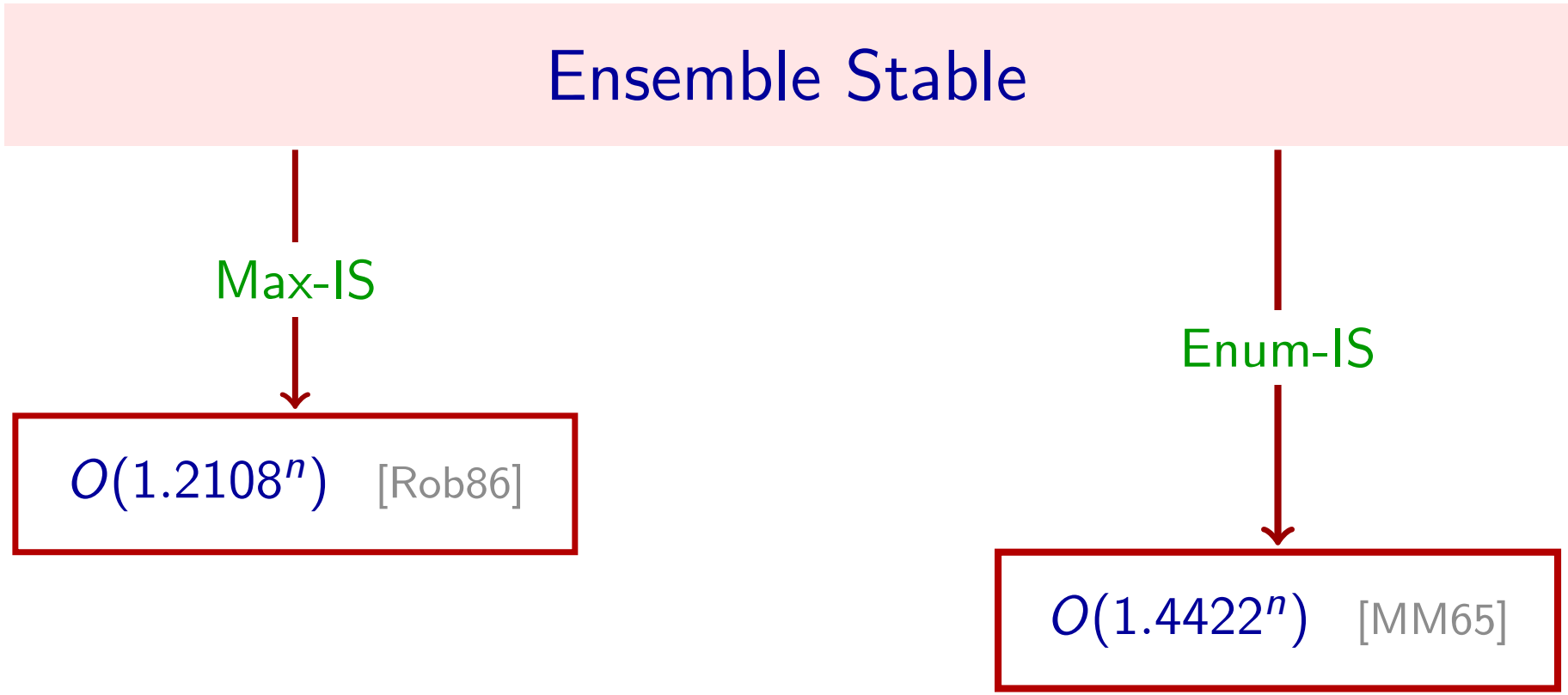
Ensemble Stable

↓ Max-IS

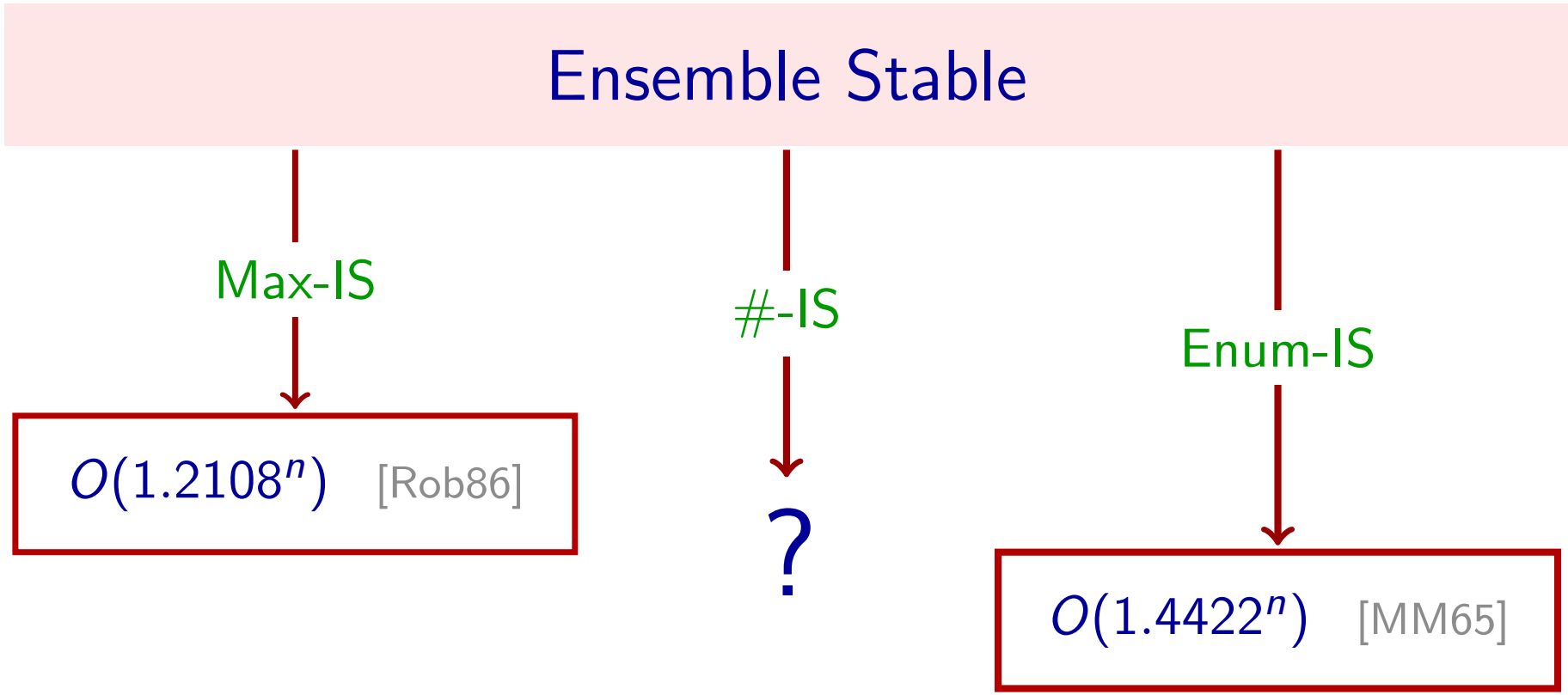
↓

$O(1.2108^n)$ [Rob86]

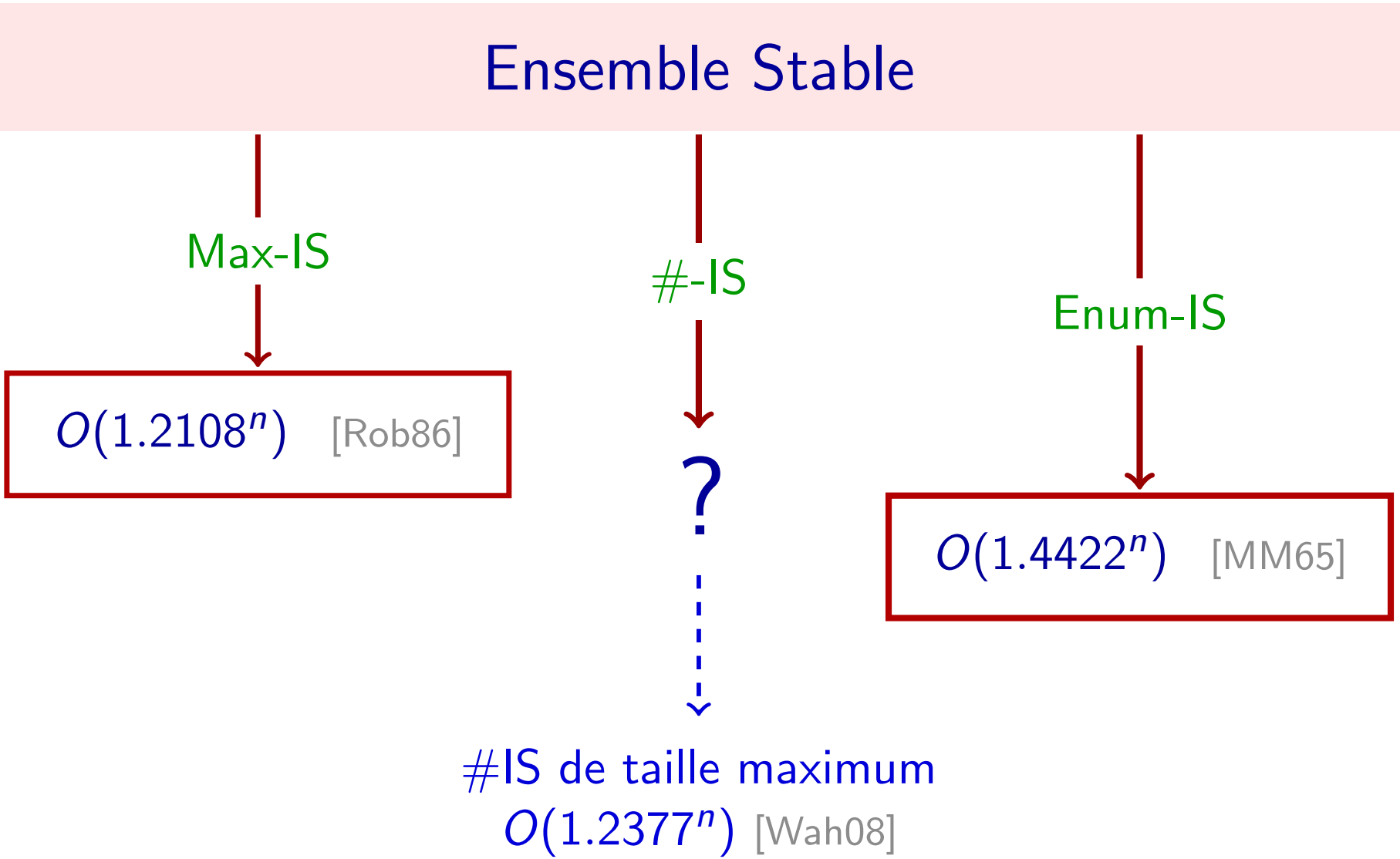
Résultats connus pour Max-Ensemble Stable



Résultats connus pour Max-Ensemble Stable



Résultats connus pour Max-Ensemble Stable



Un algorithme pour dénombrer les IS maximaux

Soit un graphe $G = (V, E)$ et une partition $F \uplus M$ de ses sommets

- F : sommets libres
- M : sommets marqués

Un ensemble $S \subseteq F$ est un Π-stable maximal de $G = (F, M, E)$ si

- S est un stable maximal de $G[F]$
- chaque sommet de M a un voisin dans S (propriété Π)

