Graph based parsing
GBP

Graph based Parsing

- Les structures à prédire dans le cadre de l'analyse en dépendance sont des graphes (arbres de dépendances).
- Peut-on prédire directement le graphe correspondant à une phrase ?
- Au cœur de GBP, la **fonction de score** d'un arbre de dépendance $T = (V, A) \in \mathcal{T}(S)$:

$$\lambda(T) = \lambda(V, A) \in \mathbb{R}$$

- Etant donné une phrase S, on cherche l'arbre $\hat{T}$ de score maximal

$$\hat{T} = \underset{T \in \mathcal{T}(S)}{\arg\max}\, \lambda(T)$$

Différences clefs avec Transition based Parsing

- On prédit directement la structure finale et pas les actions qui permettront de construire le graphe.
  - Pas besoin de décomposer un arbre en séquence de configurations et transitions à l'aide d'un oracle.
- Méthode exacte : on produit l'arbre qui maximise la fonction score
  - Considérer l'ensemble $\mathcal{T}(S)$ de tous les arbres possibles d'une phrase S sans énumérer les éléments de l'ensemble.

Schéma général

1. Définir une fonction de factorisation d'un arbre T ($\Psi(T)$)
2. Définir une fonction de score $\lambda(\psi)$ pour un facteur ψ
3. Définir une fonction de score $\lambda(T)$ pour un arbre T
4. Choisir un algorithme d'analyse

$$\hat{T} = \underset{T \in \mathcal{T}(S)}{\arg\max}\, \lambda(T)$$

Notations

- $S = w_0 w_1 \dots w_n$ une phrase
- $T = (V, A)$ un arbre de dépendances
- $\mathcal{T}(S)$ l'ensemble des arbres possibles pour la phrase S
- $\Psi(T)$ ensemble des facteurs d'un arbre T
- $\Psi(S)$ ensemble des facteurs potentiels d'une phrase S
- $\mathcal{L} = \{l_1 \dots l_m\}$ ensemble des étiquettes des relations syntaxique

Décomposition en facteurs

- On ne peut estimer directement le score d'un arbre à partir des données car la probabilité d'occurrence d'un arbre particulier est, en général, nulle.
- On décompose un arbre en **facteurs** à l'aide d'une fonction de factorisation.

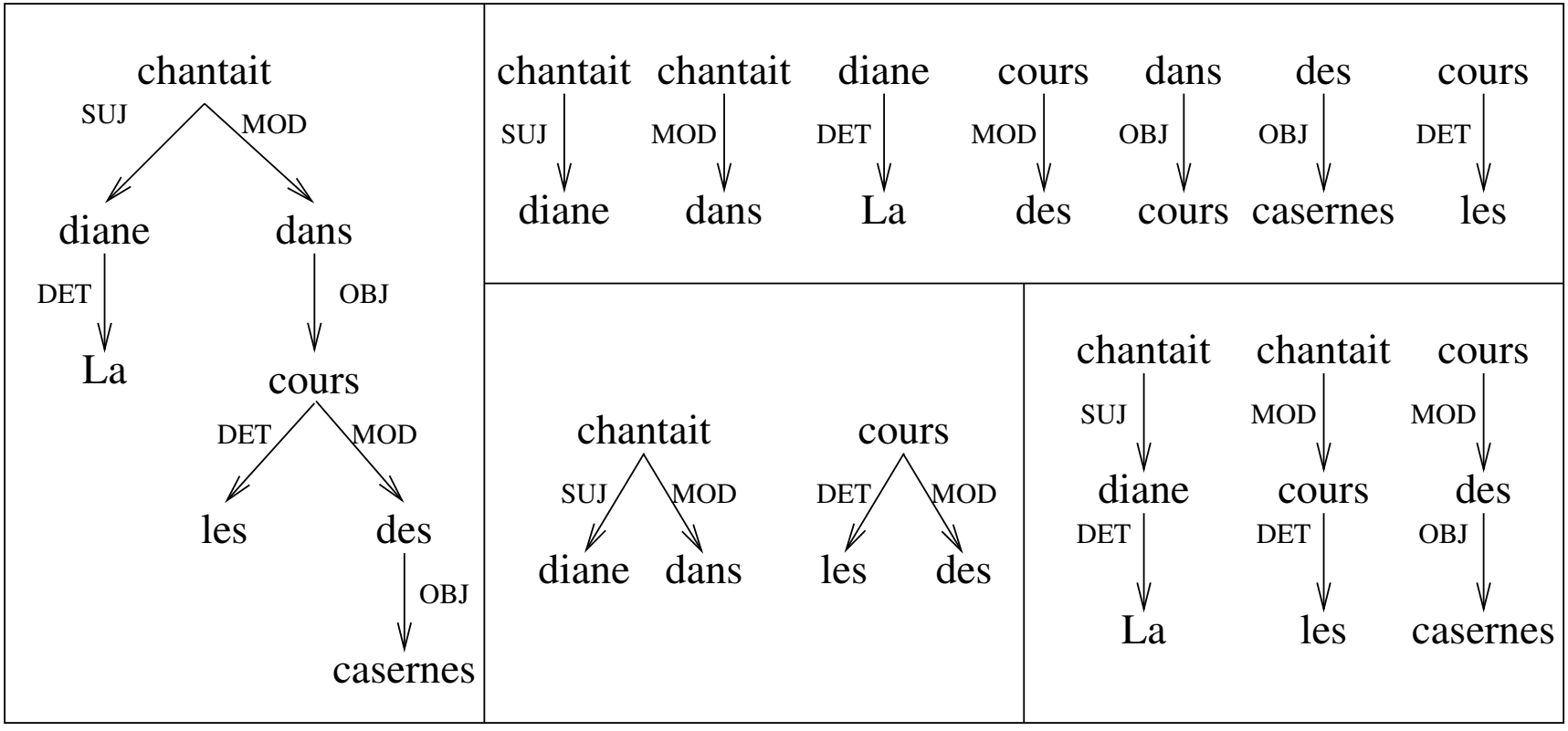
$$\Psi(T) = \{\psi_1, \ldots, \psi_q\}$$

- Le score d'un arbre est une fonction du score de ses facteurs

Fonction de factorisation

- Cas le plus simple : modèles d'ordre 1 ($\Psi_1(T)$)
  - un facteur correspond à une dépendance.
  - un arbre comportant n nœuds comporte $n-1$ dépendances donc $n-1$ facteurs ($O(n)$)
- modèles d'ordre 2 ($\Psi_2(T)$) :
  - les facteurs sont composés de deux dépendances
  - plusieurs manières de combiner deux dépendances au sein d'un facteur
    - Grand parents : $A \rightarrow B \rightarrow C$
    - Frères : $A \leftarrow B \rightarrow C$

Exemple


chantait
SUJ
MOD
diane
dans
DET
OBJ
La
cours
DET
MOD
les
des
OBJ
casernes
chantait
SUJ
diane
chantait
MOD
dans
diane
DET
La
cours
MOD
des
dans
OBJ
cours
des
OBJ
casernes
cours
DET
les
chantait
SUJ
MOD
diane
dans
cours
DET
MOD
les
des
chantait
SUJ
diane
DET
La
chantait
MOD
cours
DET
les
cours
MOD
des
OBJ
casernes

Facteurs d'une phrase

- Les facteurs d'une phrase $\Psi(S)$ est l'ensemble de tous les facteurs de tous les arbres possibles de S

$$\Psi(S) = \bigcup_{T \in \mathcal{T}(S)} \Psi(T)$$

- On a en général

$$|\Psi(S)| \ll |\mathcal{T}(S)|$$

- Etant donné une phrase S de longueur n :
  - $|\Psi_1(S)|$ est une fonction quadratique de n
  - $|\Psi_2(S)|$ est une fonction cubique de n
- On se limitera ici aux modèles d'ordre 1.

