Implanter les algorithmes Oum-Seymour et Oum

J.-F Raymond, B.-M Bui-Xuan et P. Trébuchet
jeanflorent.raymond@ens-lyon.fr

LIP6, Université Pierre et Marie Curie

17/11/2011

Sommaire

Motivations

But : la résolution des problèmes difficiles en théorie des graphes.

Idée : Décomposer l'instance du problème en sous-instances peut permettre une résolution plus rapide.

Complexité : $\mathcal{O}(f(k) \cdot poly(n))$

Références

Plusieurs *largeurs* existent sur les graphes :

- largeur arborescente (*tree*-*width*) ou de branche (*branch*-*width*) : Robertson et Seymour (1991) ;
- largeur de clique (*clique*-*width*) : Courcelle, Engelfriet et Rozenberg (1993) ;
- largeur modulaire (*module*-*width*) : Lanlignel (2001), Rao (2008) ;
- largeur de rang (*rank*-*width*) : Oum and Seymour (2006) ;
- largeur booléenne (*bool*-*width*) : Bui-Xuan, Telle et Vatshelle (2009) ;
- etc.

Références

Quelques résultats sur la largeur de rang :

- Oum et Seymour 2006, Oum 2008 : algorithme de décomposition qui à un graphe quelconque à n sommets associe une 3-approximation de la décomposition optimale en temps $\mathcal{O}(opt^2 \cdot n^4 \cdot 2^{3 \cdot opt})$, où opt est la largeur de la décomposition optimale ;
- Courcelle 2000 : à partir d'une décomposition d'un graphe, on peut résoudre une grande quantité de problèmes en temps exponentiel où le facteur exponentiel dépend de la largeur de la décomposition.

Sommaire

Largeur de rang

Soient $A \subset V(G)$ et M la matrice d'adjacence de G.
On définit le rang de coupe par

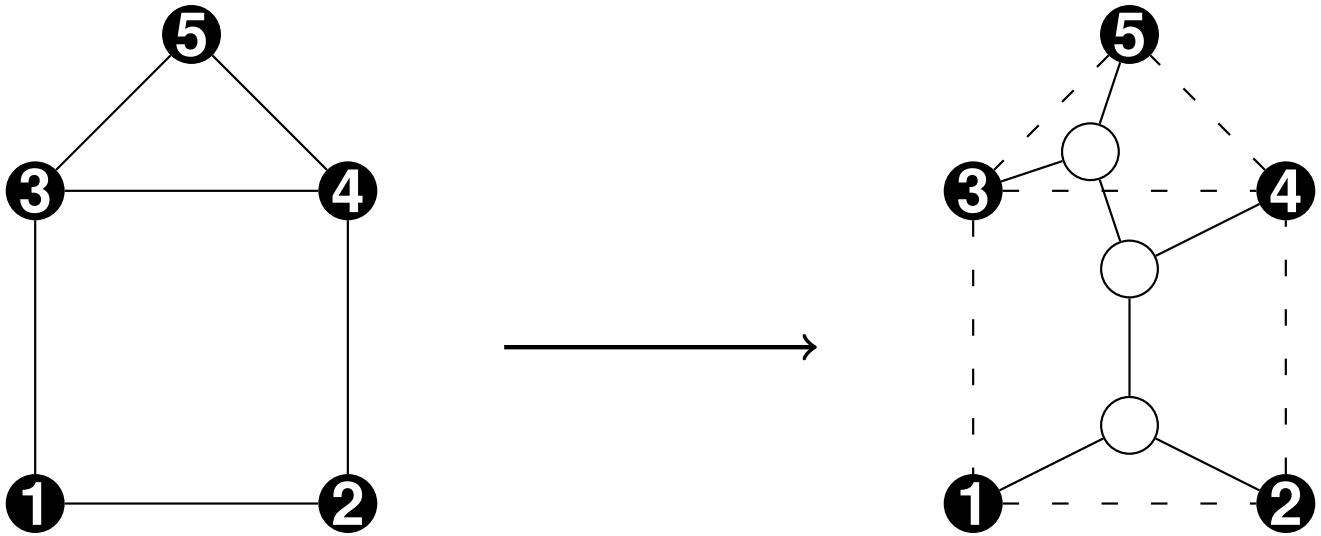
$$\text{cut-rank}(A) = \text{rg}(M_{(A, V(G)\setminus A)})$$

On remarque que :

- le rang de coupe est symétrique ;
- le rang de coupe est une fonction sous-modulaire.

