Triangle Width

Entre l'ordonnancement et la visite de graphes

Luc Libralesso, en collaboration avec
Khadija Hadj-Salem, Vincent Jost et Frédéric Maffray

14 juin 2020

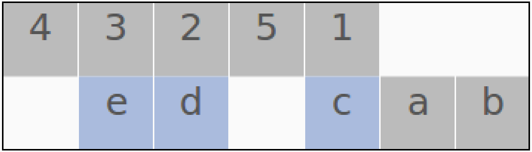
Midi ROSP - G-SCOP

Table of contents

Introduction, examples, vos réalisations

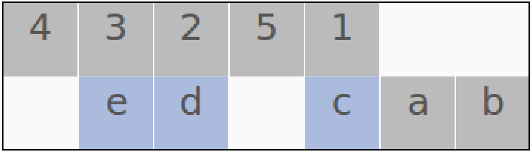
MCTP - nouveau problème d'ordonnancement

- Problème d'ordonnancement (deux machines)
- processeur d'input/output
- processeur de calcul
- certains calculs ont des prérequis



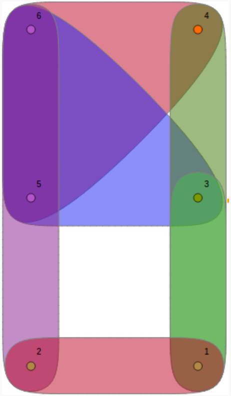
MCTP - nouveau problème d'ordonnancement

- Problème d'ordonnancement (deux machines)
- processeur d'input/output
- processeur de calcul
- certains calculs ont des prérequis



Visualiser une instance demande de visualiser un hypergraphe.

Example

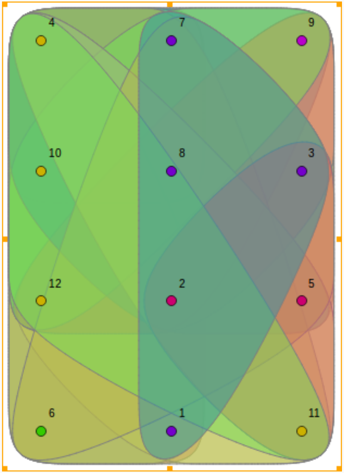


Instance 3

10	4	5	2	11	3	12	9	1	8	7	6							
				10	11	13	14	2	6	8	1	9	7	3	12	4	15	5

Instance 3

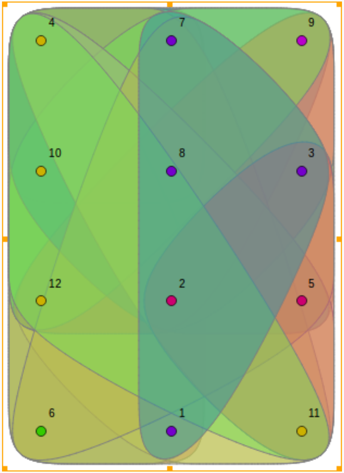
10	4	5	2	11	3	12	9	1	8	7	6							
				10	11	13	14	2	6	8	1	9	7	3	12	4	15	5

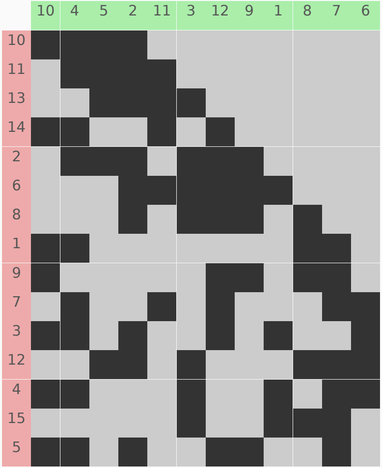


Instance 3

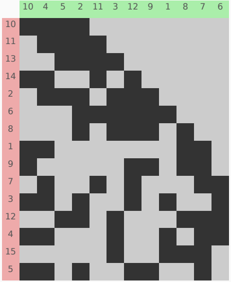
10 4 5 2 11 3 12 9 1 8 7 6

10 11 13 14 2 6 8 1 9 7 3 12 4 15 5



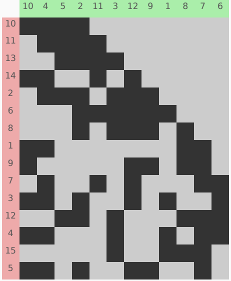


Remarques sur la matrice



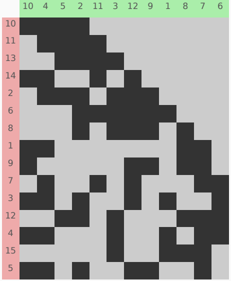
- Bien plus pratique (matrice de taille jusqu'à 1000x1000)