Facteurs d'une phrase

- Les facteurs d'ordre 1 de la phrase S peuvent être stockés dans une table $\lambda_1[g][d][l]$
- avec :
  - $1 \leq g \leq n$ l'indice du gouverneur
  - $1 \leq d \leq n$ l'indice du dépendant
  - $1 \leq l \leq |\mathcal{L}|$ une étiquette de dépendance
- Le remplissage de λ_1 est réalisé en $O(n^2)$

Score des facteurs d'ordre 1

- Le score des facteurs est calculé à l'aide d'un classifieur.
- Ce dernier prend en entrée un couple (paire ordonnée) de mots (g, d) et calcule une distribution de probabilités sur l'ensemble des étiquettes.
- La probabilité p_l associée à l'étiquette l est interprétée comme le score $\lambda((g, l, d))$ de la dépendance (g, l, d).

Fonction de décomposition (Feature function)

- Une feature représente un aspect d'un couple de mots (g, d) qui semble intéressant pour prédire l'existence d'une dépendance entre g et d.
- Exemples
  - la forme de g ou de d
  - la partie de discours de g ou de d
  - la distance entre g et d dans la phrase
  - ...
- La fonction de décomposition représente une couple de mots (g, d) sous la forme d'un vecteur de features.

$$f_D : V_S \times V_S \to \mathcal{X}$$

- Le vecteur de features constitue l'entrée du classifieur.

Score d'un arbre

- le score d'un arbre est une fonction des scores de ses facteurs

$$\lambda(T) = f(\lambda(\psi_1), \ldots, \lambda(\psi_q)) \; \forall \; \psi_i \in \Psi(T)$$

- f peut prendre n'importe quelle forme, on utilisera ici une simple somme :

$$\lambda(T) = \sum_{\psi \in \Psi(T)} \lambda(\psi)$$

Recherche de $\hat{T}$

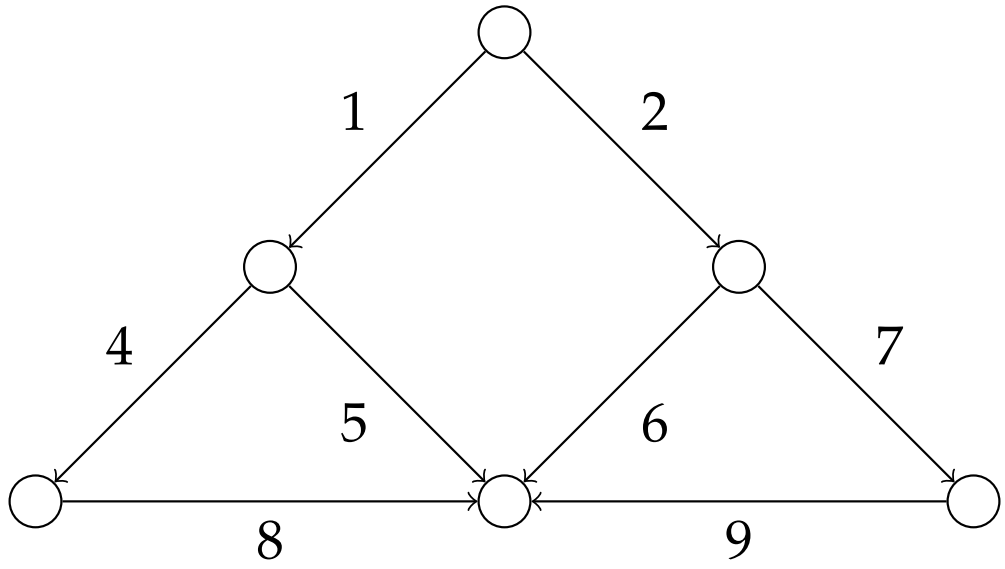
Deux familles de méthodes :

- Analyse par recherche de l'arbre couvrant maximal (Maximum Spanning Tree)
  - Algorithme de Chu-Liu-Edmonds
- Programmation dynamique
  - Algorithme CYK
  - variante de Eisner

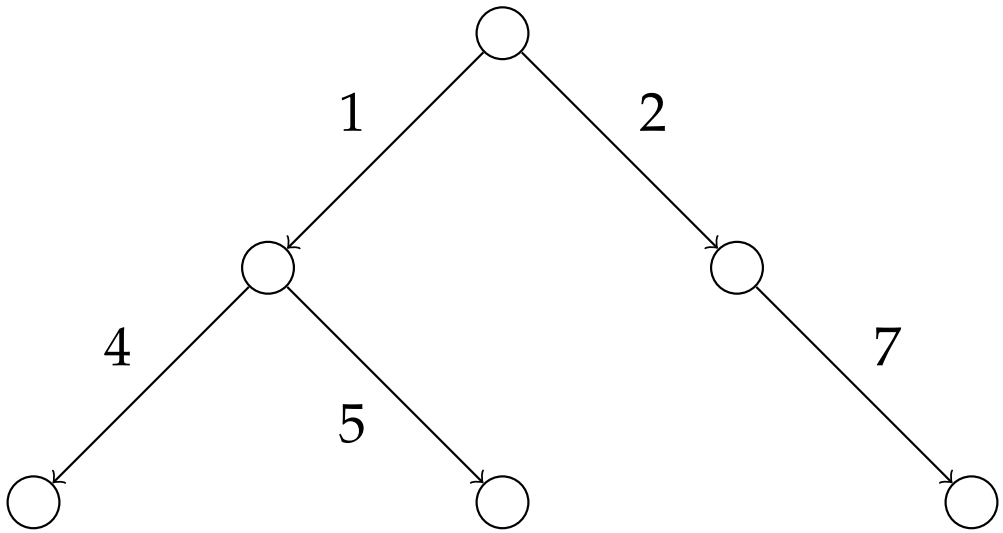
Arbre couvrant maximal

- Etant donné :
  - un graphe orienté $G = (V, A)$
  - une fonction λ associant un poids à chaque arc de A
- Un arbre couvrant de G est un sous-graphe $G' = (V', A')$ tel que
  - $V' = V$
  - G' est un arbre orienté
- Un arbre couvrant maximal de G est
  - un arbre couvrant de G de poids maximal
  - où le poids d'un sous-graphe est la somme des poids de ses arcs
- En général, la recherche d'un arbre couvrant maximal ne peut être faite de manière triviale en énumérant les arbres couvrants de G car ils sont trop nombreux !