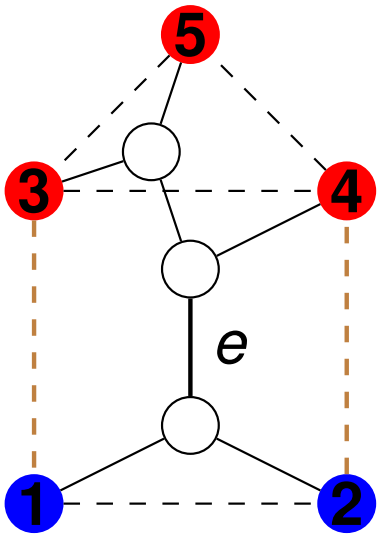
Décomposition

Arbre de décomposition $\mathcal{T}$ d'un graphe G : arbre binaire entier non enraciné dont les feuilles sont en bijection avec les sommets du graphe.



Largeur de rang

Choisir une arête e de $\mathcal{T}$: découper $V(G)$ en V_1 et V_2 ;
Largeur de e : rang de la matrice d'adjacence dans G entre V_1 et V_2 ;
Largeur de $\mathcal{T}$: maximum des largeurs des arêtes de $\mathcal{T}$



$$\text{cut-rank}(e) = \text{rg}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = 2$$

L'algorithme de Oum-Seymour

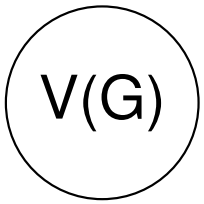
Entrée : Un graphe connexe G et un entier k.
Sortie :

- soit un message d'erreur ;
- soit une décomposition arborescente de largeur de rang inférieure à k.

Principe : On affine petit à petit une décomposition arborescente partielle.