Un algorithme pour dénombrer les IS maximaux

Soit un graphe $G = (V, E)$ et une partition $F \uplus M$ de ses sommets

- F : sommets libres
- M : sommets marqués

Un ensemble $S \subseteq F$ est un Π-stable maximal de $G = (F, M, E)$ si

- S est un stable maximal de $G[F]$
- chaque sommet de M a un voisin dans S (propriété Π)

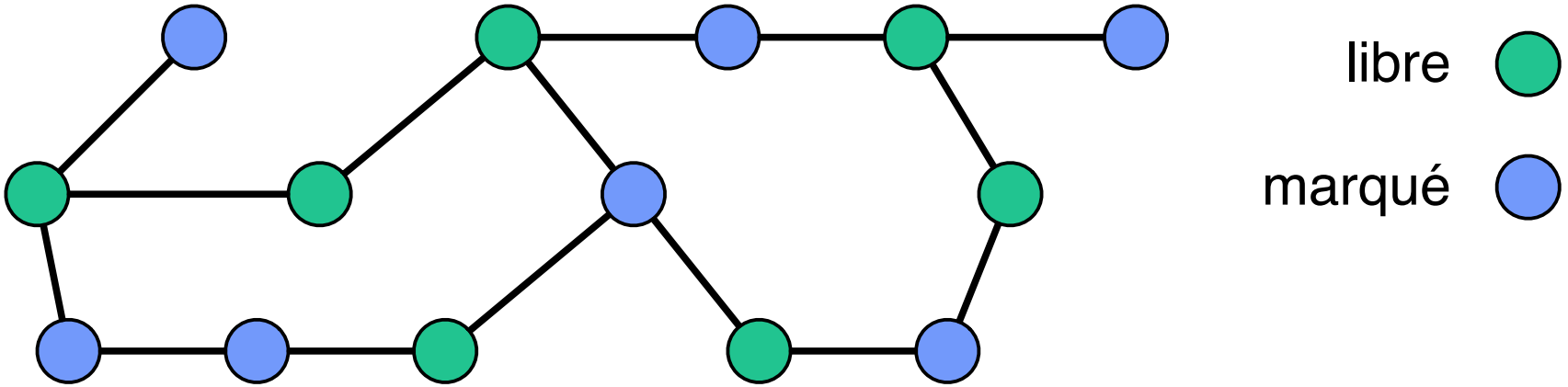
Un algorithme pour dénombrer les IS maximaux

Soit un graphe $G = (V, E)$ et une partition $F \uplus M$ de ses sommets

- F : sommets libres
- M : sommets marqués

Un ensemble $S \subseteq F$ est un Π-stable maximal de $G = (F, M, E)$ si

- S est un stable maximal de $G[F]$
- chaque sommet de M a un voisin dans S (propriété Π)



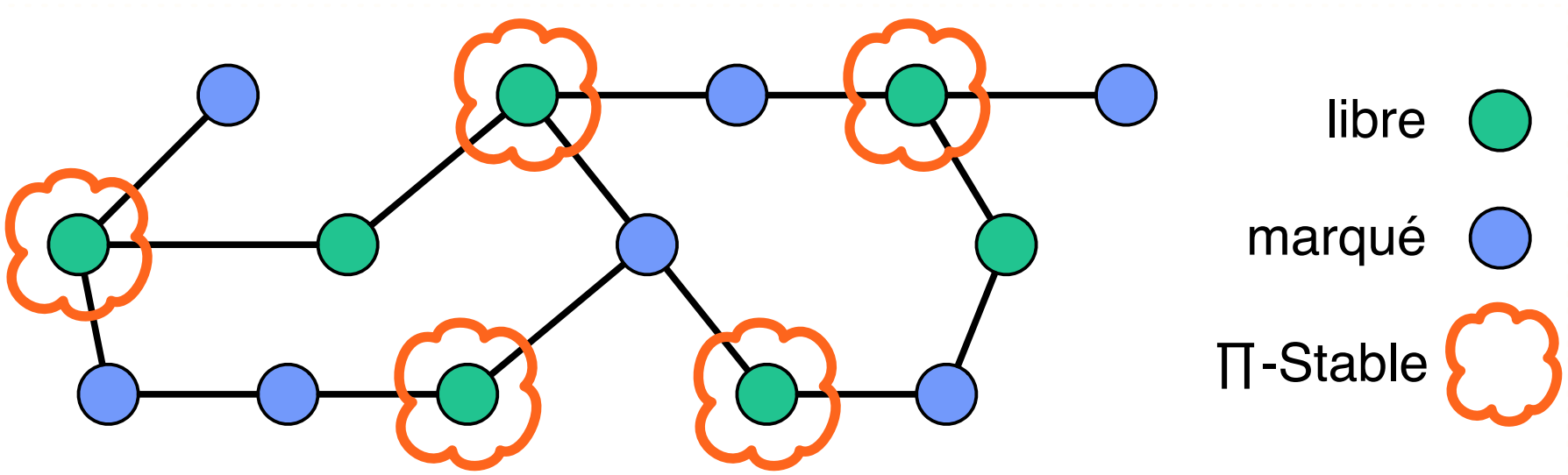
Un algorithme pour dénombrer les IS maximaux

Soit un graphe $G = (V, E)$ et une partition $F \uplus M$ de ses sommets

- F : sommets libres
- M : sommets marqués

Un ensemble $S \subseteq F$ est un Π-stable maximal de $G = (F, M, E)$ si

- S est un stable maximal de $G[F]$
- chaque sommet de M a un voisin dans S (propriété Π)



$\longrightarrow$ notre algorithme dénombre les Π-stables maximaux de $G = (F, M, E)$ en utilisant :

$\longrightarrow$ notre algorithme dénombre les Π-stables maximaux de $G = (F, M, E)$ en utilisant :

- des règles de branchement
- des règles d'arrêt et de réduction

$\longrightarrow$ notre algorithme dénombre les Π-stables maximaux de $G = (F, M, E)$ en utilisant :

- des règles de branchement : résoudre le problème en résolvant récursivement des instances plus petites ;
- des règles d'arrêt et de réduction

$\longrightarrow$ notre algorithme dénombre les Π-stables maximaux de $G = (F, M, E)$ en utilisant :

- des règles de branchement : résoudre le problème en résolvant récursivement des instances plus petites ;
- des règles d'arrêt et de réduction :
  - réduire la taille de l'instance ou la simplifier ;
  - stopper la résolution récursive.

Description de l'algorithme

Les règles d'arrêt :

1. Si G est vide, retourner "1" *($\emptyset$ est le seul stable)*
2. Si $\exists u \in M$ sans voisin libre, retourner "0"

Description de l'algorithme

Les règles d'arrêt :

1. Si G est vide, retourner "1" *($\emptyset$ est le seul stable)*
2. Si $\exists u \in M$ sans voisin libre, retourner "0"

Description de l'algorithme

Les règles de réduction (1/2) :

1. Si $\exists u \in M$ avec un seul voisin libre v,
   "prendre v et résoudre récursivement sur $G - N[v]$"

2. Si $\exists u \in F$ sans voisin libre,
   "prendre u et résoudre récursivement sur $G - N[u]$"

3. "Supprimer toutes les arêtes de $G[M]$"

Description de l'algorithme

Les règles de réduction (1/2) :

1. Si $\exists u \in M$ avec un seul voisin libre v,
   "prendre v et résoudre récursivement sur $G - N[v]$"

2. Si $\exists u \in F$ sans voisin libre,
   "prendre u et résoudre récursivement sur $G - N[u]$"

3. "Supprimer toutes les arêtes de $G[M]$"

Description de l'algorithme

Les règles de réduction (1/2) :

1. Si $\exists u \in M$ avec un seul voisin libre v,
   "prendre v et résoudre récursivement sur $G - N[v]$"

2. Si $\exists u \in F$ sans voisin libre,
   "prendre u et résoudre récursivement sur $G - N[u]$"

3. "Supprimer toutes les arêtes de $G[M]$"

Description de l'algorithme

Les règles de réduction (2/2) :

4. Si $\exists u, v \in F$ t.q. $N[u] = N[v]$,
   "supprimer v et résoudre récursivement sur $G - v$ puis compter 2 fois les stables contenant u"
   (u peut être remplacé par v dans ces stables)

5. Si $\exists u \in M$, $v \in N(u)$ t.q. $N[v] \subseteq N[u]$,
   "supprimer u et résoudre récursivement sur $G - u$"

6. Si $\exists u, v \in M$ t.q. $N(u) = N(v)$,
   "supprimer u et résoudre récursivement sur $G - u$"

7. Si $\exists u \in F \cup M$, $v \in F$ t.q. $N(u) = N(v)$,
   "supprimer v et résoudre récursivement sur $G - v$"
   (v appartient seulement aux stables contenant u)

Description de l'algorithme

Les règles de réduction (2/2) :

4. Si $\exists u, v \in F$ t.q. $N[u] = N[v]$,
   "supprimer v et résoudre récursivement sur $G - v$ puis compter 2 fois les stables contenant u"
   (u peut être remplacé par v dans ces stables)

5. Si $\exists u \in M$, $v \in N(u)$ t.q. $N[v] \subseteq N[u]$,
   "supprimer u et résoudre récursivement sur $G - u$"

6. Si $\exists u, v \in M$ t.q. $N(u) = N(v)$,
   "supprimer u et résoudre récursivement sur $G - u$"

7. Si $\exists u \in F \cup M$, $v \in F$ t.q. $N(u) = N(v)$,
   "supprimer v et résoudre récursivement sur $G - v$"
   (v appartient seulement aux stables contenant u)

Description de l'algorithme

Les règles de réduction (2/2) :

4. Si $\exists u, v \in F$ t.q. $N[u] = N[v]$,
   "supprimer v et résoudre récursivement sur $G - v$ puis compter 2 fois les stables contenant u"
   (u peut être remplacé par v dans ces stables)

5. Si $\exists u \in M$, $v \in N(u)$ t.q. $N[v] \subseteq N[u]$,
   "supprimer u et résoudre récursivement sur $G - u$"

6. Si $\exists u, v \in M$ t.q. $N(u) = N(v)$,
   "supprimer u et résoudre récursivement sur $G - u$"

7. Si $\exists u \in F \cup M$, $v \in F$ t.q. $N(u) = N(v)$,
   "supprimer v et résoudre récursivement sur $G - v$"
   (v appartient seulement aux stables contenant u)

Description de l'algorithme

Les règles de réduction (2/2) :

4. Si $\exists u, v \in F$ t.q. $N[u] = N[v]$,
   "supprimer v et résoudre récursivement sur $G - v$ puis compter 2 fois les stables contenant u"
   (u peut être remplacé par v dans ces stables)

5. Si $\exists u \in M$, $v \in N(u)$ t.q. $N[v] \subseteq N[u]$,
   "supprimer u et résoudre récursivement sur $G - u$"

6. Si $\exists u, v \in M$ t.q. $N(u) = N(v)$,
   "supprimer u et résoudre récursivement sur $G - u$"

7. Si $\exists u \in F \cup M$, $v \in F$ t.q. $N(u) = N(v)$,
   "supprimer v et résoudre récursivement sur $G - v$"
   (v appartient seulement aux stables contenant u)

Description de l'algorithme

La règle de branchement :

1. S'il existe $u \in M$ avec 2 voisins libres, choisir u
2. Sinon, choisir $u \in F \cup M$ t.q.
   - u a le plus petit degré possible
   - u a un voisin de degré maximum
   - la somme des degrés des voisins de u est la plus grande possible