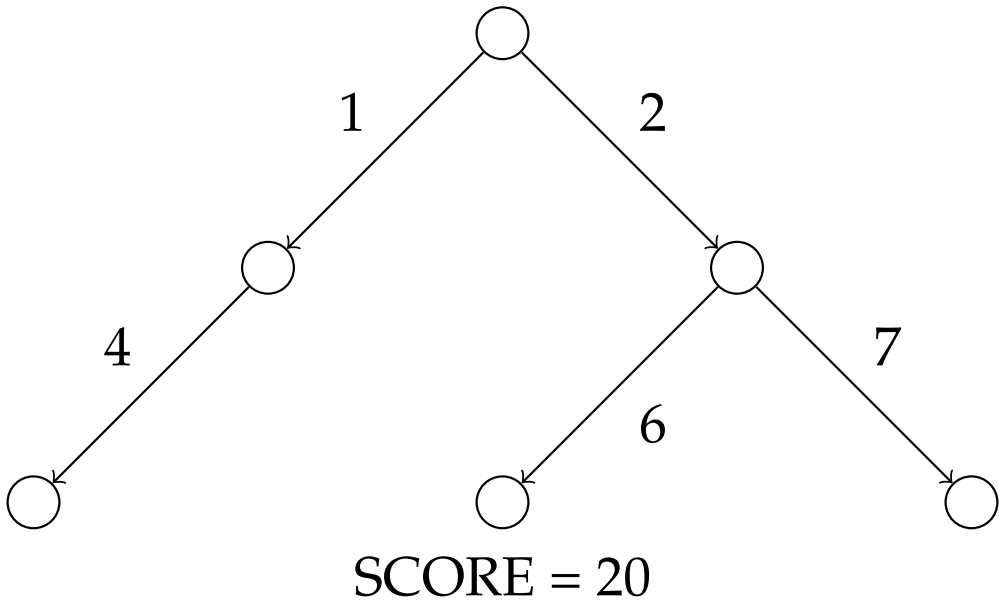
Exemple



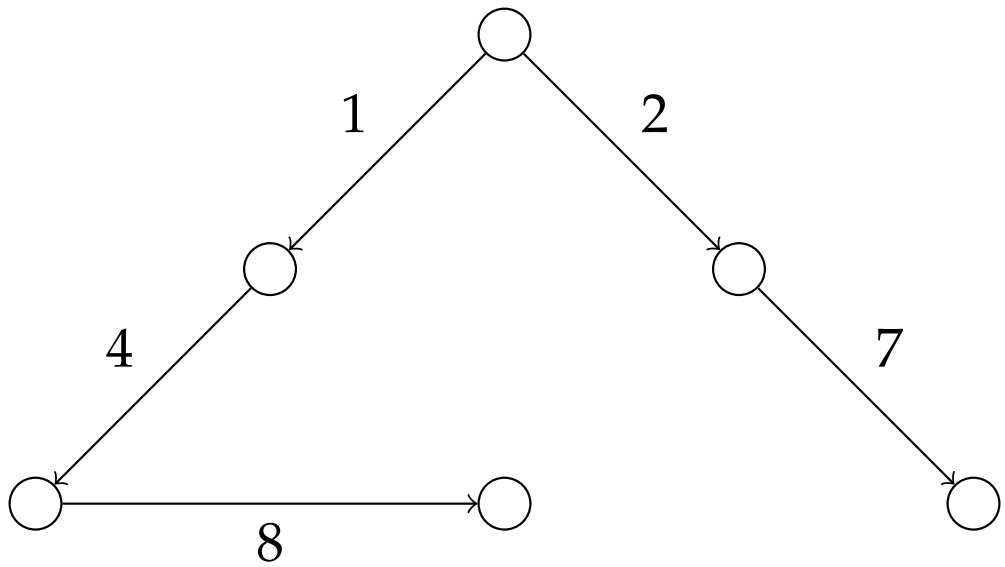
Exemple



Exemple

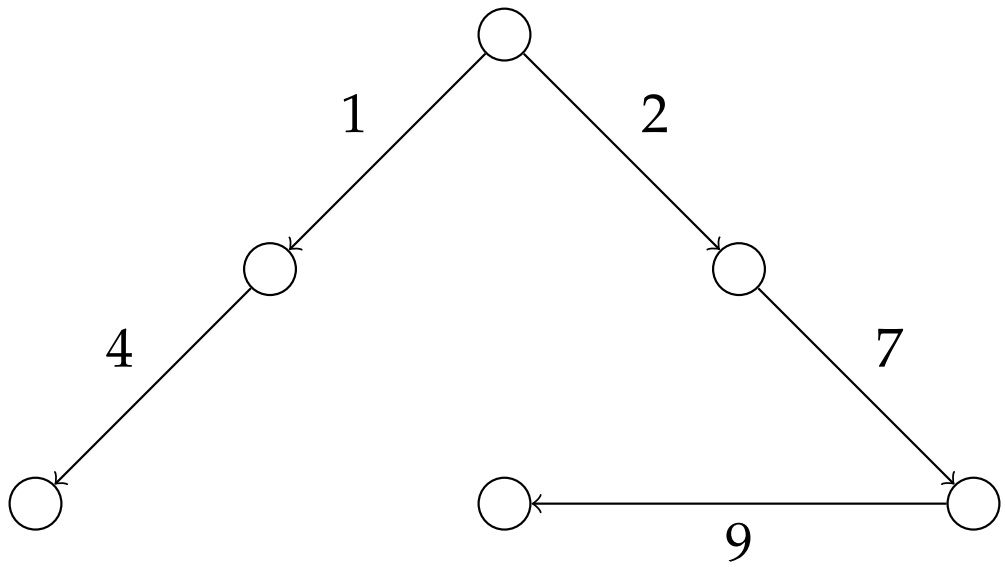


Exemple



SCORE = 22

Exemple



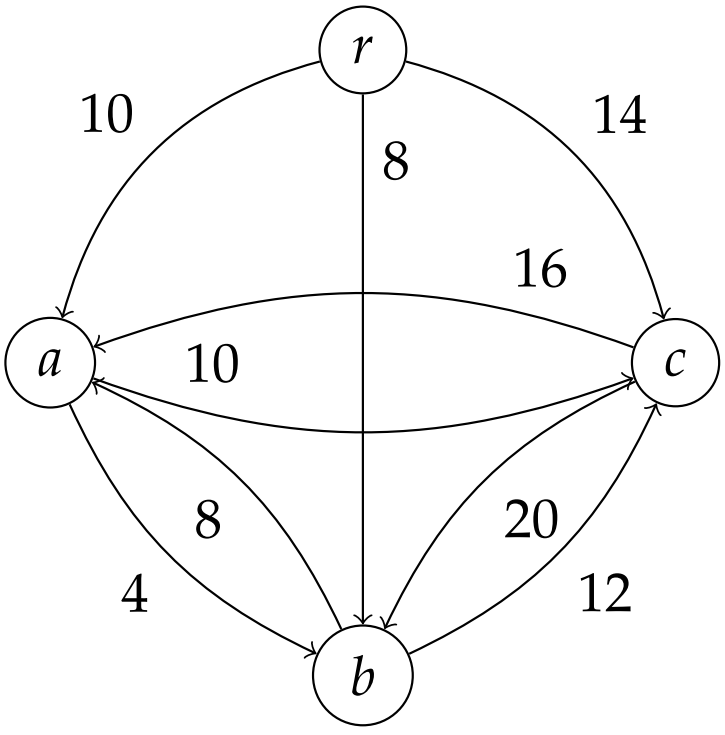
SCORE = 23

Algorithme de Chu-Liu-Edmonds : Idée générale

Tout nœud qui n'est pas la racine r doit avoir exactement un arc entrant

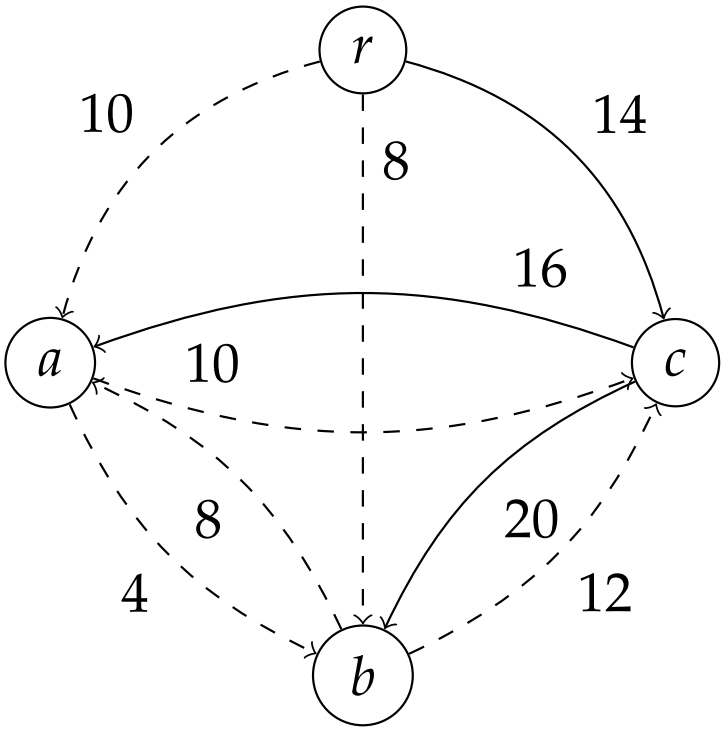
- Choisir pour chaque sommet (sauf r) l'arc entrant de poids maximal
- Si la structure produite est un arbre, on s'arrête
- Sinon, elle contient un cycle C
- Choisir un arc entrant pour C
- le choix de l'arc entrant de C élimine un arc de C