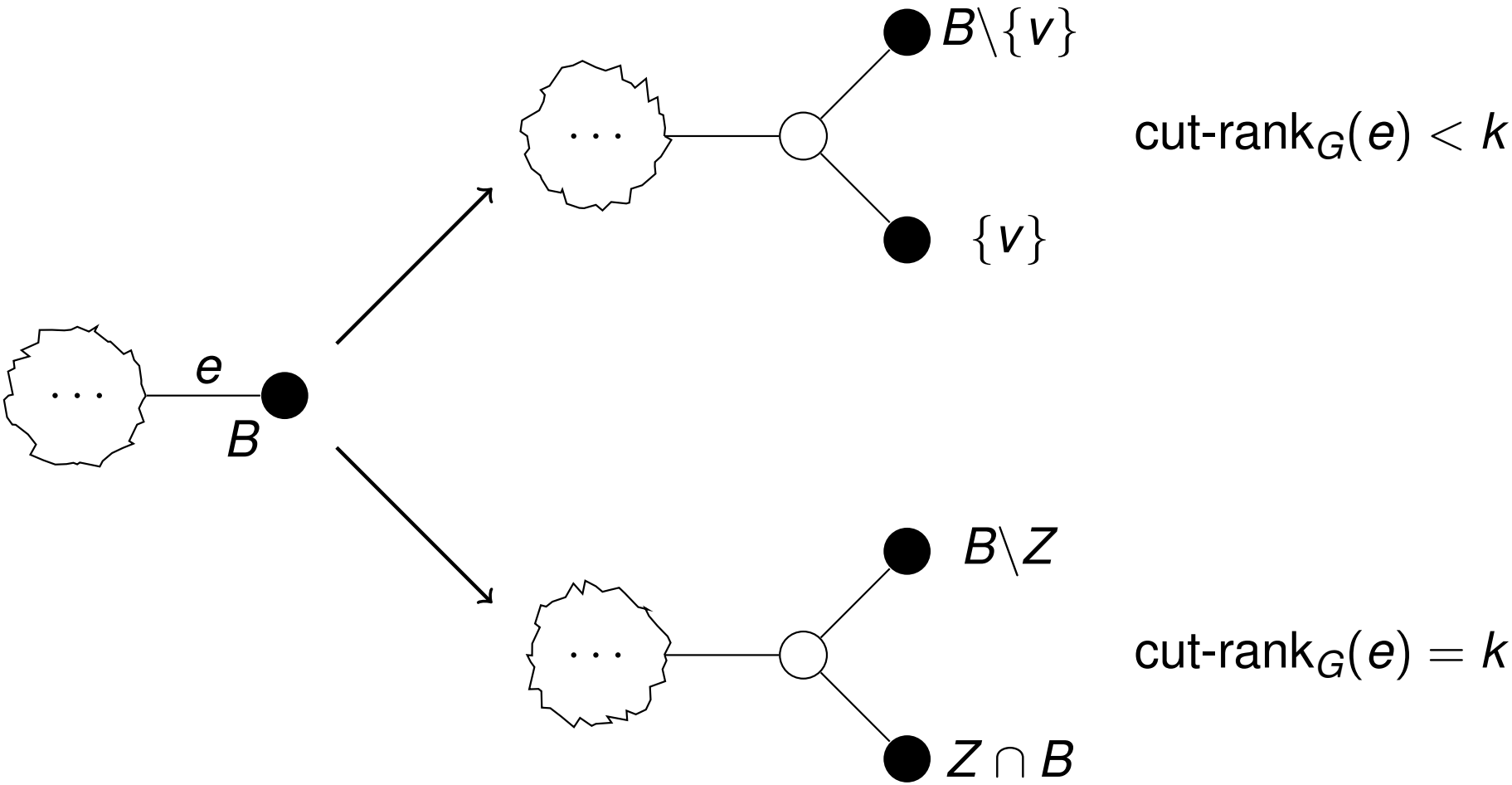
L'algorithme de Oum-Seymour

État initial : l'unique feuille de T est associée aux sommets de G :



L'algorithme de Oum-Seymour

Ensuite, si $|B| > 1$



L'algorithme de Oum-Seymour

Avec Z qui découpe B en deux parties de cut-rank inférieur à celui de B :

- $Z \cap B \neq \emptyset$ et cut-rank$_G(Z \cap B) \leq$ cut-rank$_G(B)$;
- $B \backslash Z \neq \emptyset$ et cut-rank$_G(Z \backslash B) \leq$ cut-rank$_G(B)$.

Si un tel Z n'existe pas, on renvoit un message d'erreur.

Mais comment calculer cet ensemble ?

Calcul de Z

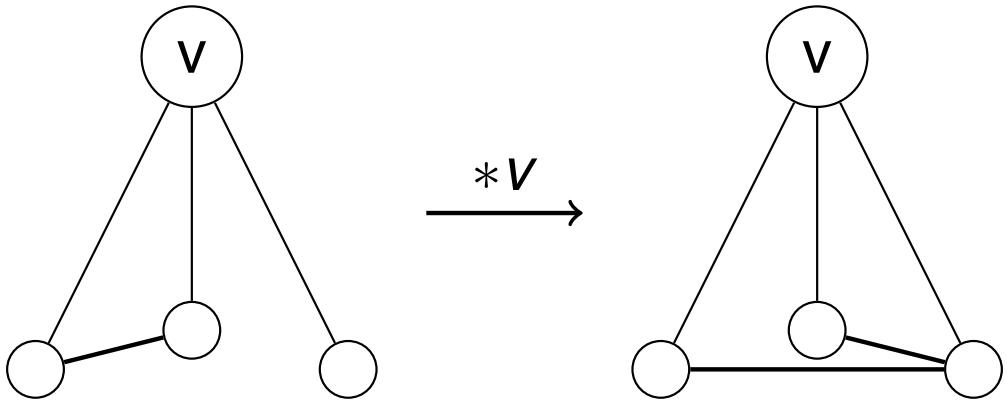
1. on calcule $X \subseteq V(G) \backslash B$ tel que les lignes de $M_{(V(G)\backslash B, B)}$ associées aux éléments de X forment une base de cette matrice.
2. Pour toute partition (X_1, X_2) de X, on cherche un ensemble $Z \in V(G)$ de cut-rank minimal et inférieur à $\min\{|X_1|, |X_2|)\}$ à tel que :
   i) $Z \cap X = X_1$;
   ii) $X \backslash Z = X_2$

Si un tel ensemble n'existe pas, on renvoie un message d'erreur.

