Rappels sur le modèle relationnel

Théorème: SQL=CALC=ALG

Théorème: SFW=CQ=SRPJ

Plusieurs façons d'exprimer la même requête dans SQL.

Si q est dans SQL, chaque q' de ALG équivalent à q peut-être vu comme un plan d'évaluation de q.

Plusieurs façons d'exprimer la même requête dans ALG. Donc plusieurs plans d'évaluation possibles.

$\Rightarrow$ comment trouver le meilleur?

Aujourd'hui:

Complexité pour évaluer une requête.

Complexité de l'optimisation.

Quelle est la difficulté d'évaluer une requête?

On cherche en pratique la complexité du problème suivant:

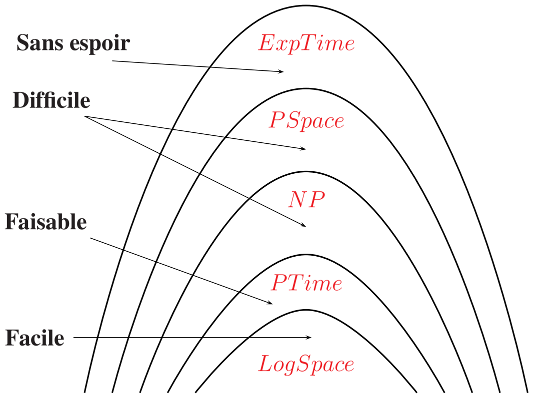
Entrée:	Une instance I, une requête q de L
Sortie:	q(I)=$\{\bar{c} \mid I \models_f q(\bar{x}), \quad f(\bar{x}) = \bar{c}\}$

On va regarder le problème décisionnel associé:

Eval(L)

Entrée:	Une instance I, une requête q de L, $\bar{c}$
Sortie:	$\bar{c} \in q(I)$?

Rappel sur la complexité



CALC vs. ALG vs. SQL

Eval(L)

Entrée:	Une instance I, une requête q de L, $\bar{c}$
Sortie:	$\bar{c} \in q(I)$?

Théorème: SQL=CALC=ALG
Le passage de l'un à l'autre se fait en temps linéaire!

Donc modulo une constante:

Eval(CALC)= Eval(ALG) = Eval(SQL)

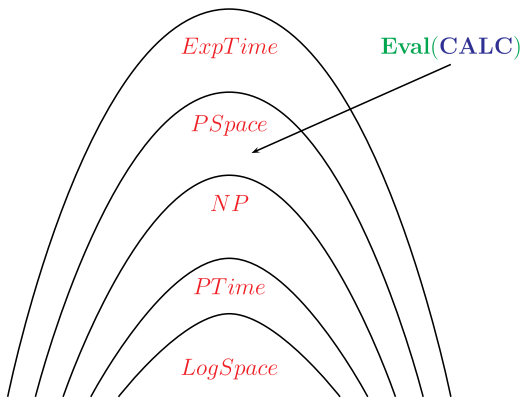
Évaluation de CALC: c'est très complexe!!!

Théorème: **Eval(CALC)** est PSPACE-complet

Preuve: on montre que **Eval(CALC)** est dans PSPACE et dans EXPTIME par induction sur la formule.

Pour la preuve de la borne inférieure vous n'avez pas encore les connaissances nécessaires. Retenez juste que parmi les problèmes PSPACE **Eval(CALC)** est ce qu'on peut faire de plus dur. Pour les motivés voir:

Moshe Vardi, "The complexity of relational query languages" Symposium on the Theory of Computing (STOC) 1982.



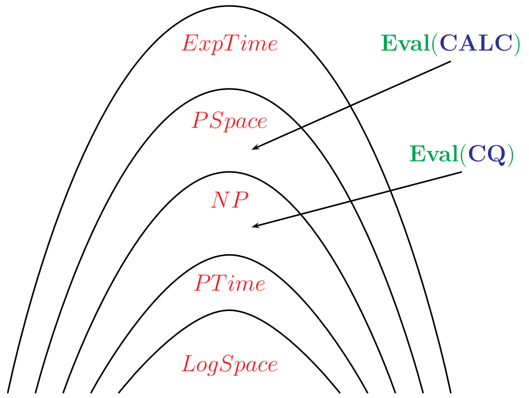
Même pour CQ il semble impossible d'éviter un temps exponentiel!!!

Théorème: **Eval(CQ)** est NP-complet

Preuve: On montre que c'est équivalent au problème suivant:

HOM

Entrée:	Deux graphes G et G'
Sortie:	Existe-t-il un homomorphisme de G dans G'?



Un graphe G est la donnée d'un ensemble V de noeuds et d'un ensemble E d'arêtes.

Un homomorphisme h d'un graphe G=(V,E) dans un graphe G'=(V',E') est une fonction de V dans V' telle que pour tout x,y dans V on a:

$$E(x,y) \quad \Rightarrow \quad E'(h(x),h(y))$$

Théorème: **Eval(CQ)** est NP-complet

Preuve: On montre que c'est équivalent au problème suivant:

HOM

Entrée:	Deux graphes G et G'
Sortie:	Existe-t-il un homomorphisme de G dans G'?

Rappel: toute formule de CQ peut s'écrire sous la forme:

$$\exists\, y_1 \cdots y_n \bigwedge_i \theta_i(\bar{x}, \bar{y})$$

Définition: Soit une requête conjonctive q en forme normale, on appelle graphe de q le graphe G_q défini par:

Les noeuds de G_q sont les variables de q ($\bar{x}$ et $\bar{y}$).