Algorithme de Chu-Liu-Edmonds



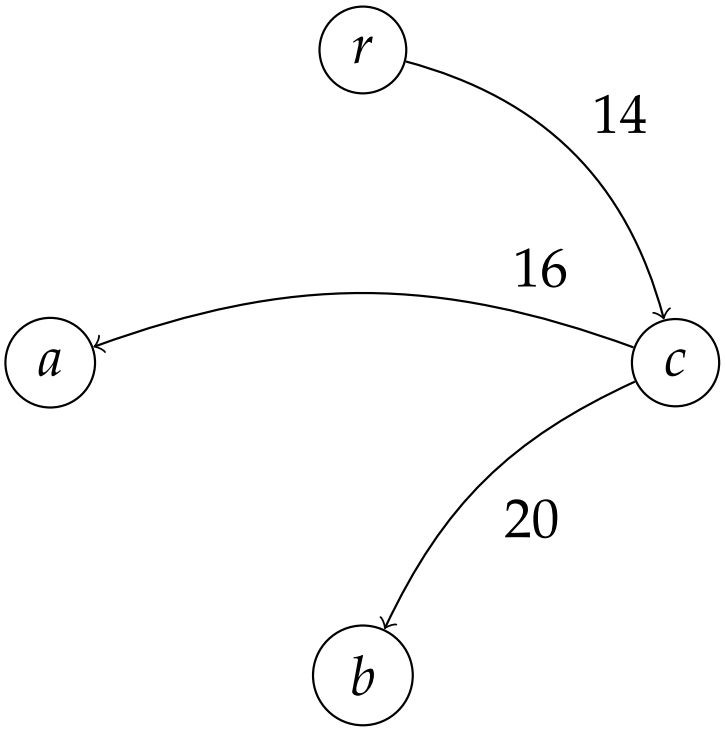
On choisit pour chaque sommet l'arc entrant de poids maximal

Algorithme de Chu-Liu-Edmonds



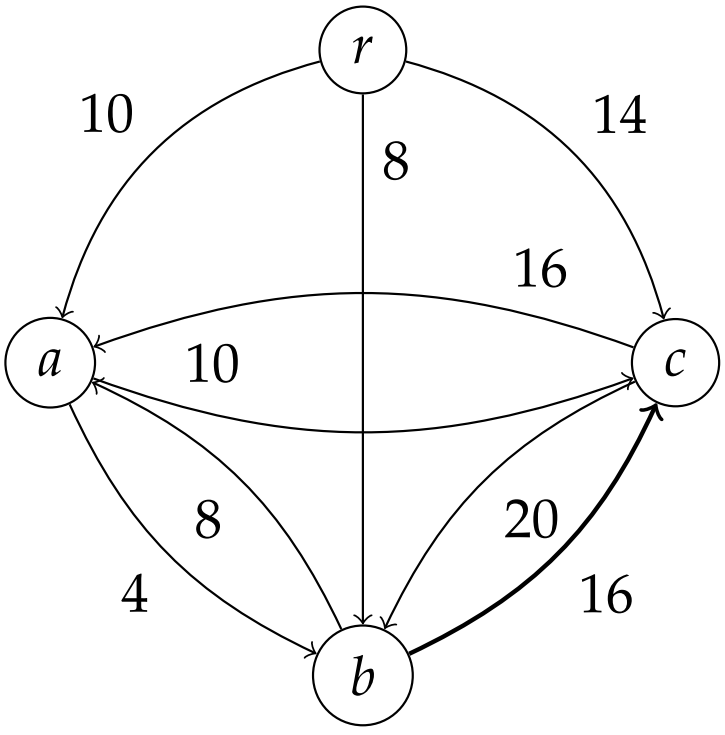
On choisit pour chaque sommet l'arc entrant de poids maximal

Algorithme de Chu-Liu-Edmonds



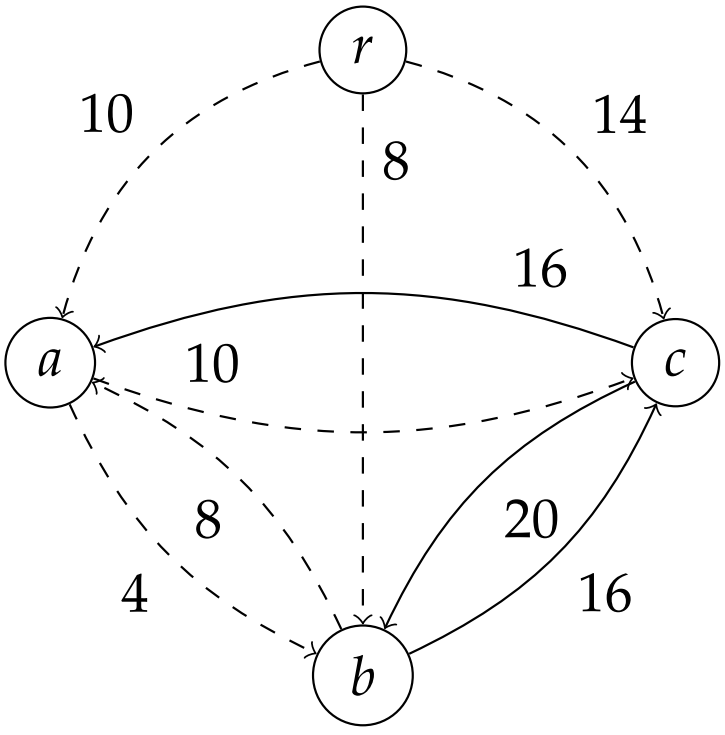
On a de la chance, le résultat est un arbre

Algorithme de Chu-Liu-Edmonds



On modifie le poids de l'arête $b \to c$, qui valait 12 et qui vaut maintenant 16.

Algorithme de Chu-Liu-Edmonds

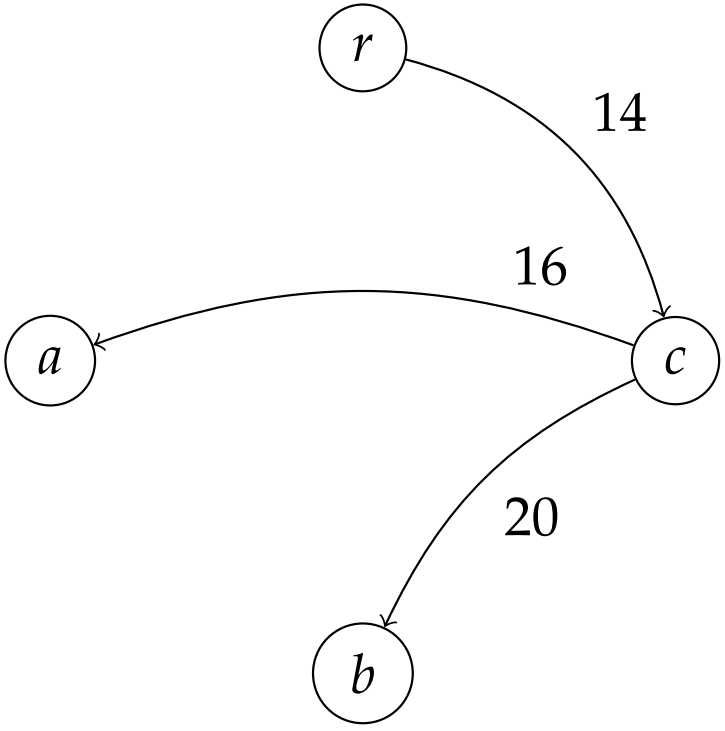


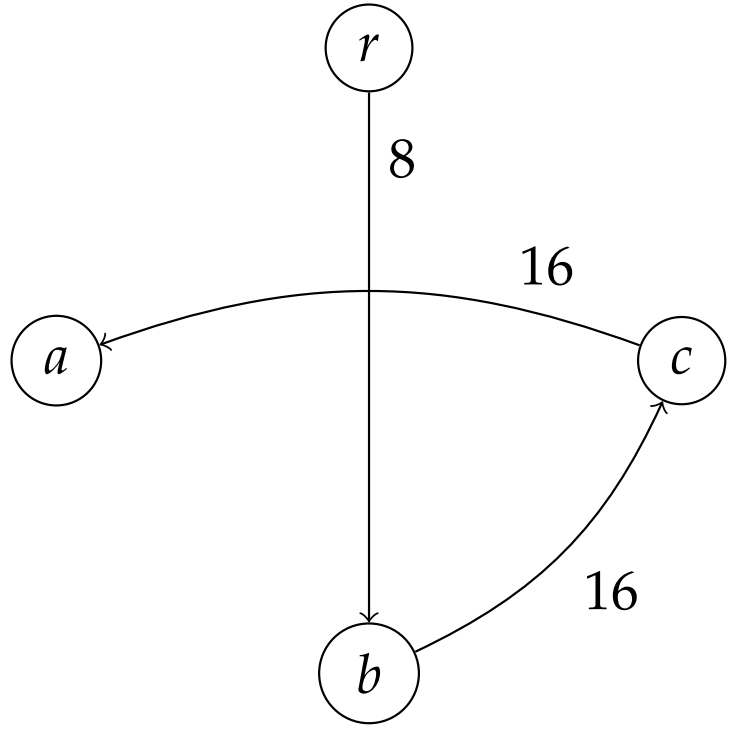
Pas de chance, il y a maintenant un cycle !