Lemme (Oum-Seymour 06) : l'ensemble Z ainsi trouvé vérifie les propriétés demandées.

Définitions et propriétés

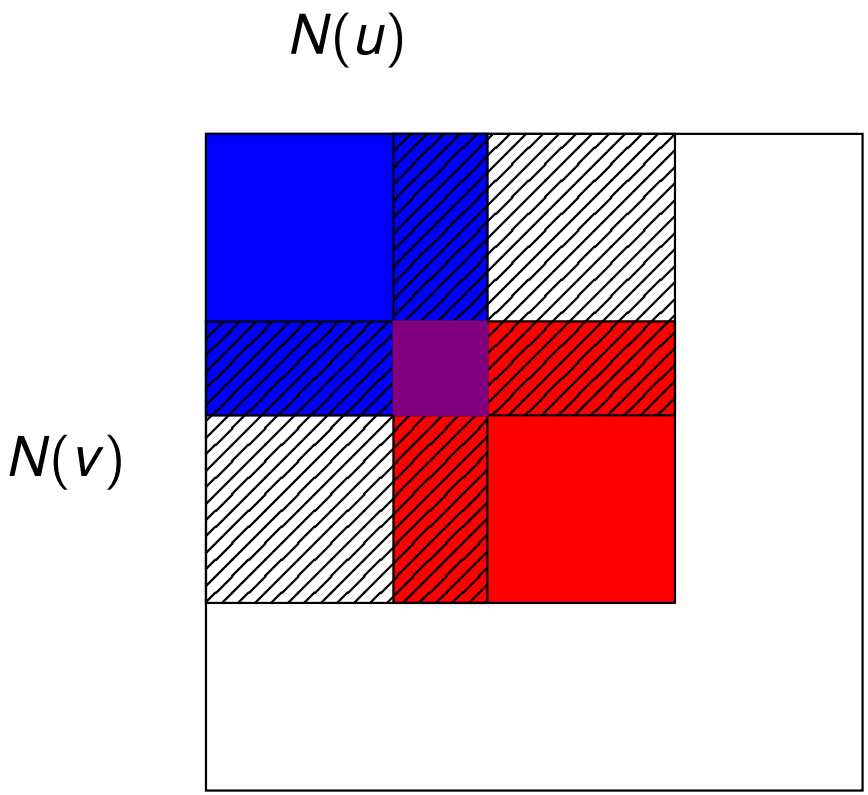
L'opération de complémentation locale de G en un sommet v, notée $G * v$



ne change pas le cut-rank.
De même pour l'opération de pivot de l'arête uv de G, notée $G \wedge uv$ et définie par $G \wedge uv = G * u * v * u$.

Pivot

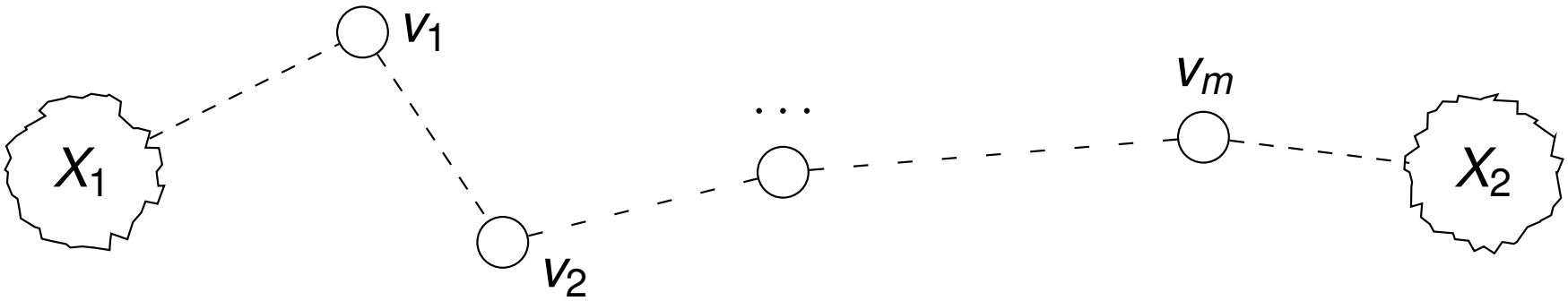
Le pivot est une opération fréquemment utilisée. Elle est implantée de la sorte :



On inverse les valeurs des parties hachurées.