Construire la liste des voisins libres $\mathrm{BL}(u) = [v_1, \ldots, v_k]$ t.q.
- v_1 a le moins de voisins possible dans $V \setminus N(u)$
- ajouter les sommets de $N(u) \cap N(v_1)$ à $\mathrm{BL}(u)$
- ajouter les sommets de $N(u) \setminus N[v_1]$ à $\mathrm{BL}(u)$ par nombre croissant de voisins dans $V \setminus N(u)$

Résolution récursive :

- Si $u \in F$, "Dénombrer récursivement les stables contenant u"
- Pour chaque sommet $v_i \in \mathrm{BL}(u)$, "Mettre $v_1, \ldots, v_{i-1}$ dans M puis dénombrer récursivement les stables contenant v_i"

Description de l'algorithme

La règle de branchement :

1. S'il existe $u \in M$ avec 2 voisins libres, choisir u
2. Sinon, choisir $u \in F \cup M$ t.q.
   - u a le plus petit degré possible
   - u a un voisin de degré maximum
   - la somme des degrés des voisins de u est la plus grande possible

   Construire la liste des voisins libres $\mathrm{BL}(u) = [v_1, \ldots, v_k]$ t.q.
   - v_1 a le moins de voisins possible dans $V \setminus N(u)$
   - ajouter les sommets de $N(u) \cap N(v_1)$ à $\mathrm{BL}(u)$
   - ajouter les sommets de $N(u) \setminus N[v_1]$ à $\mathrm{BL}(u)$ par nombre croissant de voisins dans $V \setminus N(u)$

Résolution récursive :

- Si $u \in F$, "Dénombrer récursivement les stables contenant u"
- Pour chaque sommet $v_i \in \mathrm{BL}(u)$, "Mettre $v_1, \ldots, v_{i-1}$ dans M puis dénombrer récursivement les stables contenant v_i"

Description de l'algorithme

La règle de branchement :

1. S'il existe $u \in M$ avec 2 voisins libres, choisir u
2. Sinon, choisir $u \in F \cup M$ t.q.
   - u a le plus petit degré possible
   - u a un voisin de degré maximum
   - la somme des degrés des voisins de u est la plus grande possible

   Construire la liste des voisins libres $\mathtt{BL}(u) = [v_1, \ldots, v_k]$ t.q.
   - v_1 a le moins de voisins possible dans $V \setminus N(u)$
   - ajouter les sommets de $N(u) \cap N(v_1)$ à $\mathtt{BL}(u)$
   - ajouter les sommets de $N(u) \setminus N[v_1]$ à $\mathtt{BL}(u)$ par nombre croissant de voisins dans $V \setminus N(u)$

Résolution récursive :

- Si $u \in F$, "Dénombrer récursivement les stables contenant u"
- Pour chaque sommet $v_i \in \mathtt{BL}(u)$, "Mettre $v_1, \ldots, v_{i-1}$ dans M puis dénombrer récursivement les stables contenant v_i"

Description de l'algorithme

La règle de branchement :

1. S'il existe $u \in M$ avec 2 voisins libres, choisir u
2. Sinon, choisir $u \in F \cup M$ t.q.
   - u a le plus petit degré possible
   - u a un voisin de degré maximum
   - la somme des degrés des voisins de u est la plus grande possible

   Construire la liste des voisins libres $\mathtt{BL}(u) = [v_1, \ldots, v_k]$ t.q.
   - v_1 a le moins de voisins possible dans $V \setminus N(u)$
   - ajouter les sommets de $N(u) \cap N(v_1)$ à $\mathtt{BL}(u)$
   - ajouter les sommets de $N(u) \setminus N[v_1]$ à $\mathtt{BL}(u)$ par nombre croissant de voisins dans $V \setminus N(u)$

Résolution récursive :

- Si $u \in F$, "Dénombrer récursivement les stables contenant u"
- Pour chaque sommet $v_i \in \mathtt{BL}(u)$, "Mettre $v_1, \ldots, v_{i-1}$ dans M puis dénombrer récursivement les stables contenant v_i"

Analyse du temps d'exécution

L'analyse du temps d'exécution demande :

- de considérer l'arbre de recherche généré par l'algorithme ;
- d'établir une borne supérieure sur le nombre de sous-problèmes récursivement résolus, c-à-d le nombre de nœuds dans l'arbre de recherche.

Analyse du temps d'exécution

L'analyse du temps d'exécution demande :

- de considérer l'arbre de recherche généré par l'algorithme ;
- d'établir une borne supérieure sur le nombre de sous-problèmes récursivement résolus, c-à-d le nombre de nœuds dans l'arbre de recherche.

Analyse du temps d'exécution

Borner le nombre de nœuds dans l'arbre de recherche :

1. définir une mesure $\mu(\mathcal{I})$ sur la taille de l'instance $\mathcal{I}$;
2. borner le progrès fait par l'algorithme à chaque branchement ;
3. calculer les récurrences pour toutes les règles de branchement ;
4. résoudre ces récurrences et prendre la plus grande solution.

Pour l'analyse de notre algorithme :

$$\mu(G) = w_1 V_1 + w_2 V_2 + w_3 V_3 + w_{\geq 4} V_{\geq 4}$$

où :

- V_i est le nombre de sommets de degré i ;
- $w_1, w_2, w_3, w_{\geq 4} \in [0, 1]$.

Analyse du temps d'exécution

Borner le nombre de nœuds dans l'arbre de recherche :

1. définir une mesure $\mu(\mathcal{I})$ sur la taille de l'instance $\mathcal{I}$;
2. borner le progrès fait par l'algorithme à chaque branchement ;
3. calculer les récurrences pour toutes les règles de branchement ;
4. résoudre ces récurrences et prendre la plus grande solution.

Pour l'analyse de notre algorithme :

$$\mu(G) = w_1 V_1 + w_2 V_2 + w_3 V_3 + w_{\geq 4} V_{\geq 4}$$

où :

- V_i est le nombre de sommets de degré i ;
- $w_1, w_2, w_3, w_{\geq 4} \in [0, 1]$.

Analyse du temps d'exécution

Borner le nombre de nœuds dans l'arbre de recherche :

1. définir une mesure $\mu(\mathcal{I})$ sur la taille de l'instance $\mathcal{I}$;
2. borner le progrès fait par l'algorithme à chaque branchement ;
3. calculer les récurrences pour toutes les règles de branchement ;
4. résoudre ces récurrences et prendre la plus grande solution.

Pour l'analyse de notre algorithme :

$$\mu(G) = w_1 V_1 + w_2 V_2 + w_3 V_3 + w_{\geq 4} V_{\geq 4}$$

où :

- V_i est le nombre de sommets de degré i ;
- $w_1, w_2, w_3, w_{\geq 4} \in [0, 1]$.

Analyse du temps d'exécution

Borner le nombre de nœuds dans l'arbre de recherche :

1. définir une mesure $\mu(\mathcal{I})$ sur la taille de l'instance $\mathcal{I}$;
2. borner le progrès fait par l'algorithme à chaque branchement ;
3. calculer les récurrences pour toutes les règles de branchement ;
4. résoudre ces récurrences et prendre la plus grande solution.

Pour l'analyse de notre algorithme :

$$\mu(G) = w_1 V_1 + w_2 V_2 + w_3 V_3 + w_{\geq 4} V_{\geq 4}$$

où :

- V_i est le nombre de sommets de degré i ;
- $w_1, w_2, w_3, w_{\geq 4} \in [0, 1]$.

Analyse du temps d'exécution

Borner le nombre de nœuds dans l'arbre de recherche :

1. définir une mesure $\mu(\mathcal{I})$ sur la taille de l'instance $\mathcal{I}$;
2. borner le progrès fait par l'algorithme à chaque branchement ;
3. calculer les récurrences pour toutes les règles de branchement ;
4. résoudre ces récurrences et prendre la plus grande solution.

Pour l'analyse de notre algorithme :

$$\mu(G) = w_1 V_1 + w_2 V_2 + w_3 V_3 + w_{\geq 4} V_{\geq 4}$$

où :

- V_i est le nombre de sommets de degré i ;
- $w_1, w_2, w_3, w_{\geq 4} \in [0, 1]$.

Analyse du temps d'exécution

Borner le nombre de nœuds dans l'arbre de recherche :

1. définir une mesure $\mu(\mathcal{I})$ sur la taille de l'instance $\mathcal{I}$;
2. borner le progrès fait par l'algorithme à chaque branchement ;
3. calculer les récurrences pour toutes les règles de branchement ;
4. résoudre ces récurrences et prendre la plus grande solution.

Pour l'analyse de notre algorithme :

$$\mu(G) = w_1 V_1 + w_2 V_2 + w_3 V_3 + w_{\geq 4} V_{\geq 4}$$

où :

- V_i est le nombre de sommets de degré i ;
- $w_1, w_2, w_3, w_{\geq 4} \in [0, 1]$.

Analyse du temps d'exécution

Si on note $T(\mu)$ le temps d'exécution sur un graphe de taille μ, l'analyse de la règle de branchement donne la récurrence générale :

$$T(\mu) \leq T(\mu - \Delta_u) + \sum_{v_i \in \mathtt{BL}(u)} T(\mu - \Delta_{\mathtt{Op}(v_i)})$$

où

- $\Delta_u = w_{d(u)} + \sum_{v \in N(u)} w_{d(v)} + \sum_{x \in N^2(u)} (w_{d(x)} - w_{d(x)-d_{N(u)}(x)})$
- $\Delta_{\mathtt{Op}(v_i)} = w_{d(v_i)} + \sum_{x \in N(v_i)} w_{d(x)} + \sum_{y \in N^2(v_i)} (w_{d(y)} - w_{d(y)-d_{N(v_i)}(y)}) + \mathsf{marked}_1(\mathtt{Op}(v_i))$

$\longrightarrow$ Reste à instancier les récurrences, à les résoudre et à déterminer les poids de la mesure μ. ($\approx$ 30 récurrences)