Remarques sur la matrice



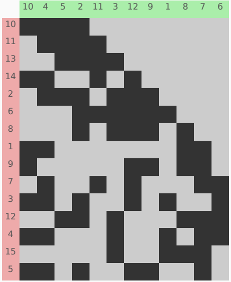
- Bien plus pratique (matrice de taille jusqu'à 1000x1000)
- Aucun chevauchement

Remarques sur la matrice



- Bien plus pratique (matrice de taille jusqu'à 1000x1000)
- Aucun chevauchement
- Structure en triangle

Remarques sur la matrice



- Bien plus pratique (matrice de taille jusqu'à 1000x1000)
- Aucun chevauchement
- Structure en triangle

Théorème : Maximiser le triangle de la matrice **équivalent** à maximiser la parallélisation des deux machines (et donc minimiser le makespan)

Ici est né **Triangle Width**

Échauffements

`librallu.gitlab.io/hypergraph-viz`

- vertical swap
- horizontal swap
- single swap
- warmup 1

Heuristique

choisir la ligne ajoutant le moins de cases noires et tasser à gauche.

Essayer sur Exemple 0

Remarques (suite)

- Transposer la matrice préserve l'optimalité

Remarques (suite)

- Transposer la matrice préserve l'optimalité
- Vrai aussi dans le problème d'ordonnancement

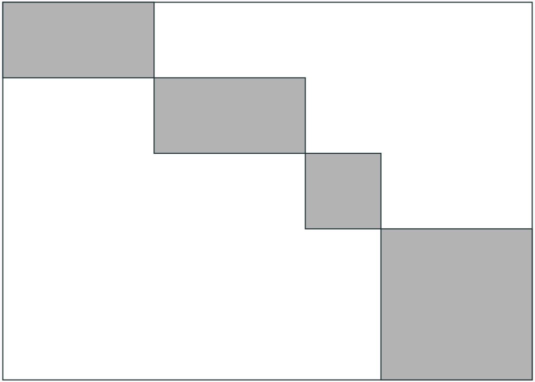
Remarques (suite)

- Transposer la matrice préserve l'optimalité
- Vrai aussi dans le problème d'ordonnancement
- **Utiliser la représentation en matrices peut aider à visualiser des propriétés d'un hypergraphe**

Remarques (suite)

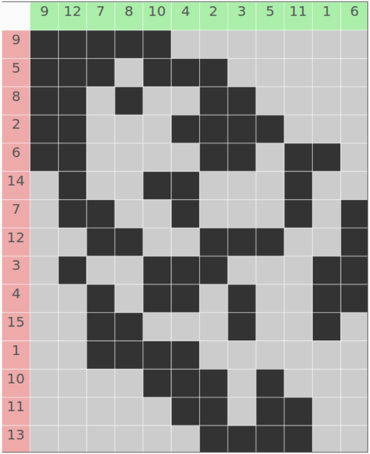
- Transposer la matrice préserve l'optimalité
- Vrai aussi dans le problème d'ordonnancement
- **Utiliser la représentation en matrices peut aider à visualiser des propriétés d'un hypergraphe**
- **A condition d'avoir la bonne fonction objectif**

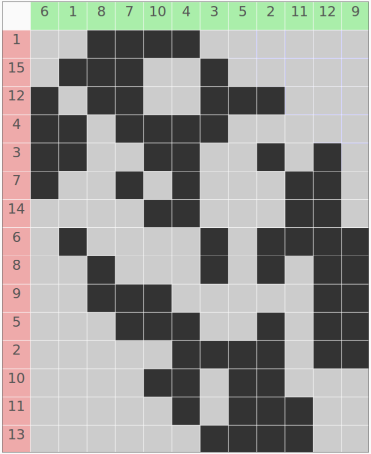
Choix du bon ordre - Composantes connexes