Séquences bloquantes

$v_1, \ldots, v_m$ suite de sommets distincts de $V(G)\backslash(X_1 \cup X_2)$ est une séquence bloquante pour (X_1, X_2) dans G si elle est la plus courte séquence telle que :



- cut-rank$^*_G(X_1, X_2 \cup \{v_1\}) > r$;
- cut-rank$^*_G(X_1 \cup \{v_i\}, X_2 \cup \{v_{i+1}\}) > r$;
- cut-rank$^*_G(X_1 \cup \{v_m\}, X_2) > r$

Avec $r =$ cut-rank$^*_G(X_1, X_2)$.

Déroulement de Oum 08

Entrée : un graphe G, deux ensembles disjoints $X_1,\ X_2 \in V(G)$ et un entier k
Sortie : l'ensemble Z dont on a besoin dans Oum-Seymour 06

On travaille sur G' une copie de G.
Pour tout $i \in [\![0, k]\!]$, cherche une séquence bloquante pour (X_1, X_2) dans G' :

- inexistante : retourne $Z = X_1 \cup \{\text{sommets accessibles dexpuis } X_1\}$ de cut-rank cut-rank$^*_{G'}(X_1, X_2)$ (lemme Oum 08) ;
- $v_1, \ldots, v_m$:
  Pour tout $j \in [\![0, m-1]\!]$
  - on cherche $w \in X_2$ adjacent à v_{m-j} dans G' ;
  - on pose $G' = G' \wedge wv_{m-j}$.

Sommaire

Choix d'implantation

- Oum-Seymour 06 (décomposition) :
  - matrice d'adjacence pour l'arbre de décomposition partielle (grossit au fur et à mesure que la décomposition s'affine) ;
  - file d'attente pour les feuilles qu'il reste à décomposer.

- Oum 08 (calcul de Z) :
  - matrice d'adjacence pour le graphe (plus facile pour les pivots) ;
  - le graphe utilisé est une copie du graphe donné en entrée, que l'on modifie à chaque pivot ;
  - recherche d'une séquence bloquante : parcours en largeur dans un graphe construit au fur et à mesure. On ne contruit tout le graphe que quand il n'existe pas de séquence bloquante.

SAGE

Les deux algorithmes ont été implémentés en python et utilisent des fonctions du système de calcul formel SAGE.

Avantages :

- syntaxe proche de python ;
- beaucoup de fonctions sont déjà codées ;
- bien documenté.

SAGE

Avec SAGE :

- calcul du rang ;
- extraction de sous-matrices ;
- pivot de Gauss ;
- accessibilité depuis un sommet.

Sans SAGE :