Elimination d'un cycle

- Pour casser le cycle, on enlève un arc de ce dernier et on rattache un des sommets du cycle à l'extérieur.
- Deux possibilités :
  - On rattache c a l'extérieur.
    - On a le choix de le rattacher à r ou a a
    - On le rattache à r car $r \xrightarrow{14} c$ alors que $a \xrightarrow{10} c$
    - $r \xrightarrow{14} c \xrightarrow{20} b$
    - le choix de $r \to c$ élimine $b \to c$
    - le score vaut alors 34
  - On rattache b a l'extérieur.
    - On a le choix de le rattacher à r ou a a
    - On le rattache à r car $r \xrightarrow{8} b$ alors que $a \xrightarrow{4} b$
    - $r \xrightarrow{8} b \xrightarrow{16} c$
    - le choix de $r \to b$ élimine $c \to b$
    - le score vaut alors 24

Deux solutions





- On choisit celle de gauche qui vaut 50 alors que celle de droite vaut 40
- Pour comparer les solutions, on compare :
  poids de l'arc ajouté - poids de l'arc éliminé
- Ici : $14 - 16 > 8 - 16$
- Mais : l'élimination d'un cycle peut en créer un nouveau !

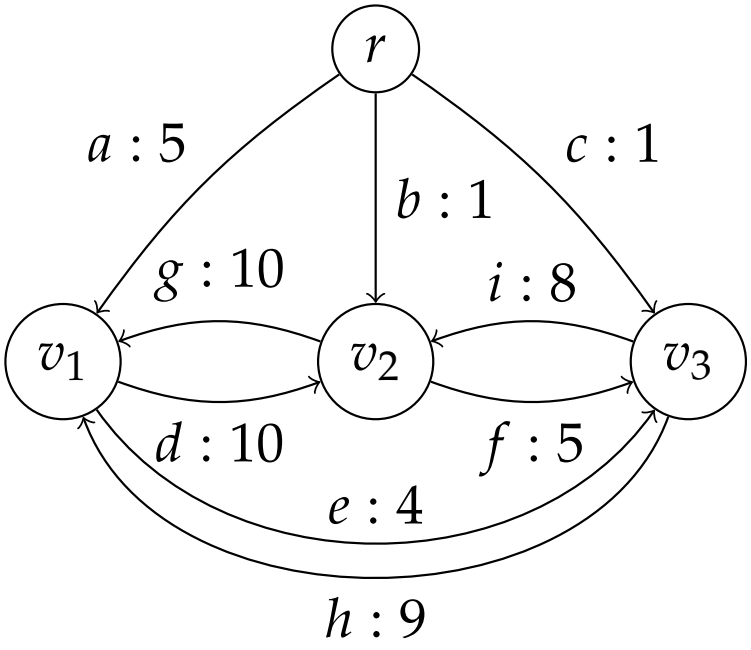
CLE : initialisation

- On part d'un graphe orienté pondéré
- Chaque sommet possède un arc entrant à partir de tous les autres sommets
- Sauf le sommet racine (r) qui ne possède pas d'arc entrant
- Le tableau MAE (Meilleur Arc Entrant) stocke, pour chaque sommet, l'arc entrant de meilleur poids
- Le tableau ELIMINE indique, pour chaque arc entrant d'un sommet d'un cycle, l'arc du cycle avec lequel il est incompatible

CLE

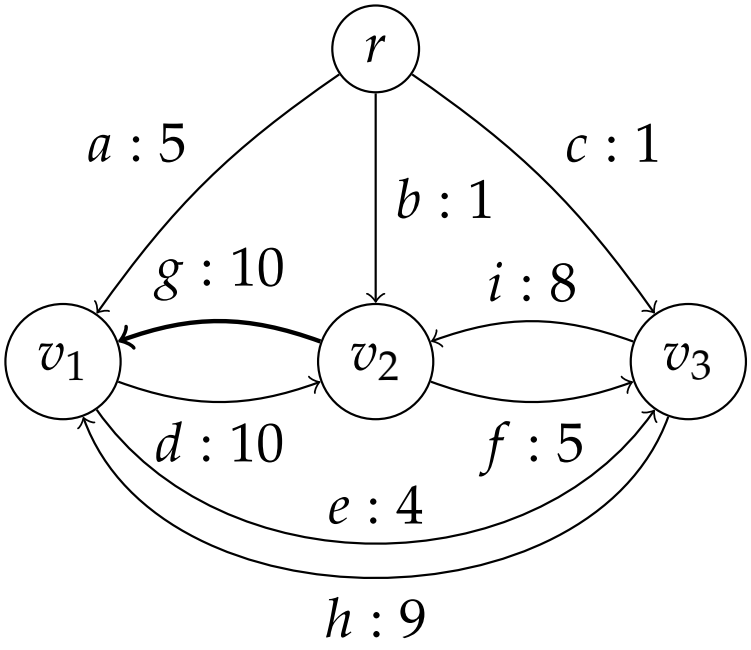
- Pour tout sommet $v \neq r$, $MAE[v] =$ arc entrant de meilleur poids.
- Si un cycle C a été créé :
  - **contracter** les sommets de C sous la forme d'un nouveau sommet v_C
  - les arcs sortants des sommets de C sortent maintenant de v_C
  - les arcs entrants des sommets de C entrent maintenant en v_C
  - pour tout sommet u de C et pour chaque arc a entrant en u depuis l'extérieur de C :
    - $ELIMINE[a] = MAE[u]$
    - $score[a] = score[a] - score[ELIMINE[a]]$
- répéter jusqu'à ce que tout sommet (sauf la racine) possède un arc entrant et qu'aucun cycle n'est créé.

Exemple



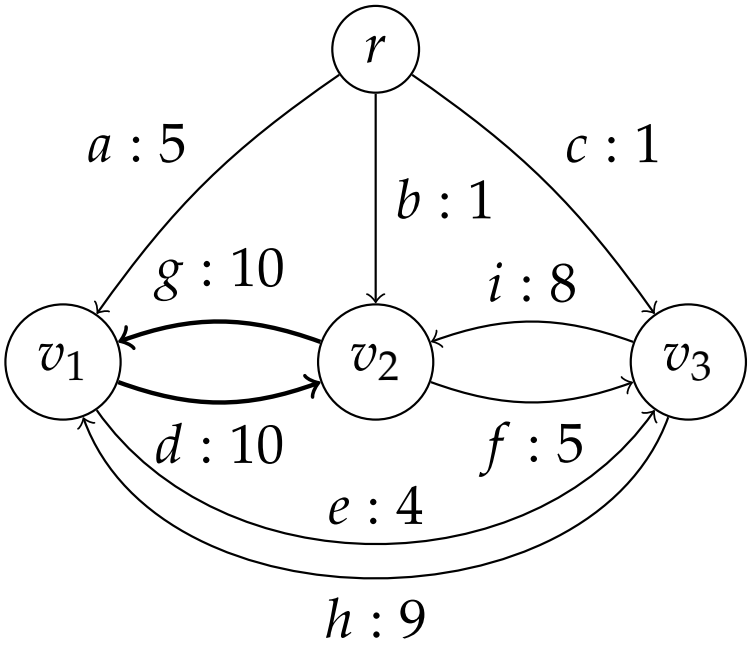
MAE		élimine	
v_1		a	
v_2		b	
v_3		c	
		d	
		e	
		f	
		g	
		h	
		i	