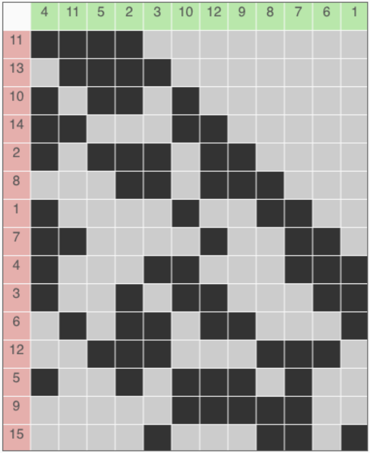
- En blanc, les zones vides
- En gris, les zones pouvant contenir des cases noires

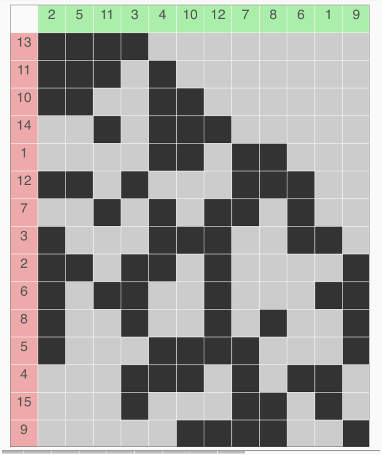
Vos réalisations - (Hypergraph) Bandwidth

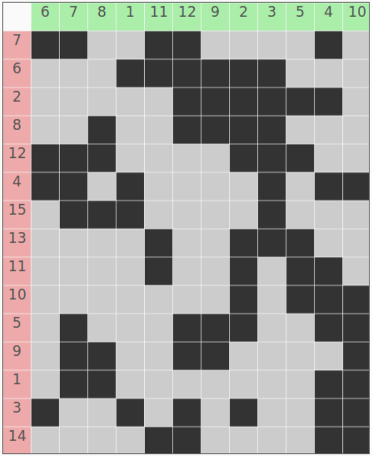


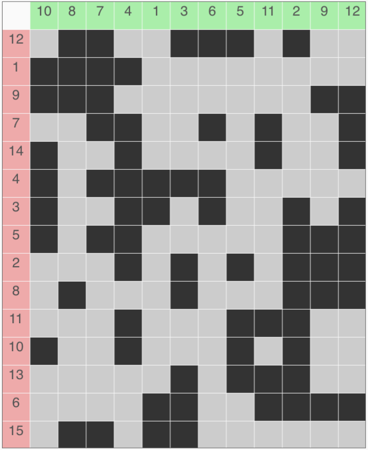


Vos réalisations - Triangle Width

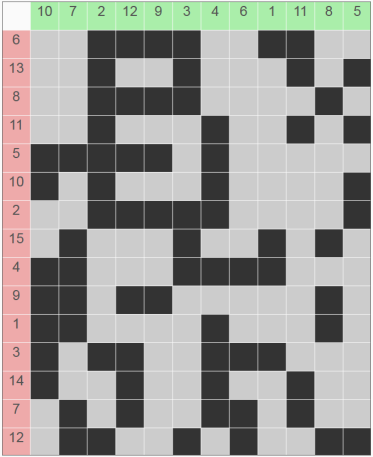


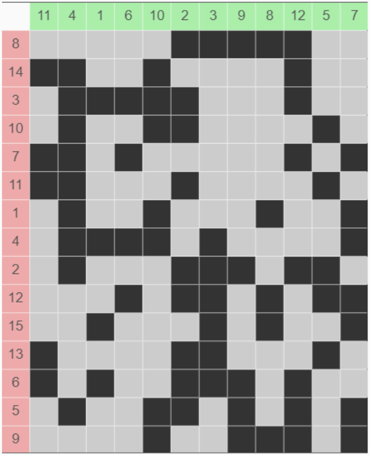



6 7 8 1 11 12 9 2 3 5 4 10
7 6 2 8 12 4 15 13 11 10 5 9 1 3 14


10 8 7 4 1 3 6 5 11 2 9 12
12 1 9 7 14 4 3 5 2 8 11 10 13 6 15

Vos réalisations - Formes





Sous problèmes et complexité

Cas d'un graphe au lieu d'un hypergraphe

- Dans le cas général, *Triangle Width* est $\mathcal{NP}$-Complet.
- Dans le cas d'un graphe, *Triangle Width* est polynômial.