Analyse du temps d'exécution

Si on note $T(\mu)$ le temps d'exécution sur un graphe de taille μ, l'analyse de la règle de branchement donne la récurrence générale :

$$T(\mu) \leq T(\mu - \Delta_u) + \sum_{v_i \in \mathtt{BL}(u)} T(\mu - \Delta_{\mathtt{Op}(v_i)})$$

où

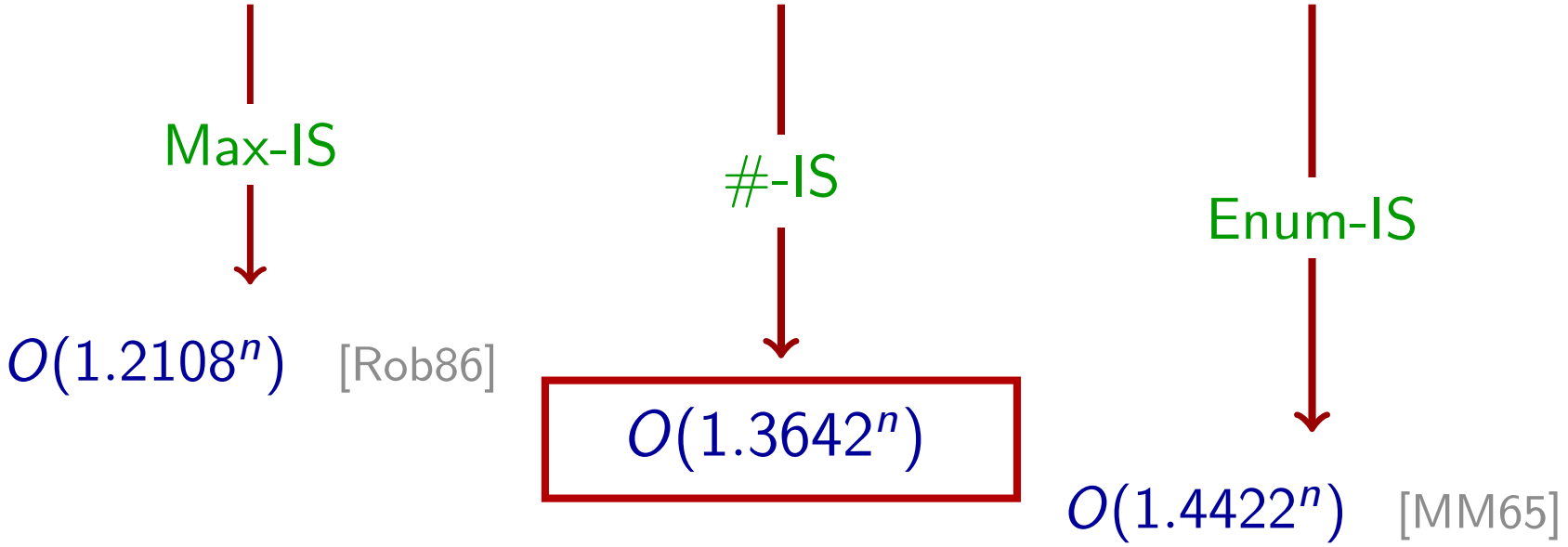
- $\Delta_u = w_{d(u)} + \sum_{v \in N(u)} w_{d(v)} + \sum_{x \in N^2(u)} (w_{d(x)} - w_{d(x)-d_{N(u)}(x)})$
- $\Delta_{\mathtt{Op}(v_i)} = w_{d(v_i)} + \sum_{x \in N(v_i)} w_{d(x)} + \sum_{y \in N^2(v_i)} (w_{d(y)} - w_{d(y)-d_{N(v_i)}(y)}) + \mathsf{marked}_1(\mathtt{Op}(v_i))$

$\longrightarrow$ Reste à instancier les récurrences, à les résoudre et à déterminer les poids de la mesure μ. ($\approx$ 30 récurrences)

Analyse du temps d'exécution

Théorème (borne supérieure)

L'algorithme dénombre les ensembles stables maximaux d'un graphe en temps $O(1.3642^n)$.



Analyse du temps d'exécution

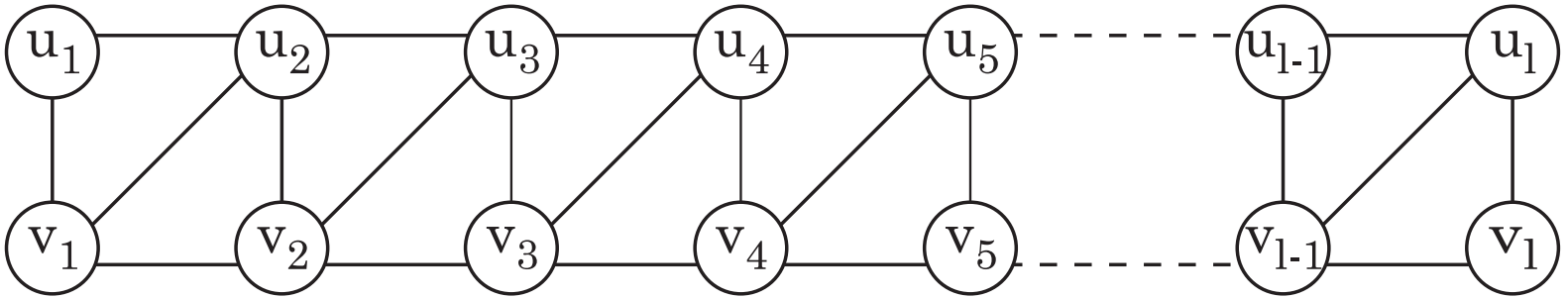
$\longrightarrow$ Avec cette technique d'analyse, la borne sup est surestimée ...

Analyse du temps d'exécution

$\longrightarrow$ Avec cette technique d'analyse, la borne sup est surestimée ...

Théorème (borne inférieure)

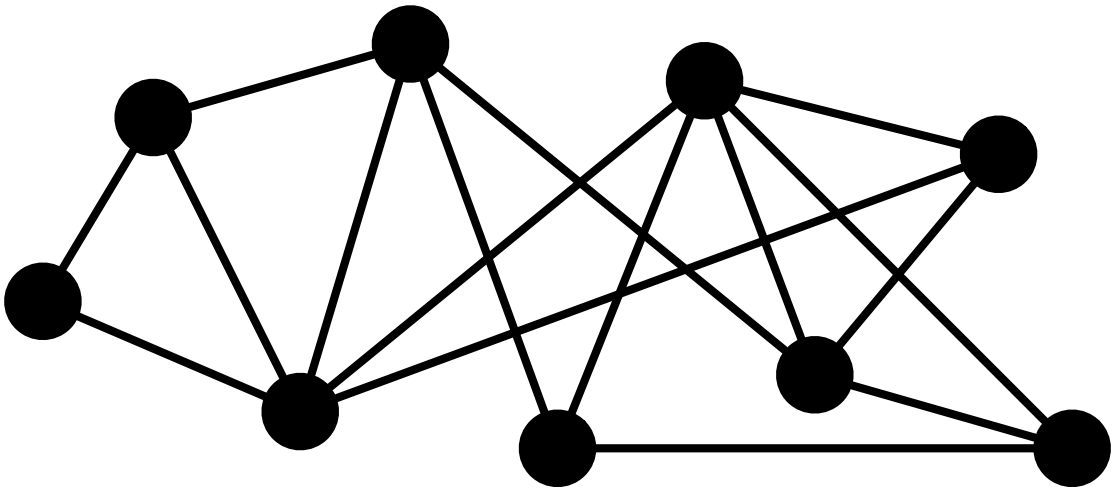
L'algorithme dénombre les ensembles stables maximaux d'un graphe en temps $\Omega(1.3247^n)$.



$$T(n) = T(n-3) + T(n-4) + T(n-5)$$

Définition

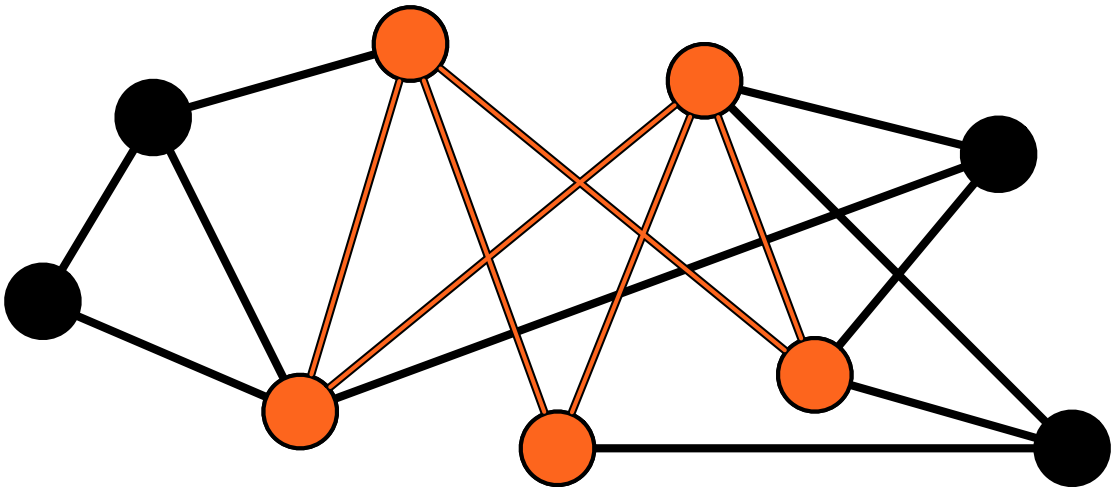
Biclique : $B \subseteq V$ est une biclique ssi $G[B]$ est biparti complet



Biclique maximale : B n'est contenu dans aucune autre biclique

Définition

Biclique : $B \subseteq V$ est une biclique ssi $G[B]$ est biparti complet



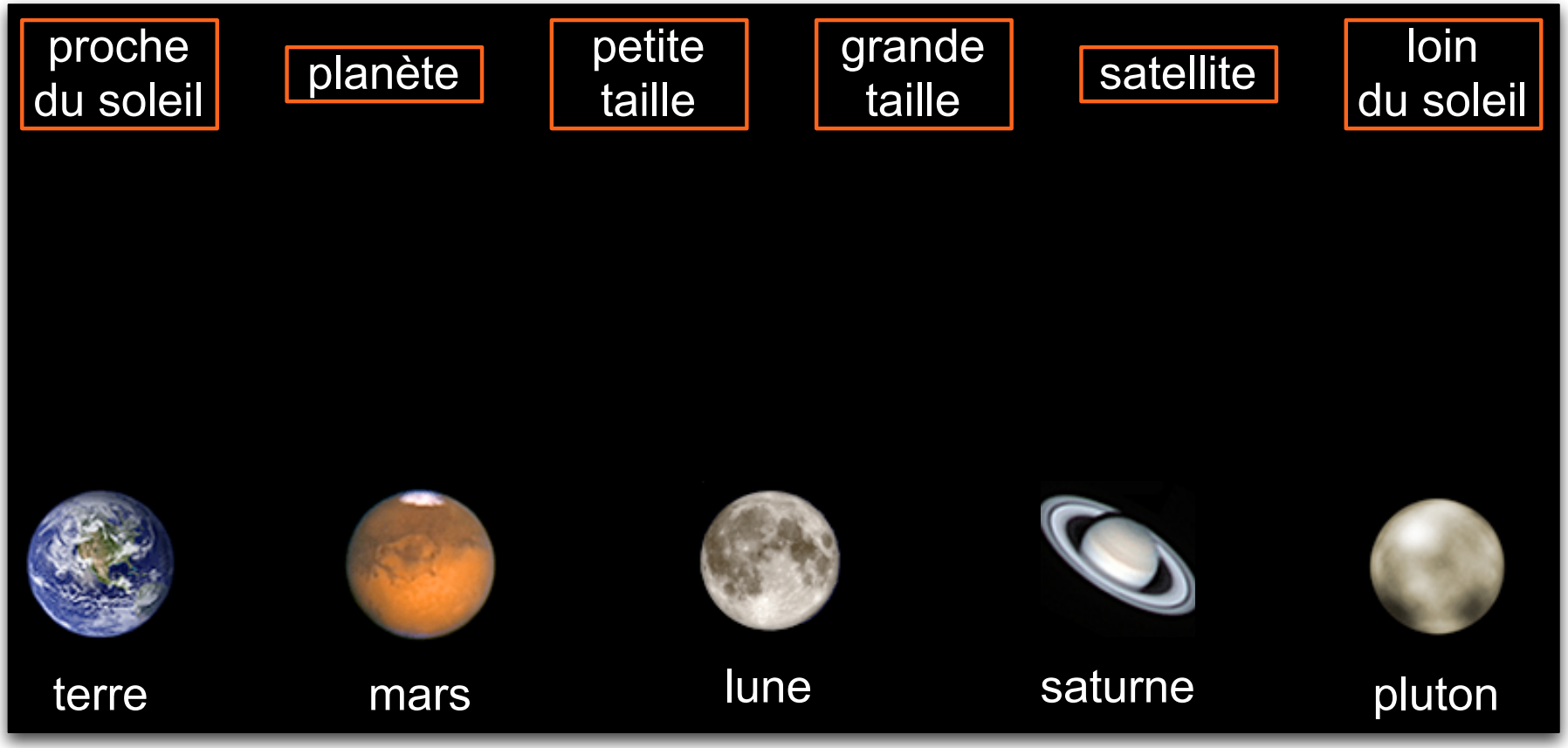
Biclique maximale : B n'est contenu dans aucune autre biclique