On traite v_1



MAE		élimine	
v_1	g	a	
v_2		b	
v_3		c	
		d	
		e	
		f	
		g	
		h	
		i	

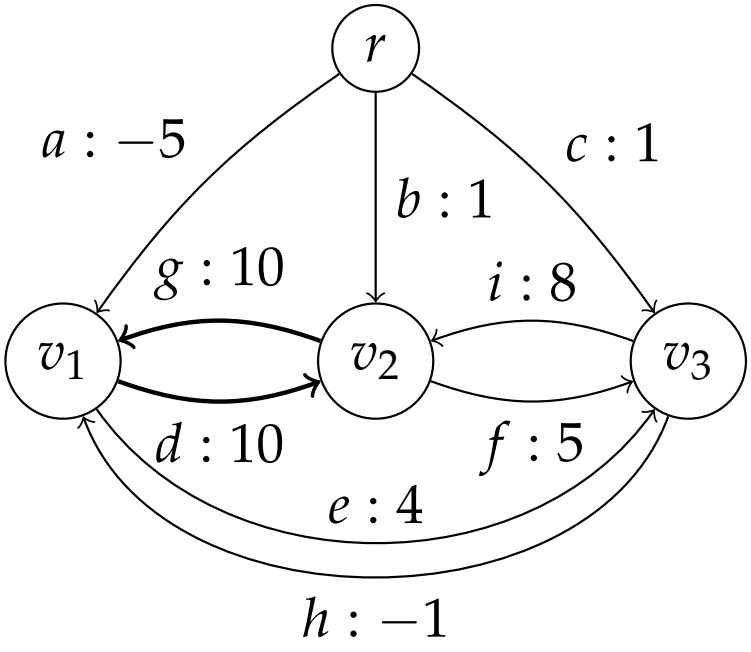
On traite v_2



MAE		élimine	
v_1	g	a	
v_2	d	b	
v_3		c	
		d	
		e	
		f	
		g	
		h	
		i	

- Création d'un cycle

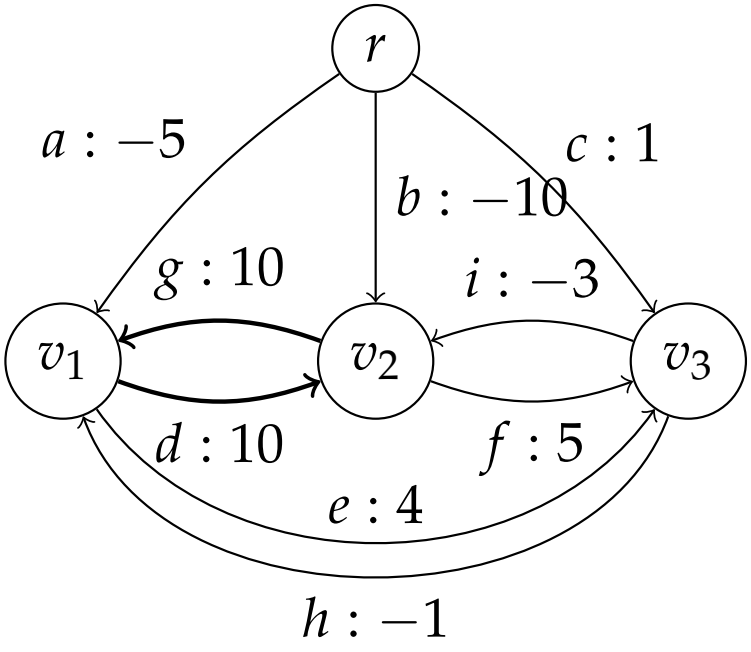
Contraction $\{v_1, v_2\}$



MAE		élimine	
v_1	g	a	g
v_2	d	b	
v_3		c	
		d	
		e	
		f	
		g	
		h	g
		i	

- Traitement des arcs entrant en v_1 de l'extérieur du cycle
- Mise à jour des poids des arcs

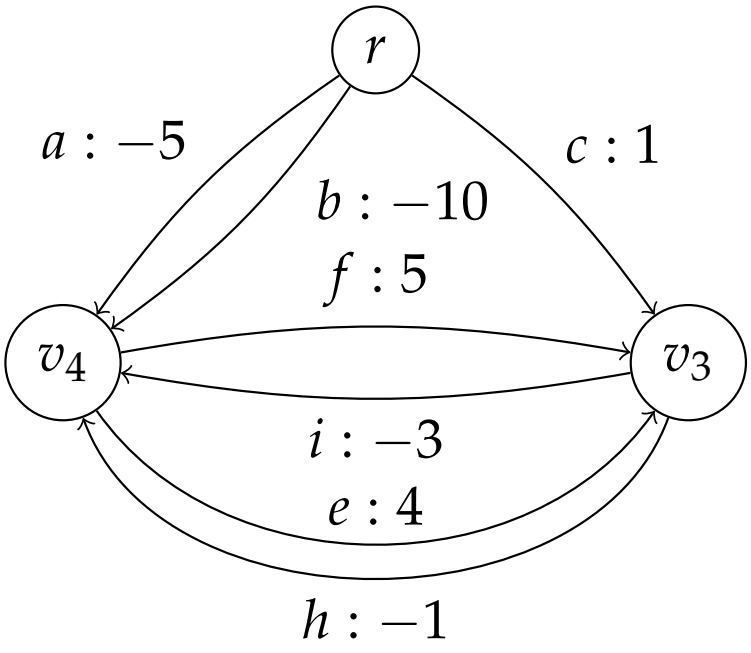
Contraction $\{v_1, v_2\}$



MAE		élimine	
v_1	g	a	g
v_2	d	b	d
v_3		c	
		d	
		e	
		f	
		g	
		h	g
		i	d

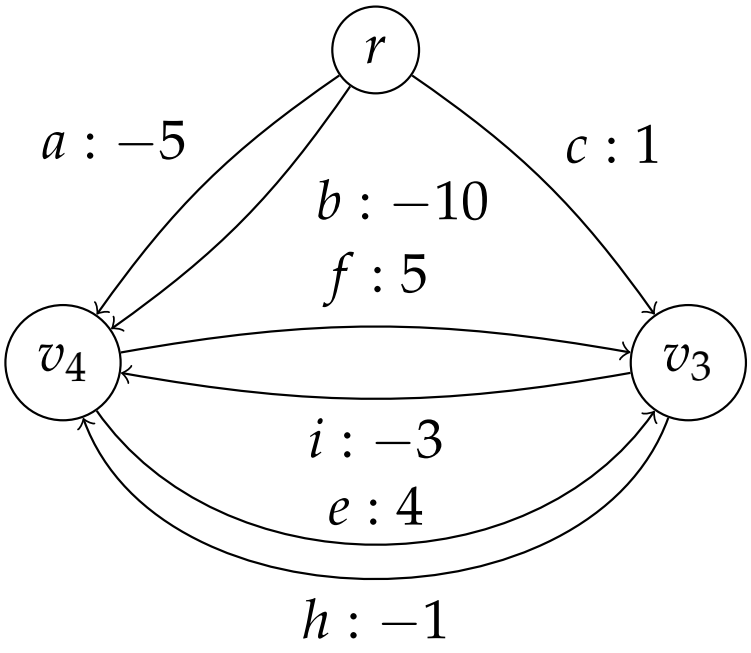
- Traitement des arcs entrant en v_2
- Mise à jour des poids des arcs

Contraction $\{v_1, v_2\}$



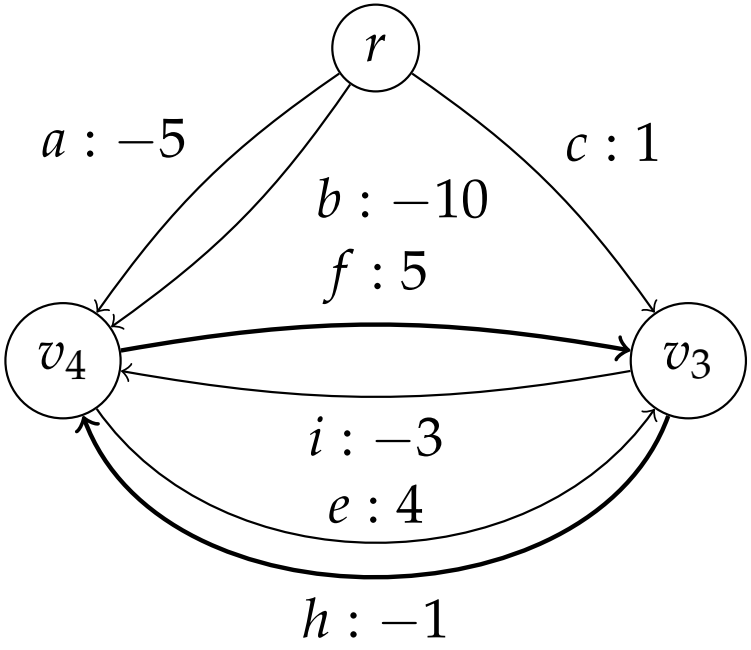
MAE		élimine	
v_1	g	a	g
v_2	d	b	d
v_3		c	
v_4		d	
		e	
		f	
		g	
		h	g
		i	d