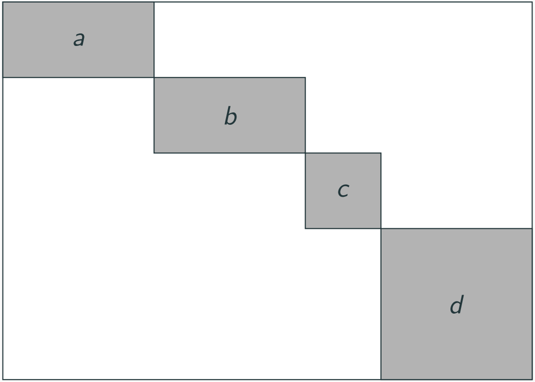
Algorithme exact pour le cas d'un graphe

Les sommets sont les prérequis et les arêtes les tâches

1. Décomposition en composantes connexes
2. Trier les composantes connexes par densité décroissante ($m - n$)
3. Parcours en largeur de chaque composante connexe

Cas de carrés noirs

Dans le cas de carrés noirs dans la matrice, il suffit d'ordonnancer les blocs



Il s'agit d'un FlowShop à 2 machines ($F2//C_{max}$)

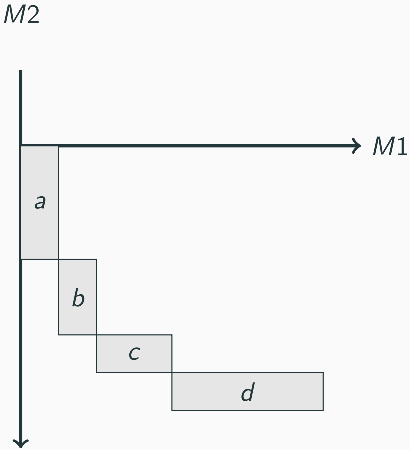
Flowshop à 2 machines - $F2//C_{max}$

Flowshop à 2 machines - $F2//C_{max}$

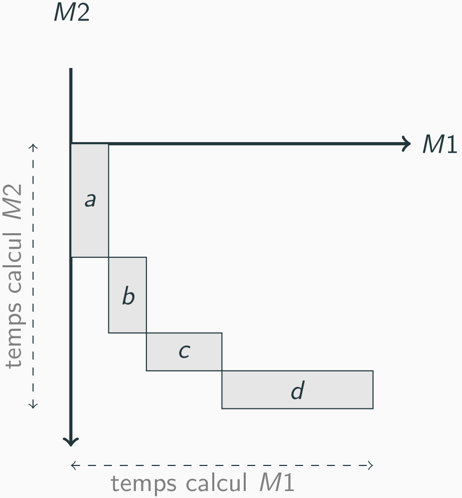
- Algorithme de Johnson : optimal $O(n \log n)$
- Trier les tâches suivant l'ordre : $\min(P_{i1}, P_{j2}) \leq \min(P_{j1}, P_{i2})$

Maximiser la Triangle Width revient à minimiser le makespan

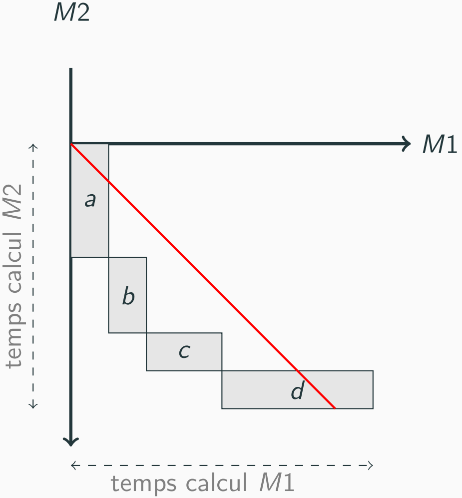
Maximiser la Triangle Width revient à minimiser le makespan