Applications

- automates
- intelligence artificielle
- biologie
- fouille de données (analyse formelle de concepts)

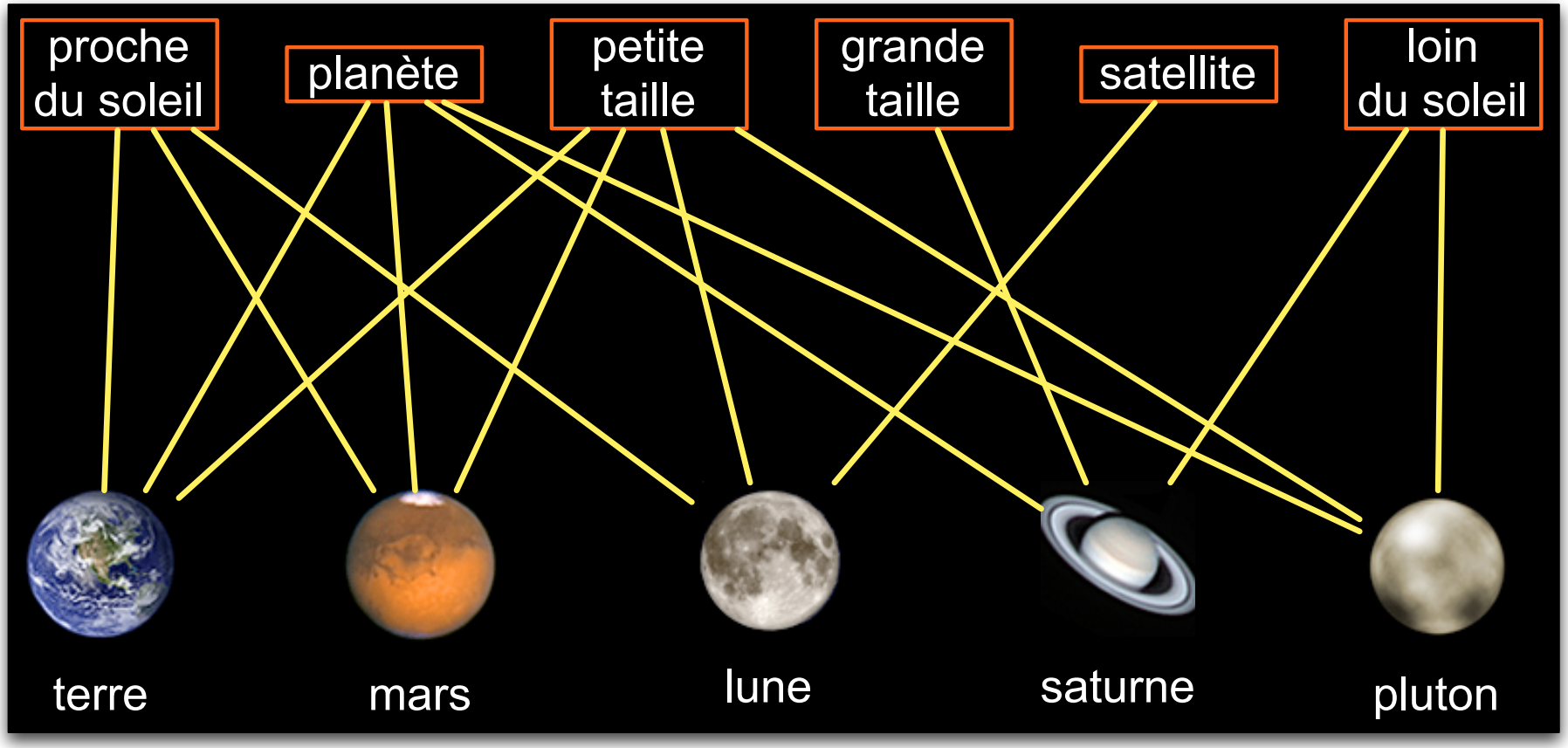
Applications

- automates
- intelligence artificielle
- biologie
- fouille de données (analyse formelle de concepts)



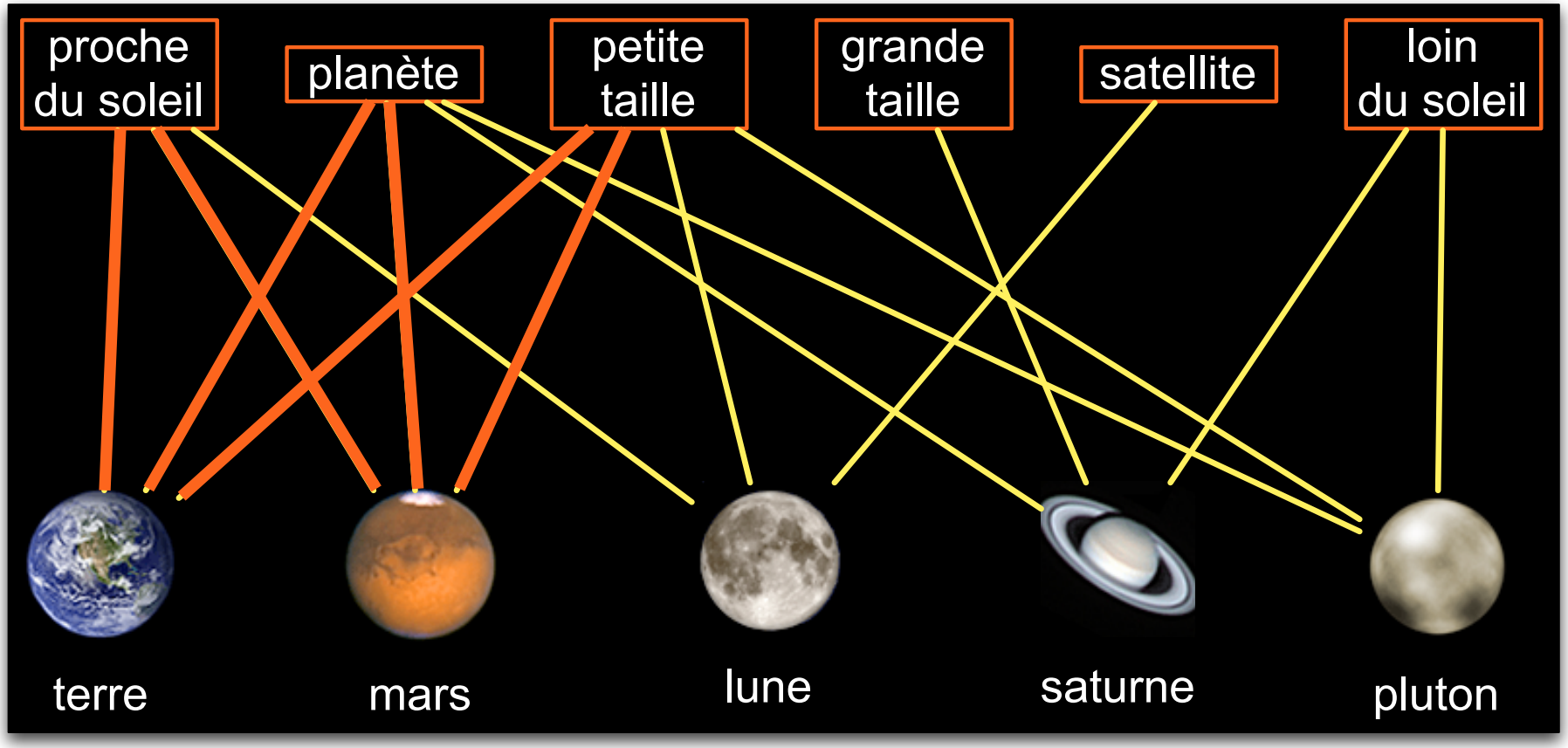
Applications

- automates
- intelligence artificielle
- biologie
- fouille de données (analyse formelle de concepts)



Applications

- automates
- intelligence artificielle
- biologie
- fouille de données (analyse formelle de concepts)

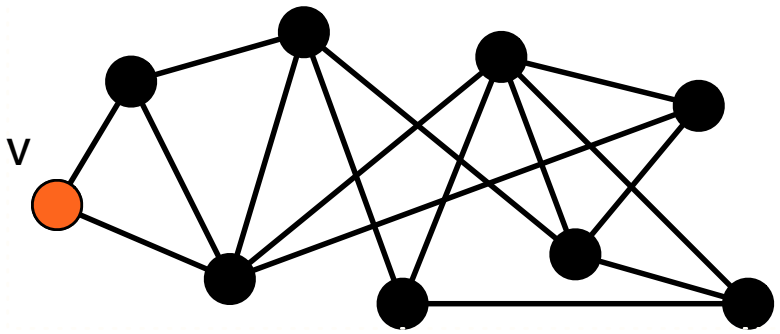


Réductions polynomiales

Définition de H_v

Soit $G = (V, E)$. Etant donné $v \in V$, on définit H_v par $V(H_v) = N(v) \cup N^2(v)$ et $uv \in E(H_v)$ ssi

- $uv \in E$ et $u, v \in N(v)$
- $uv \in E$ et $u, v \in N^2(v)$
- $uv \notin E$, $u \in N(v)$ et $v \in N^2(v)$

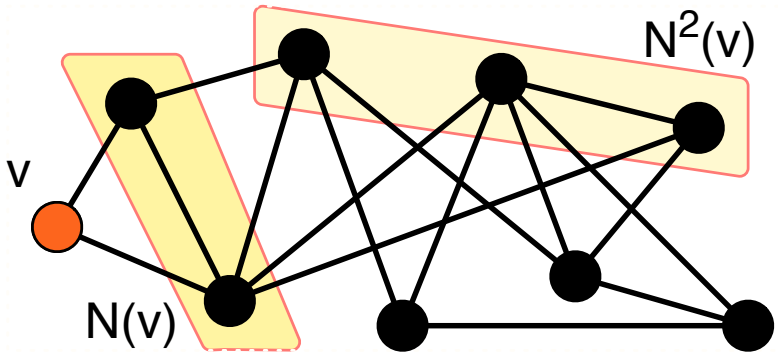


Réductions polynomiales

Définition de H_v

Soit $G = (V, E)$. Etant donné $v \in V$, on définit H_v par $V(H_v) = N(v) \cup N^2(v)$ et $uv \in E(H_v)$ ssi

- $uv \in E$ et $u, v \in N(v)$
- $uv \in E$ et $u, v \in N^2(v)$
- $uv \notin E$, $u \in N(v)$ et $v \in N^2(v)$