On traite v_3



MAE		élimine	
v_1	g	a	g
v_2	d	b	d
v_3	f	c	
v_4		d	
		e	
		f	
		g	
		h	g
		i	d

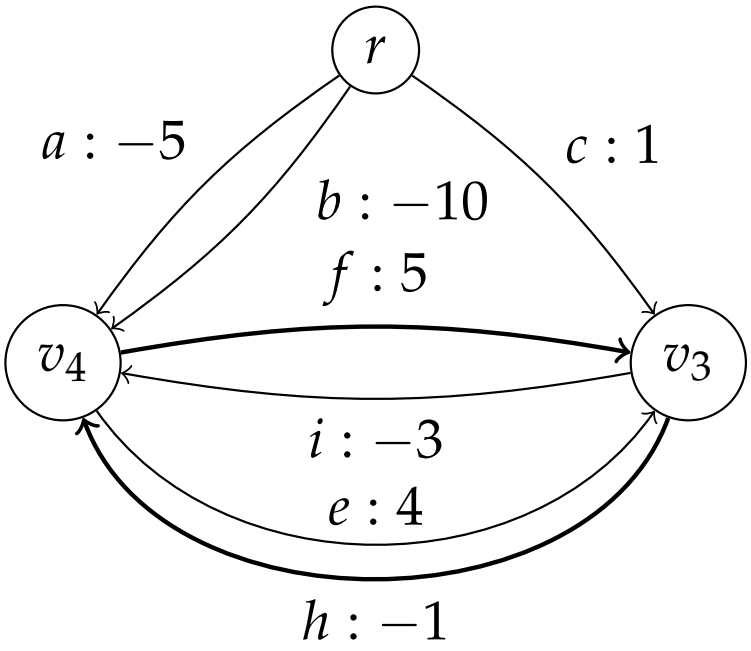
On traite v_4



MAE		élimine	
v_1	g	a	g
v_2	d	b	d
v_3	f	c	
v_4	h	d	
		e	
		f	
		g	
		h	g
		i	d

- création d'un cycle

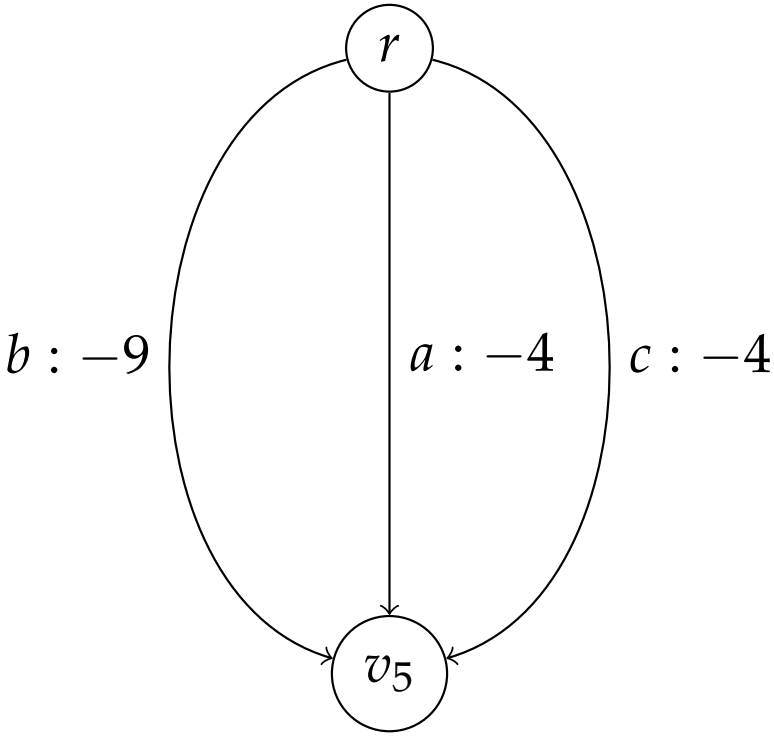
Contraction $\{v_3, v_4\}$



MAE		élimine		
v_1	g	a	g	h
v_2	d	b	d	h
v_3	f	c		f
v_4	h	d		
v_5		e		
		f		
		g		
		h	g	
		i	d	

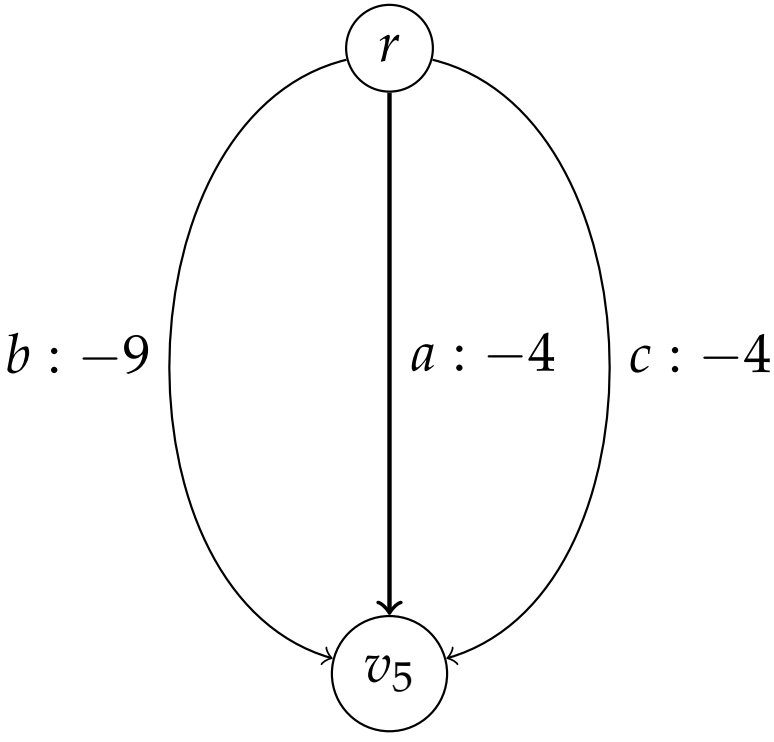
- création d'un cycle

Contraction $\{v_3, v_4\}$



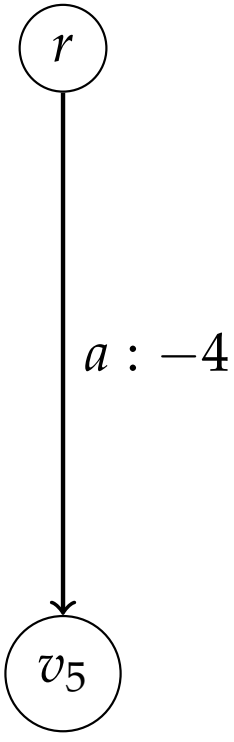
MAE		élimine		
v_1	g	a	g	h
v_2	d	b	d	h
v_3	f	c		f
v_4	h	d		
v_5		e		
		f		
		g		
		h	g	
		i	d	

On traite v_5



MAE		élimine		
v_1	g	a	g	h
v_2	d	b	d	h
v_3	f	c		f
v_4	h	d		
v_5	a	e		
		f		
		g		
		h	g	
		i	d	

Fin de la contraction

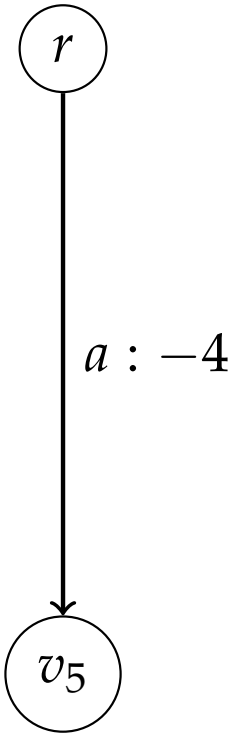


MAE		élimine		
v_1	g	a	g	h
v_2	d	b	d	h
v_3	f	c		f
v_4	h	d		
v_5	a	e		
		f		
		g		
		h	g	
		i	d	