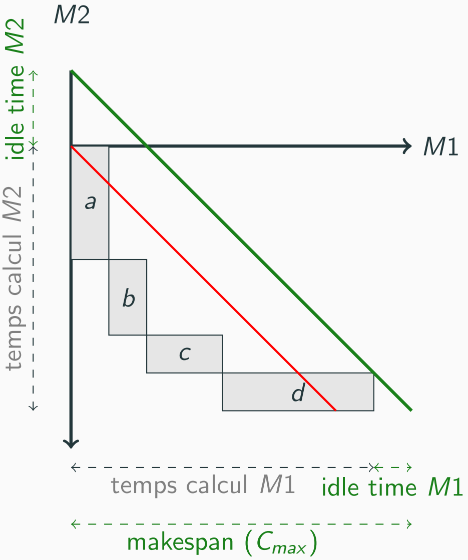
Maximiser la Triangle Width revient à minimiser le makespan



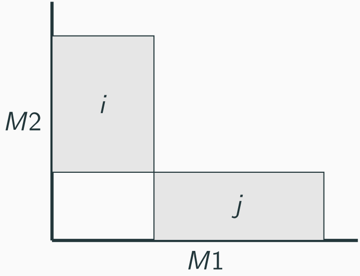
Maximiser la Triangle Width revient à minimiser le makespan



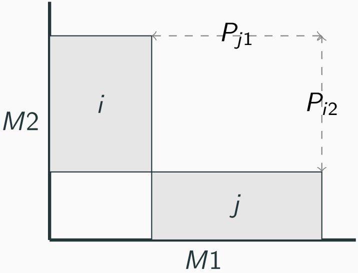
Maximiser la Triangle Width revient à minimiser le makespan



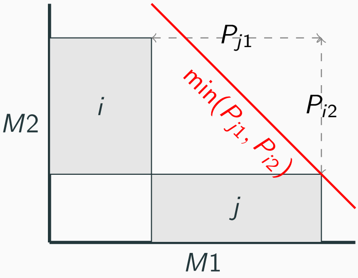
Règle de Johnson - Preuve par argument d'échange visuel



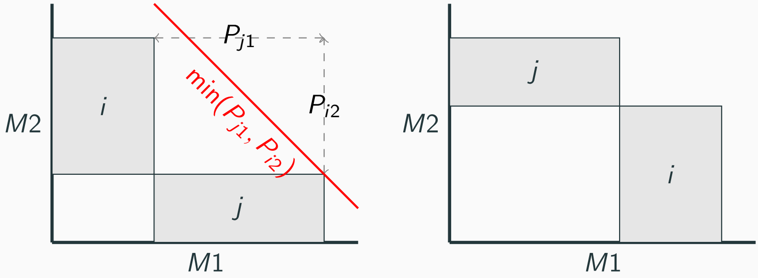
Règle de Johnson - Preuve par argument d'échange visuel



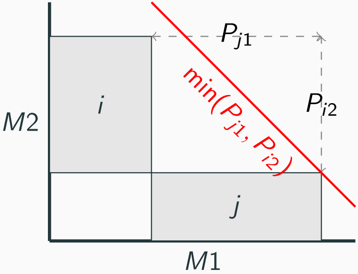
Règle de Johnson - Preuve par argument d'échange visuel