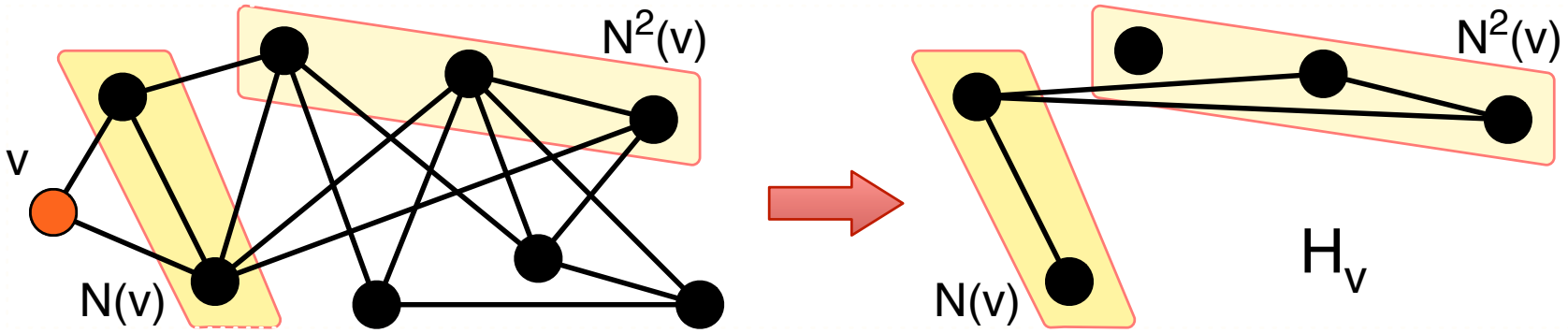


Réductions polynomiales

Définition de H_v

Soit $G = (V, E)$. Etant donné $v \in V$, on définit H_v par $V(H_v) = N(v) \cup N^2(v)$ et $uv \in E(H_v)$ ssi

- $uv \in E$ et $u, v \in N(v)$
- $uv \in E$ et $u, v \in N^2(v)$
- $uv \notin E$, $u \in N(v)$ et $v \in N^2(v)$

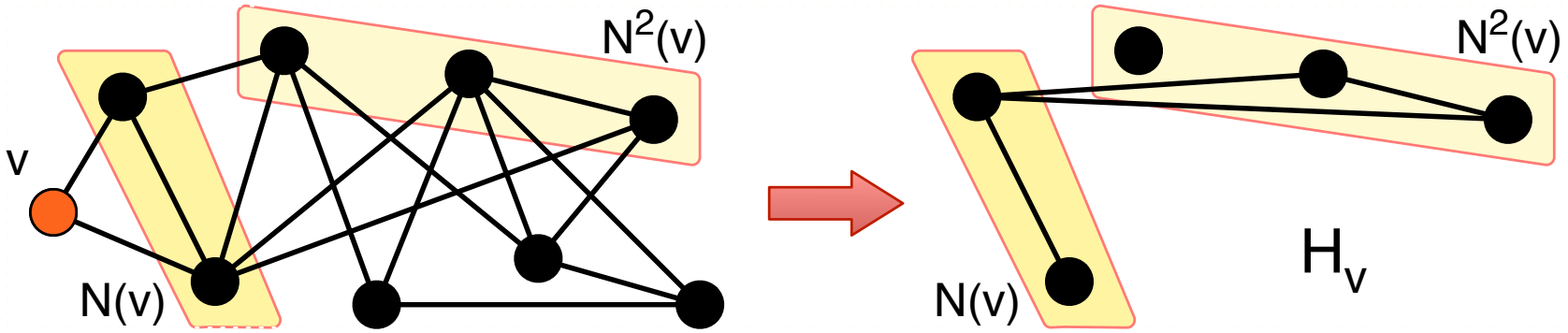


Réductions polynomiales

Définition de H_v

Soit $G = (V, E)$. Etant donné $v \in V$, on définit H_v par $V(H_v) = N(v) \cup N^2(v)$ et $uv \in E(H_v)$ ssi

- $uv \in E$ et $u, v \in N(v)$
- $uv \in E$ et $u, v \in N^2(v)$
- $uv \notin E$, $u \in N(v)$ et $v \in N^2(v)$



Théorème (bijection entre stables et bicliques)

$B \subseteq V$ est une biclique (maximale) de G ssi $B - v$ est un stable (maximal) de H_v pour un certain sommet $v \in B$.

Nombre de bicliques maximales

Théorème (nombre de bicliques maximales) [Prisner 2000] :

Le nombre de bicliques maximales est au plus $1.618034^n \cdot n^{5/2}$ et est au moins de $3^{n/3} \approx 1.4422^n$.

Théorème (nombre de bicliques maximales)

Le nombre de bicliques maximales est au plus $n \cdot 3^{n/3}$.

Preuve

- Pour tout sommet $v \in V$, il y a une bijection entre les bicliques maximales contenant v et les ensembles stables maximaux de H_v.
- Le nombre maximum de stables maximaux est $3^{n/3}$ [MM65].

Nombre de bicliques maximales

Théorème (nombre de bicliques maximales) [Prisner 2000] :

Le nombre de bicliques maximales est au plus $1.618034^n \cdot n^{5/2}$ et est au moins de $3^{n/3} \approx 1.4422^n$.

Théorème (nombre de bicliques maximales)

Le nombre de bicliques maximales est au plus $n \cdot 3^{n/3}$.

Preuve

- Pour tout sommet $v \in V$, il y a une bijection entre les bicliques maximales contenant v et les ensembles stables maximaux de H_v.
- Le nombre maximum de stables maximaux est $3^{n/3}$ [MM65].

Conséquences

Théorème (biclique maximum) :

Etant donné un algorithme calculant un stable maximum en temps $O^*(c^n)$, alors il existe un algorithme calculant une biclique maximum en temps $O^*(c^n)$.

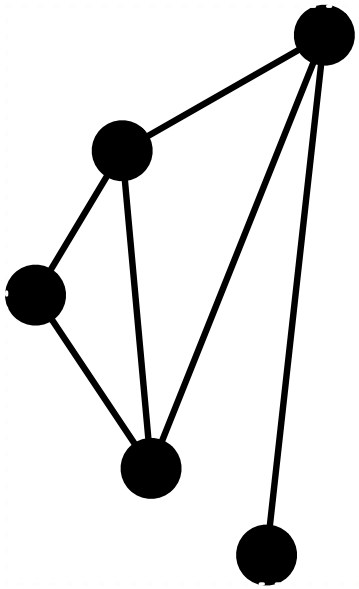
$\longrightarrow$ **Difficulté :** pour dénombrer les bicliques maximales, il faut éviter de les compter plusieurs fois.

(La construction précédente ne le permet pas.)

Dénombrer les bicliques maximales

Soit $G = (V, E)$. Soit $G' = (V', E')$ une copie de G.
On construit $G'' = (V'', E'')$ avec

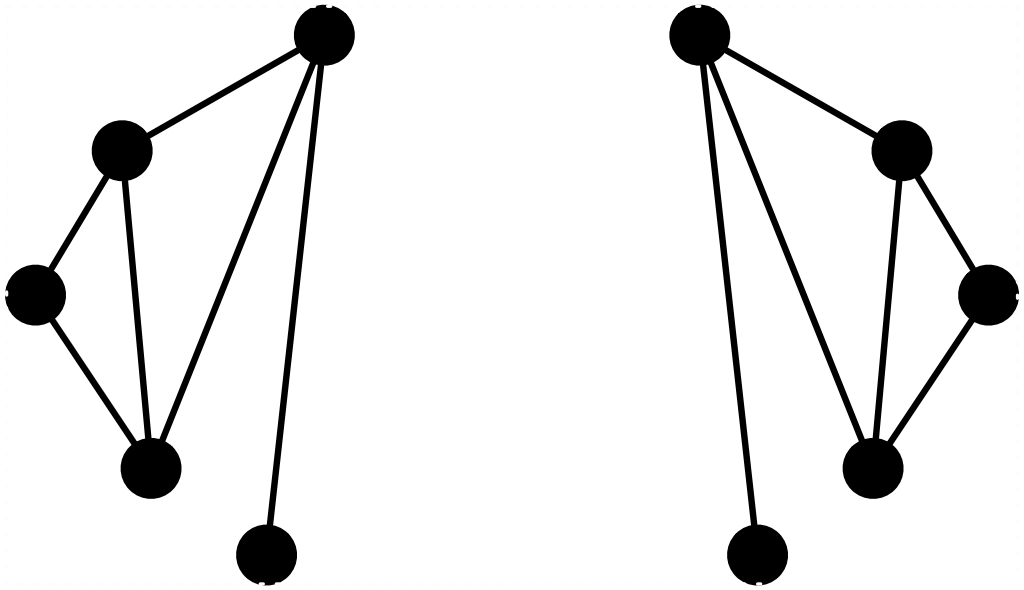
- $V'' = V \cup V$
- $E'' = E \cup E' \cup \{uv' : u = v \text{ ou } uv \notin E\}$



Dénombrer les bicliques maximales

Soit $G = (V, E)$. Soit $G' = (V', E')$ une copie de G.
On construit $G'' = (V'', E'')$ avec

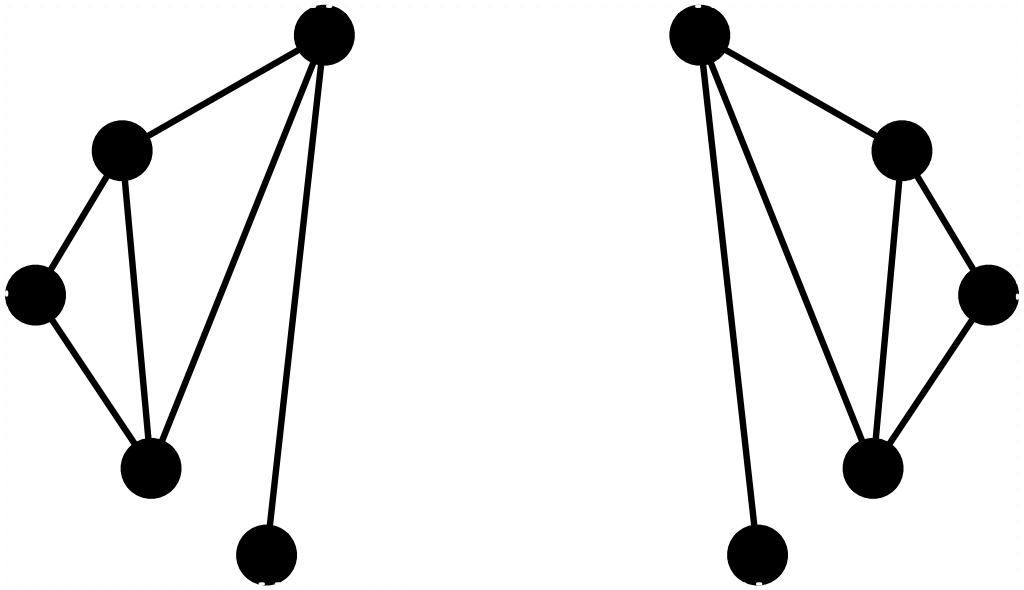
- $V'' = V \cup V$
- $E'' = E \cup E' \cup \{uv' : u = v \text{ ou } uv \notin E\}$



Dénombrer les bicliques maximales

Soit $G = (V, E)$. Soit $G' = (V', E')$ une copie de G.
On construit $G'' = (V'', E'')$ avec

- $V'' = V \cup V$
- $E'' = E \cup E' \cup \{uv' : u = v \text{ ou } uv \notin E\}$