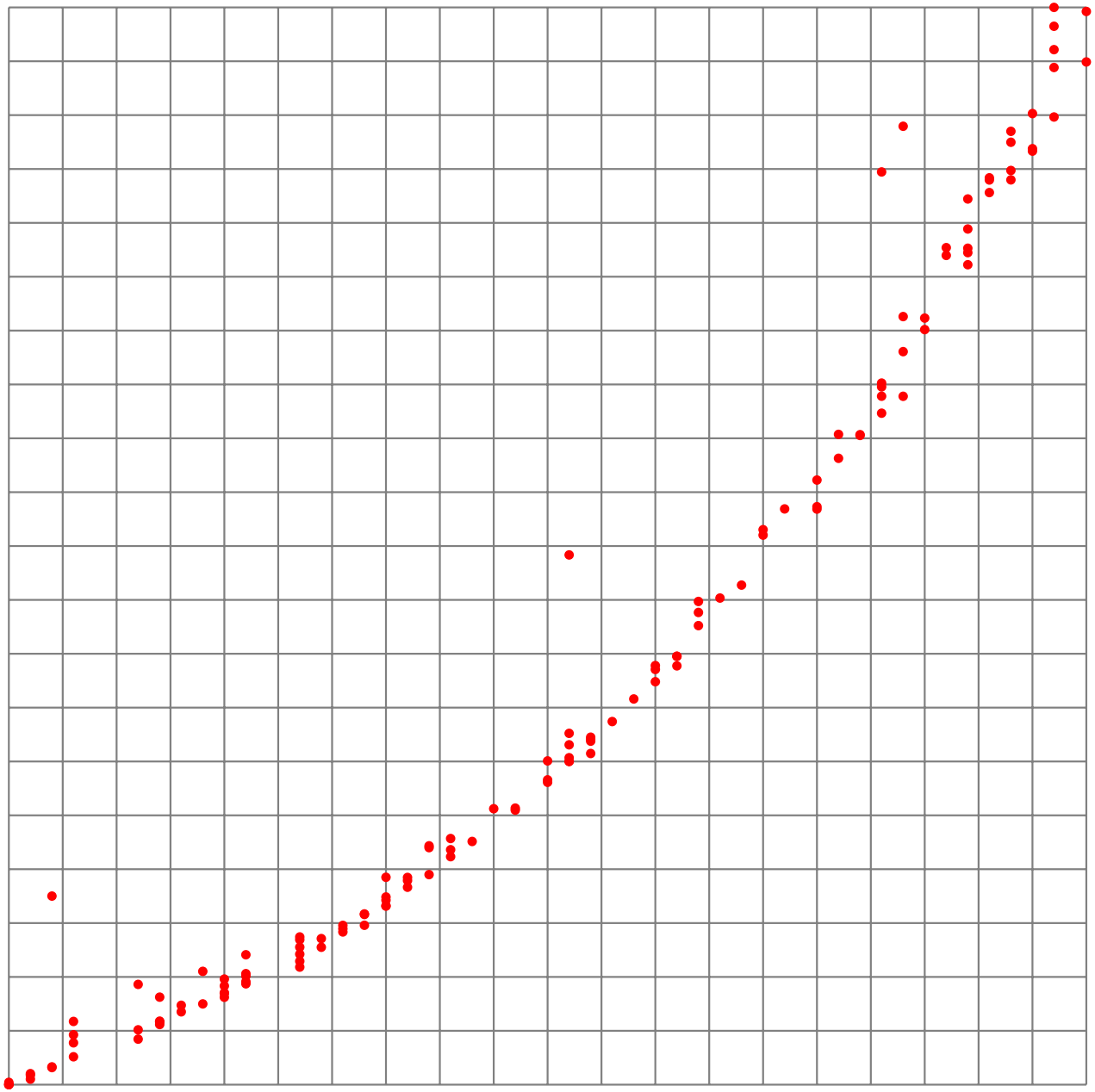
- cut-rank ;
- pivot d'une arête ;
- recherche de séquence bloquante (parcours en largeur, etc.).

Stockage du résultat

La classe suivante est utilisée pour stocker la décomposition en construction :

```
class ArbreDec:
    def __init__(self,M)
    def __repr__(self, pere=None)
    def show(self,size =(8,8))
    def save(self,nomFich)
    def feuilles(self,i,pere=None)
    def rang_de_coupe(self)
    def augmenteT(self,pere)
```

Résultats



Perspectives

Première implantation $\rightsquigarrow$ améliorations possibles.

Par exemple, pour le calcul de Z (coûteux en temps) :

- stocker le graphe auxiliaire par listes d'adjacence plutôt que par matrices ;
- ne pas calculer l'ensemble du graphe auxiliaire mais seulement les sommets accessibles depuis X_1 ;
- utiliser la partie du graphe calculée pendant la recherche de séquence bloquante.

Remerciements

Je tiens à remercier Binh-Minh Bui-Xuan et Philippe Trébuchet pour leur aide et leurs conseils.

The end

Merci pour votre attention.

Avez-vous des questions ?