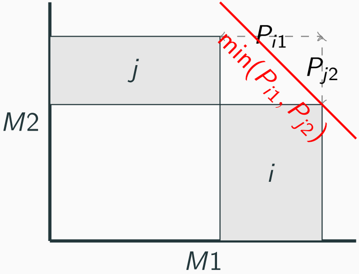


Règle de Johnson - Preuve par argument d'échange visuel

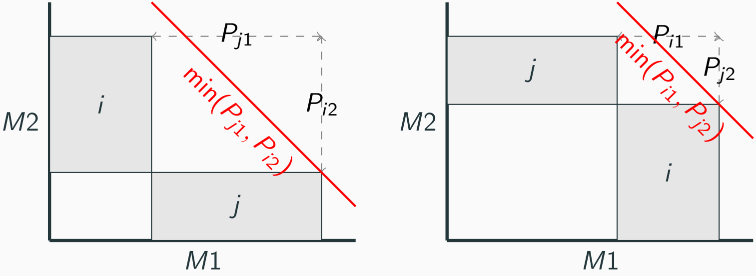


Règle de Johnson - Preuve par argument d'échange visuel



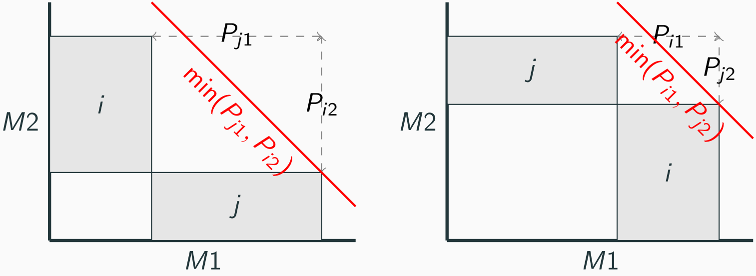


Règle de Johnson - Preuve par argument d'échange visuel



$$\min(P_{j1}, P_{i2}) \quad \geq \quad \min(P_{i1}, P_{j2})$$

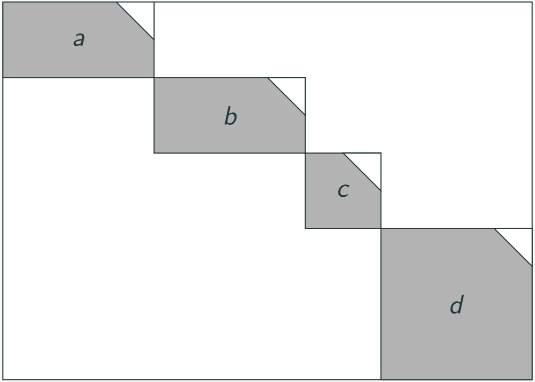
Règle de Johnson - Preuve par argument d'échange visuel



$$\min(P_{j1}, P_{i2}) \geq \min(P_{i1}, P_{j2})$$
$$\min(P_{i1}, P_{j2}) \leq \min(P_{j1}, P_{i2})$$

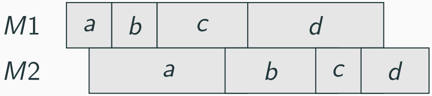
Divide and Conquer - carrés noirs moins triangle

Un autre cas un peu plus général :



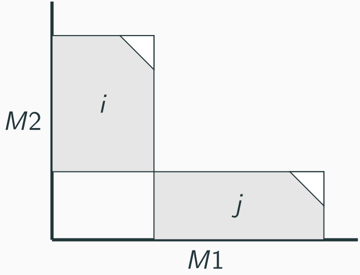
Variante de FlowShop ($F2/t_j \in [-\min(P_{j1}, P_{j2}), 0], perm/C_{max}$)

Variante de Johnson où on autorise la deuxième machine à commencer une tâche avant que cette tâche soit finie sur la machine 1.



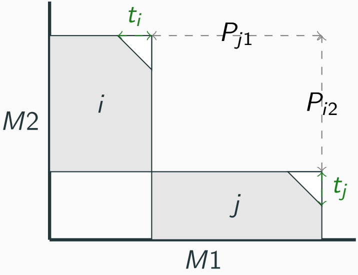
Trouver la règle associée

On applique le même principe qu'avec la règle de Johnson.



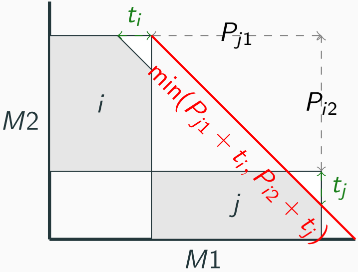
Trouver la règle associée

On applique le même principe qu'avec la règle de Johnson.