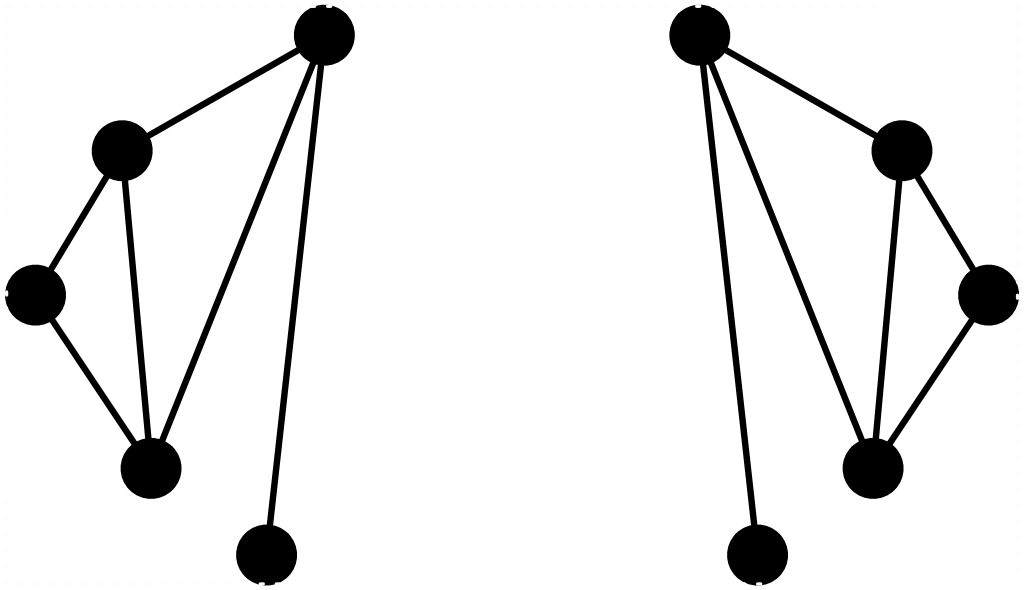


$\longrightarrow$ G'' a $2n$ sommets

Dénombrer les bicliques maximales

Soit $G = (V, E)$. Soit $G' = (V', E')$ une copie de G.
On construit $G'' = (V'', E'')$ avec

- $V'' = V \cup V$
- $E'' = E \cup E' \cup \{uv' : u = v \text{ ou } uv \notin E\}$

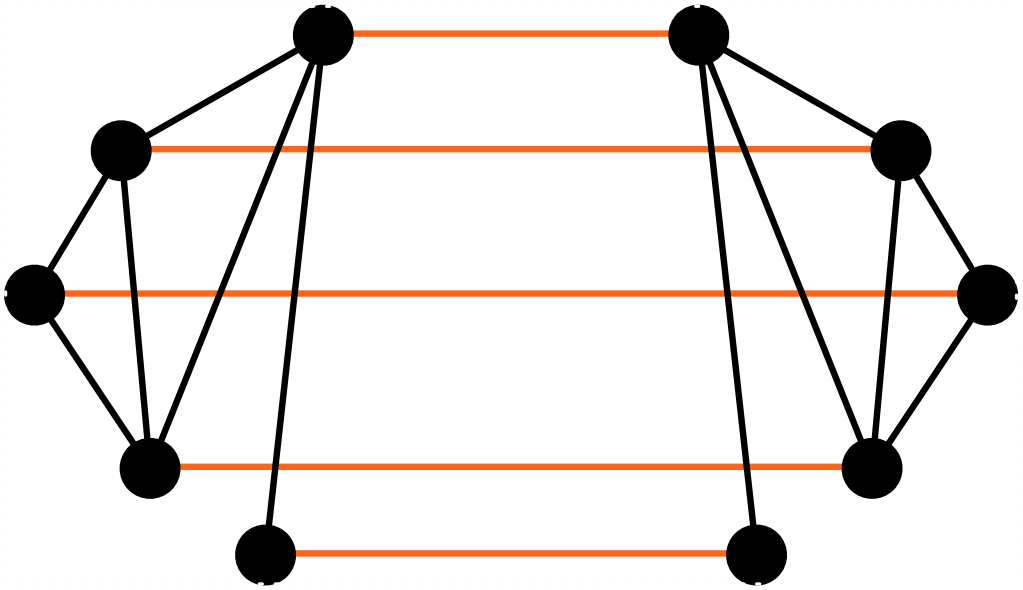


$\longrightarrow$ G'' a $2n$ sommets

Dénombrer les bicliques maximales

Soit $G = (V, E)$. Soit $G' = (V', E')$ une copie de G.
On construit $G'' = (V'', E'')$ avec

- $V'' = V \cup V$
- $E'' = E \cup E' \cup \{uv' : u = v \text{ ou } uv \notin E\}$

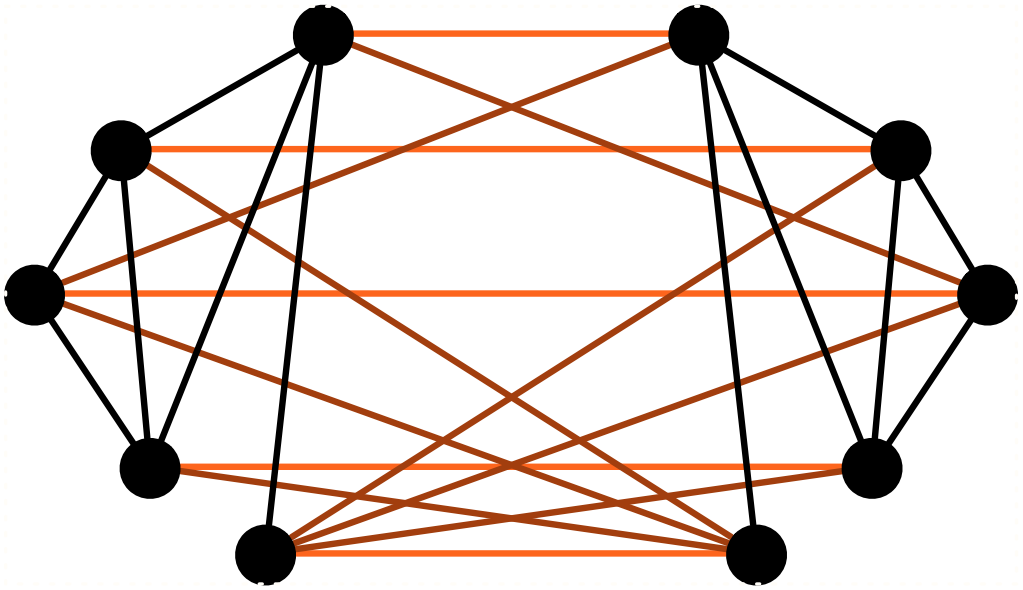


$\longrightarrow$ G'' a $2n$ sommets

Dénombrer les bicliques maximales

Soit $G = (V, E)$. Soit $G' = (V', E')$ une copie de G.
On construit $G'' = (V'', E'')$ avec

- $V'' = V \cup V$
- $E'' = E \cup E' \cup \{uv' : u = v \text{ ou } uv \notin E\}$

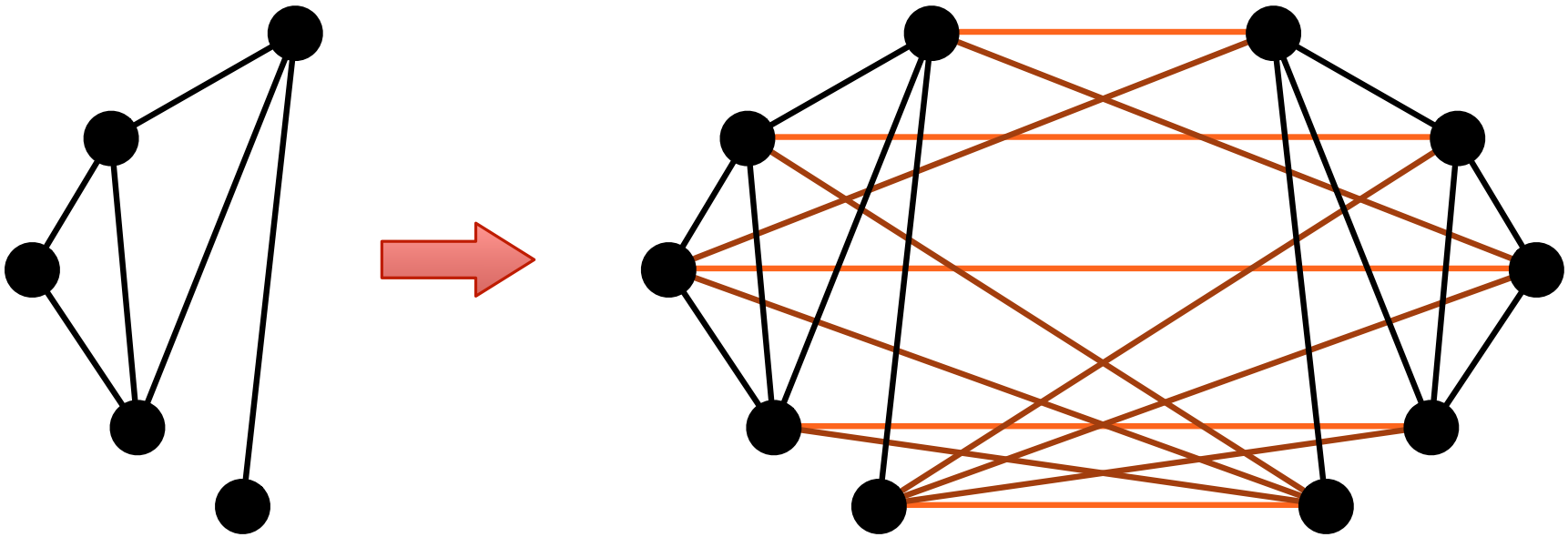


$\longrightarrow$ G'' a $2n$ sommets

Dénombrer les bicliques maximales

Théorème

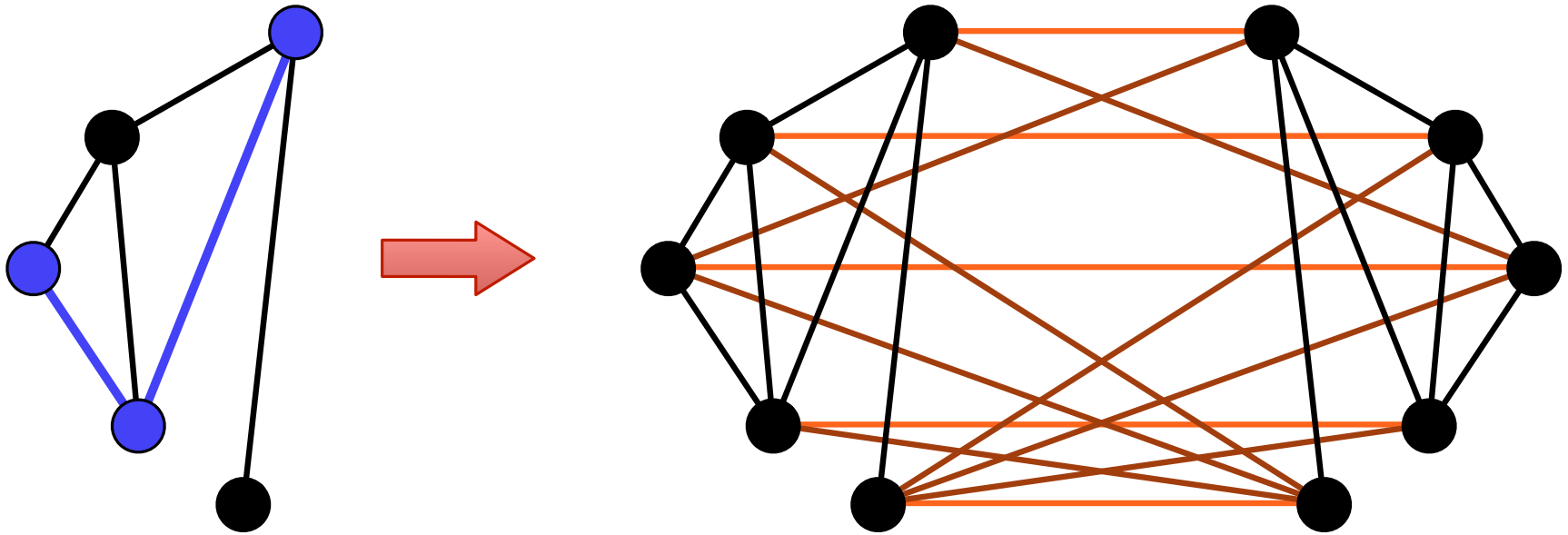
Le nombre d'ensembles stables maximaux de G'' est égal à deux fois le nombre de bicliques maximales de G.