Expansion

- A l'issue de la contraction, tout sommet s possède un meilleur arc entrant $e = MAE(s)$
- Le choix de e elimine un arc au sein d'un cycle, permettant de le casser
- Parcourir les arcs de MAE dans l'ordre inverse de leur ajout : (a, h, f, d, g)
- Pour tout arc e parcouru, choisir e et éliminer les arcs incompatibles avec e

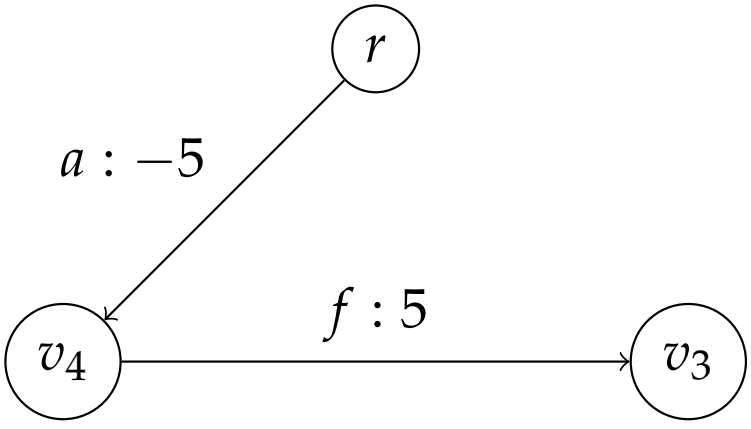
On traite v_5 (l'arc a)



MAE		élimine		
v_1	$\mathbf{a}(g)$	a	g	h
v_2	d	b	d	h
v_3	f	c		f
v_4	$\mathbf{a}(h)$	d		
v_5	a	e		
		f		
		g		
		h	g	
		i	d	

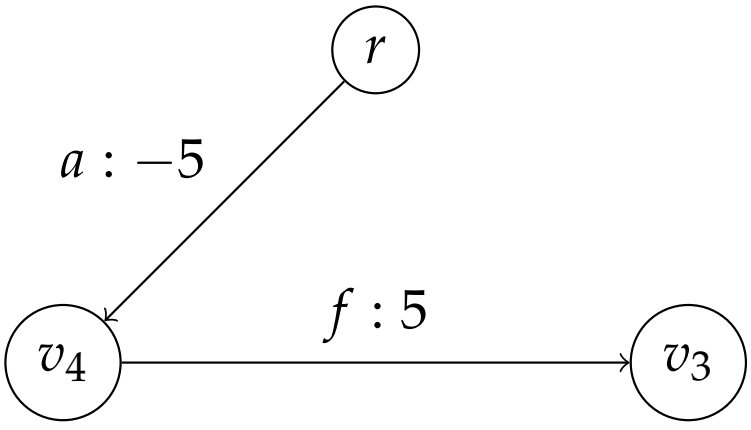
- On élimine g et h, qui sont incompatibles avec a
- On fait l'expansion de v_5

Expansion de v_5



MAE		élimine		
v_1	$\mathbf{a}(g)$	a	g	h
v_2	d	b	d	h
v_3	f	c		f
v_4	$\mathbf{a}(h)$	d		
v_5	a	e		
		f		
		g		
		h	g	
		i	d	

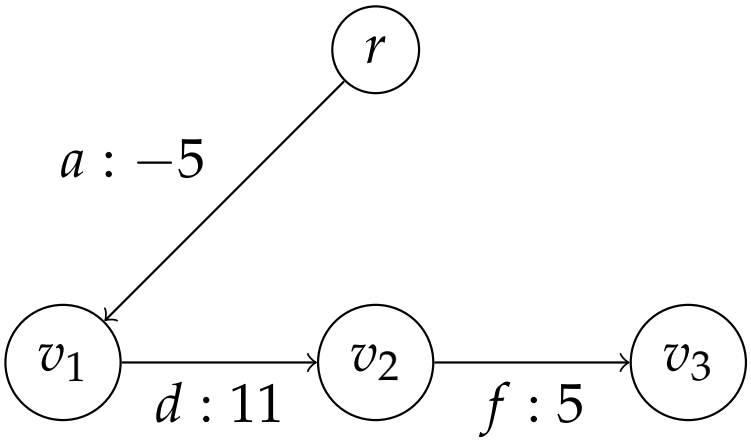
On traite v_4



MAE	
v_1	$\mathbf{a}(g)$
v_2	d
v_3	f
v_4	$\mathbf{a}(h)$
v_5	a

élimine		
a	g	h
b	d	h
c		f
d		
e		
f		
g		
h	g	
i	d	

Expansion de v_4



MAE		élimine		
v_1	$\mathbf{a}(g)$	a	g	h
v_2	d	b	d	h
v_3	f	c		f
v_4	$\mathbf{a}(h)$	d		
v_5	a	e		
		f		
		g		
		h	g	
		i	d	

Application de l'algorithme CLE au problème de l'analyse

Etant donné une phrase S :

1. On construit le multigraphe de dépendance $G(S)$
2. On réduit le multigraphe de dépendance à un graphe de dépendance $G'(S)$
3. On exécute l'algorithme CLE sur $G'(S)$

Graphe d'une phrase

- Etant donné :
  - une phrase $S = w_0 \dots w_n$,
  - un ensemble d'étiquettes syntaxiques $\mathcal{L} = \{l_1, \dots, l_m\}$
- on définit le *graphe de dépendance* de S, noté $G(S)$, le graphe orienté étiqueté (V_S, A_S), de la manière suivante :
  - $V_S = \{w_0, \dots, w_n\}$
  - $A_S = \{(w_i, l, w_j) \mid \forall w_i, w_j \in V_S \text{ et } l \in \mathcal{L}, \text{ avec } i \neq j \text{ et } j \neq 0\}$
- les poids des arcs sont définis de la façon suivante :
  $\lambda(w_i, l, w_j) = \lambda_1[w_i][w_j][l]$
- Notes :
  - $G(S)$ est un multigraphe : il existe entre deux sommets autant de dépendances qu'il existe d'étiquettes syntaxiques.
  - $G(S)$ est un graphe complet pour l'ensemble de sommets $V_S - \{w_0\}$
  - il existe un arc depuis w_0 vers tous les autres sommets.

Réduction d'un multigraphe à un graphe

- L'algorithme CLE est défini sur un graphe orienté
- Or le graphe $G(S)$ est un *multigraphe* : il existe entre deux sommets autant de dépendances qu'il existe d'étiquettes syntaxiques.
- Il est possible de réduire le multigraphe en un graphe orienté $G'(S) = (V'_S, A'_S)$ de la manière suivante
  - $V'_S = V_S$
  - $A'_S = \{(w_i, w_j) \mid \forall w_i, w_j \in V_S, \text{ avec } i \neq j \text{ et } j \neq 0\}$
- $G'(S)$ est aussi un graphe complet, mais avec un seul arc entre deux sommets
- On définit les poids des arcs de A'_S de la façon suivante :

$$\lambda(w_i, w_j) = \max_{l \in \mathcal{L}} \lambda(w_i, l, w_j)$$

Exercice

On souhaite analyser le groupe prépositionnel *sur la route* avec l'algorithme de Chu-Liu-Edmonds. Les poids des dépendances sont représentés dans la matrice ci-dessous, où les lignes correspondent aux gouverneurs et les colonnes aux dépendants (la dépendance (ROOT, sur), par exemple, vaut -2.).

	ROOT	sur	la	route
ROOT	0	-2	0	1
sur	0	0	0	2
la	0	0	0	0
route	0	2	1	0

1. Dessinez le graphe complet sur lequel se fera la recherche de l'arbre couvrant.
2. Dessinez tous les arbres couvrants et calculez leur poids.
3. Retrouvez l'arbre optimal à l'aide de l'algorithme CLE.