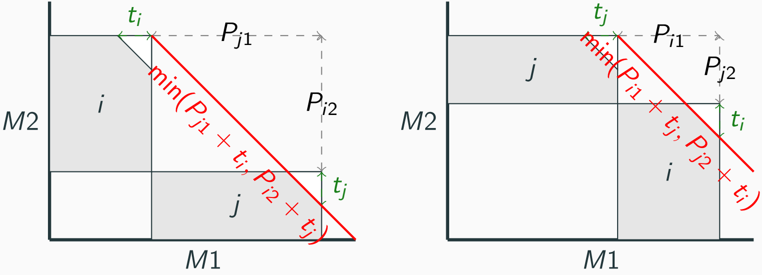
Trouver la règle associée

On applique le même principe qu'avec la règle de Johnson.



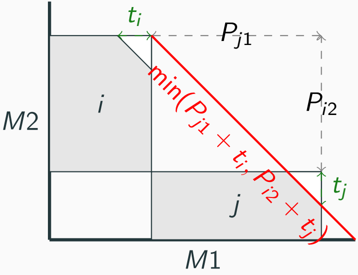
Trouver la règle associée

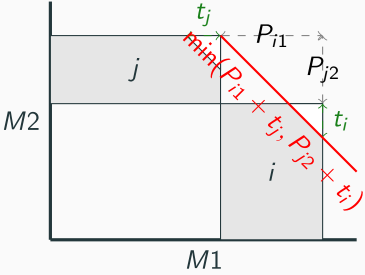
On applique le même principe qu'avec la règle de Johnson.



Trouver la règle associée

On applique le même principe qu'avec la règle de Johnson.

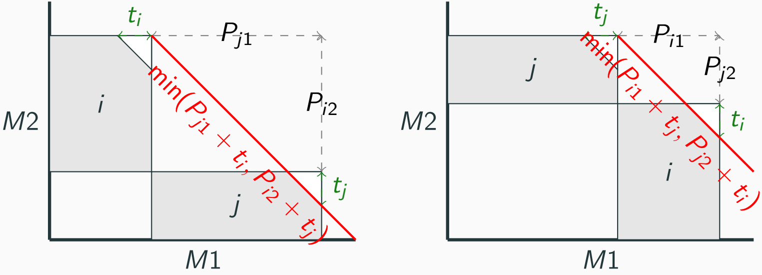




$$\min(P_{j1} + t_i, P_{i2} + t_j) \qquad \geq \qquad \min(P_{i1} + t_j, P_{j2} + t_i)$$

Trouver la règle associée

On applique le même principe qu'avec la règle de Johnson.



$$\min(P_{j1} + t_i, P_{i2} + t_j) \geq \min(P_{i1} + t_j, P_{j2} + t_i)$$
$$\min(P_{i1} + t_j, P_{j2} + t_i) \leq \min(P_{j1} + t_i, P_{i2} + t_j)$$

NP-Hardness

La preuve a été trouvée récemment.

NP-Hardness

La preuve a été trouvée récemment.
réduction utilisant k-weak visit (présenté dans [1]).
ENTRÉE : Un graphe $G = (V, E)$, un entier $k \geq 1$, les k premières positions de parcours.
QUESTION : En choisissant k sommets de départ, choisir un ordre pour le reste des sommets sans être en "manque de carburant".

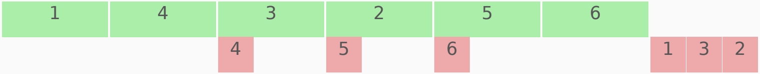
sous-cas de k-weak visit

k-weak visit prend en entrée un graphe, un entier k et une liste des k premiers sommets.