Dénombrer les bicliques maximales

Théorème

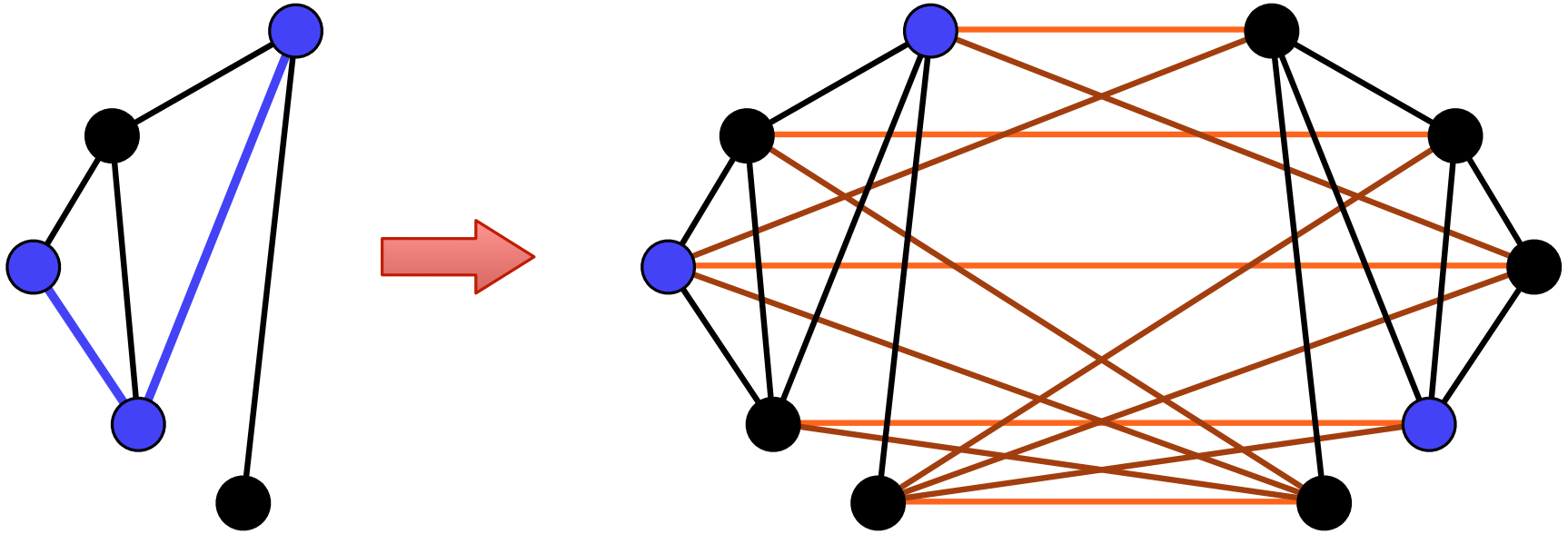
Le nombre d'ensembles stables maximaux de G'' est égal à deux fois le nombre de bicliques maximales de G.



Dénombrer les bicliques maximales

Théorème

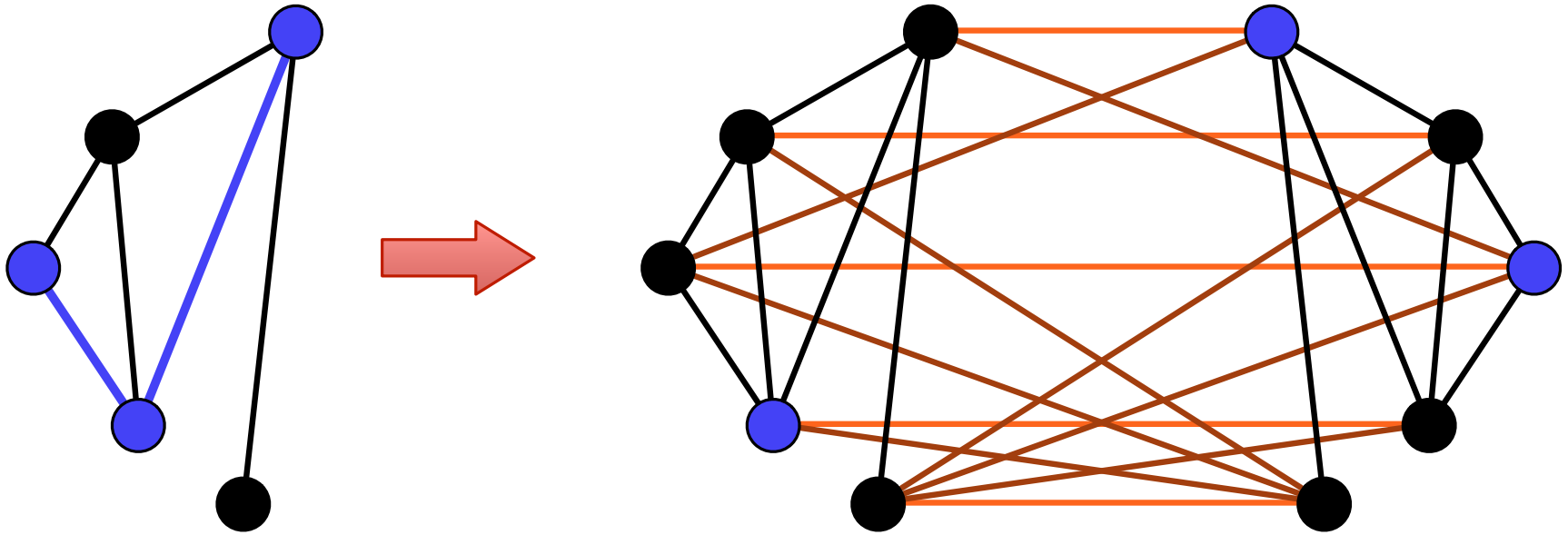
Le nombre d'ensembles stables maximaux de G'' est égal à deux fois le nombre de bicliques maximales de G.



Dénombrer les bicliques maximales

Théorème

Le nombre d'ensembles stables maximaux de G'' est égal à deux fois le nombre de bicliques maximales de G.



Dénombrer les bicliques maximales

Problème :

- Notre algorithme compte les stables maximaux en $O(1.3642^n)$.
- Le graphe G'' a $2n$ sommets.
- Peut-on dénombrer les bicliques maximales en $O(1.3642^n)$?

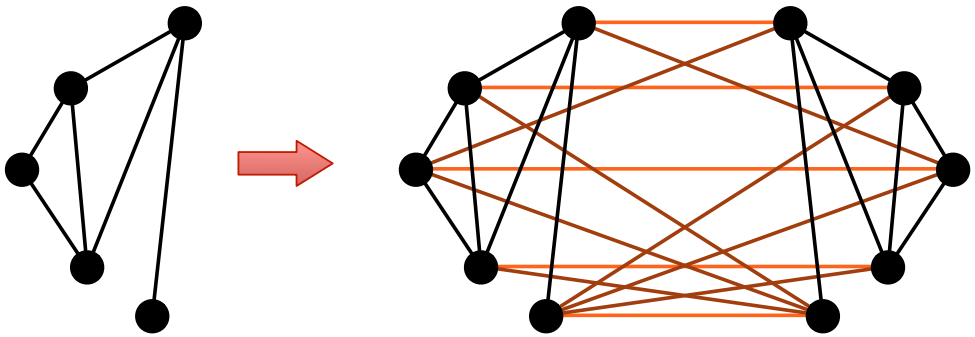
Dénombrer les bicliques maximales

Problème :

- Notre algorithme compte les stables maximaux en $O(1.3642^n)$.
- Le graphe G'' a $2n$ sommets.
- Peut-on dénombrer les bicliques maximales en $O(1.3642^n)$?

Réponse :

- Initialement le graphe G'' a des sommets de degré n.
- On exécute n fois l'algo et, à chaque appel, n sommets sont supprimés. (De plus des sommets sont marqués pour éviter de double-compter.)



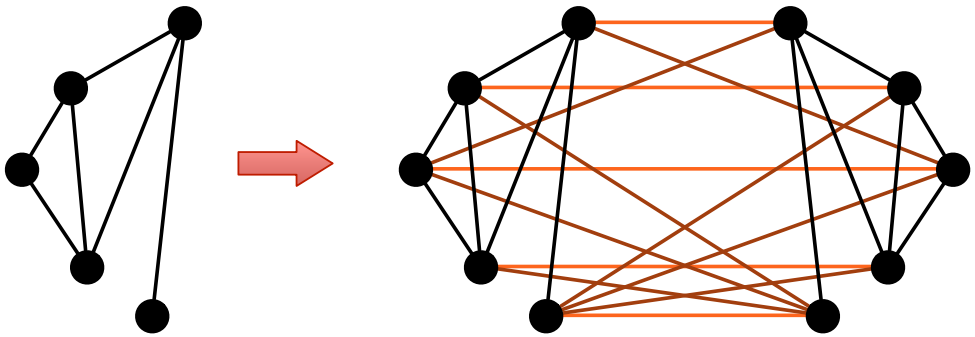
Dénombrer les bicliques maximales

Problème :

- Notre algorithme compte les stables maximaux en $O(1.3642^n)$.
- Le graphe G'' a $2n$ sommets.
- Peut-on dénombrer les bicliques maximales en $O(1.3642^n)$?

Réponse :

- Initialement le graphe G'' a des sommets de degré n.
- On exécute n fois l'algo et, à chaque appel, n sommets sont supprimés. (De plus des sommets sont marqués pour éviter de double-compter.)



Théorème (dénombrer bicliques maximales)

On peut dénombrer les bicliques maximales en temps $O(1.3642^n)$.

Conclusion

Conclusion - Questions

Résultats obtenus :

- Dénombrer les ensembles stables maximaux et les bicliques maximales en temps $O(1.3642^n)$.
- Le nombre de bicliques maximales est au plus $n \cdot 3^{n/3}$ (et au moins $n + 3^{n/3}$).

Questions intéressantes :

- Algorithmique :
  Obtenir un algorithme de dénombrement plus rapide.
- Combinatoire :
  Quelle est la vraie borne inférieure / supérieure pour le nombre de bicliques maximales ?

Merci pour votre attention.