Question : Quelle est la complexité du problème si on ne donne pas la liste de sommets ? Plusieurs problèmes similaires :

sous-cas de MCTP

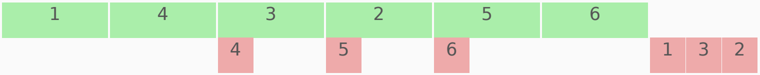
Changeons la durée des prérequis (α) et tâches (β)



- Si hypergraphe : $\mathcal{NP}$-Complet (même pour $\alpha = \beta = 1$)

sous-cas de MCTP

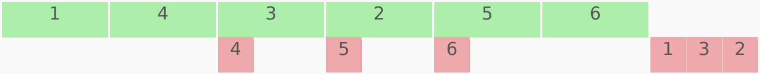
Changeons la durée des prérequis (α) et tâches (β)



- Si hypergraphe : $\mathcal{NP}$-Complet (même pour $\alpha = \beta = 1$)
- Si graphe :
    - $\alpha \leq \beta$: $\mathcal{P}$

sous-cas de MCTP

Changeons la durée des prérequis (α) et tâches (β)



- Si hypergraphe : $\mathcal{NP}$-Complet (même pour $\alpha = \beta = 1$)
- Si graphe :
  - $\alpha \leq \beta$: $\mathcal{P}$
  - $\alpha > \beta$: Ouvert

Bornes de Triangle Width

Preuve d'optimalité

Il existe une borne dans la littérature [3] :

$$lb_1 = |\mathcal{Y}| + \min_{y \in \mathcal{Y}}(|M_y|)$$

A essayer

`librallu.gitlab.io/hypergraph-viz`

- warmup 1
- warmup 2

Preuve d'optimalité

- La borne précédente ne marche pas
- **Théorème :** triangle width de $M =$ triangle width de M^T.
- La borne précédente s'applique sur M^T

$$lb'_1 = |\mathcal{X}| + \min_{x \in \mathcal{X}}(|M_x^T|)$$

Généralisation des bornes

Example 1 (hypergraph-viz)

Regarder tous les subsets de taille k. Choisir celui avec le moins de prérequis.

$$lb_k = \left(\min_{s \subseteq \mathcal{Y}, |s| = k} \left| \bigcup_{y \in s} M_y \right| \right) + (|\mathcal{Y}| - k + 1) \tag{1}$$

Conclusion

En bref

Méthode de visualisation d'hypergraphe

- A condition de trouver une bonne fonction objectif
- paradigme similaire pour les graphes (matrice carrées symétriques)

En bref

Méthode de visualisation d'hypergraphe

- A condition de trouver une bonne fonction objectif
- paradigme similaire pour les graphes (matrice carrées symétriques)

Il reste des problèmes ouverts (contributions bienvenues :-))

- *k*-weak visit modifié
- MCTP où la durée des prérequis est plus grande que la durée des tâches

Questions ?

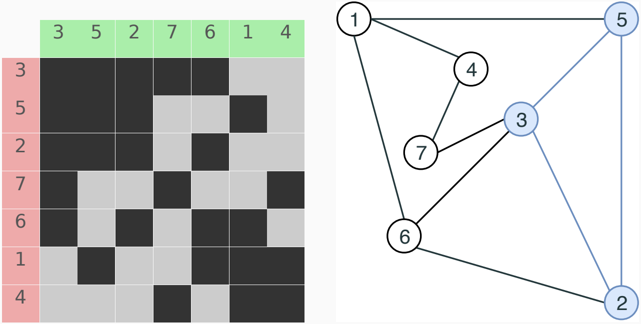
Références i

C. Arbib, M. Flammini, and E. Nardelli.
How to survive while visiting a graph.
Discrete applied mathematics, 99(1-3) :279–293, 2000.

M. R. Garey and D. S. Johnson.
Computers and intractability, volume 29.
wh freeman New York, 2002.

K. H. Salem, Y. Kieffer, and S. Mancini.
Formulation and practical solution for the optimization of memory accesses in embedded vision systems.
In *Computer Science and Information Systems (FedCSIS), 2016 Federated Conference on*, pages 609–617. IEEE, 2016.

Choix du bon ordre - Clique max



Et plein d'autres...

- clique max (carré de cases noires)
- stable max (carré de cases blanches)
- limiter le nombre de bordures
- Chemin hamiltonien (ligne diagonale noire)
- etc.

Induced subgraph with property Π

Référence dans [2]

- $\textsc{Input}$: Graphe $G = (V, E)$, entier $k \leq V$
- $\textsc{Question}$: Existe-t-il un sous-ensemble $V' \subseteq V$, $|V'| \geq k$ qui a la propriété Π ?