Les arêtes de G_q sont les paires (u,v) telles que u et v apparaissent dans le même θ_i.

Théorème: **Eval(CQ)** est NP-complet

Preuve: On suppose que le schéma de la base ne contient qu'une relation d'arité 2 et on montre que **Eval(CQ)** est équivalent au problème suivant:

HOM

Entrée:	Deux graphes G et G'
Sortie:	Existe-t-il un homomorphisme de G dans G'?

En effet on a: $\bar{c} \in$ q(I) ssi il existe un homomorphisme h de G_q dans I tel que h($\bar{x}$)=$\bar{c}$

Comment peut-on expliquer que cela marche en pratique?

Réponse 1: la mesure de complexité n'est pas adaptée à ce qui se passe en pratique.

Réponse 2: on mesure la complexité de requêtes qui ne sont pas posées en pratique.

Réponse 1: complexité sur les données

En pratique: $|q| << |I|$!!!

On va donc supposer que la taille de la requête est une constante!

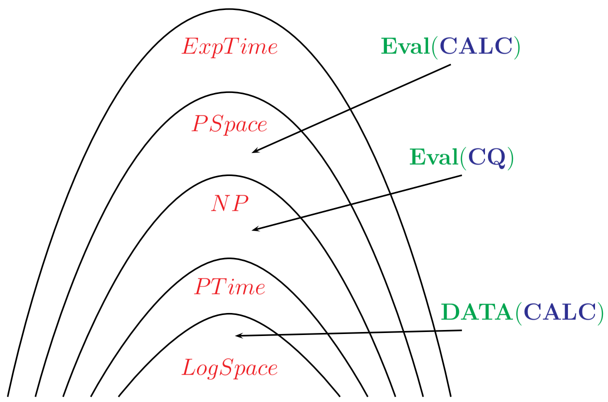
DATA(q)

Entrée:	Une instance I, $\bar{c}$
Sortie:	$\bar{c} \in q(I)$?

Théorème: ∀ q ∈ CALC, **DATA(q)** est dans LOGSPACE

Preuve: induction sur la formule.

On dit que **DATA(CALC)** est dans LOGSPACE.



Réponse 2 : acyclicité

Un graphe G est dit acyclique si il ne contient pas de cycle.

Une requête conjonctive q est dite acyclique si G_q est acyclique.

Soit ACQ l'ensemble des requêtes conjonctives acycliques.

Théorème: **Eval(ACQ)** est dans PTIME

Preuve: il faut faire les jointures dans le bon ordre! Trop dur pour ce cours. Voir:

Michalis Yannakakis "Algorithms for acyclic database schemes" Proceedings of Very Large Databases (VLDB), 1981.

Comment peut-on expliquer que cela marche en pratique?

Réponse 1: la mesure de complexité n'est pas adaptée à ce qui se passe en pratique:

si on suppose q << I, **DATA**(CQ) est dans LOGSPACE.

Réponse 2: on mesure la complexité de requêtes qui ne sont pas posées en pratique:

En pratique on utilise des requêtes acycliques et **Eval(ACQ)** est PTIME.

Si q est dans SQL, chaque q' de ALG équivalent à q peut-être vu comme un plan d'évaluation de q.

Plusieurs façons d'exprimer la même requête dans ALG. Donc plusieurs plans d'évaluation possibles.

⇒ comment trouver le meilleur?

Tentative naïve: on les essaye tous!

Problème 1: comment les essayer tous?

Problème 2: comment comparer leur efficacité?

On verra plus tard comment résoudre le problème 2.

Essayer tous les algos possibles

Pour pouvoir faire cela il faut pouvoir résoudre le problème suivant:

On dit que q ≡ q' si ∀ I q(I)=q'(I)

EQ

Entrée:	Une requête q de CALC, une requête q' de ALG.
Sortie:	q ≡ q'?

Théorème: **EQ** est indécidable!

Preuve: trop dur pour ce cours.

Même pour des requêtes conjonctives c'est difficile:

ECQ

Entrée:	Une requête q de CQ, une requête q' de SFW.
Sortie:	q ≡ q'?

Théorème: **ECQ** est NP-complet

Preuve: on retrouve **HOM**. q ≡ q' ssi il existe un homomorphisme de G_q dans $G_{q'}$ et vice-versa.

En particulier on a: $q \subseteq q'$ ssi il existe un homomorphisme de $G_{q'}$ dans G_q envoyant les variables libres de q' dans celles de q.

Conclusion

Le Problème 1, qui consiste à essayer tous les algorithmes possibles est insurmontable. Même pour des requêtes conjonctives on passerait plus de temps à optimiser qu'à évaluer.

⇒ On utilise des heuristiques

1. "Pousser les sélections"
2. Ordre sur les jointures

Exercice

On considère les q_i données par :

$$q_0(x,y) : R(x,y)$$
$$q_1(x,y) : \exists x_1, y_1 \quad R(x,y_1) \wedge R(x_1,y_1) \wedge R(x_1,y)$$
$$q_2(x,y) : \exists x_1, x_2, y_1, y_2 \ R(x,y_1) \wedge R(x_1,y_1) \wedge R(x_1,y_2) \wedge R(x_2,y_2) \wedge R(x_2,y)$$
$$q_3(x,y) : \exists x_1, y_1 \ R(x,y_1) \wedge R(x_1,y)$$

Est-ce que $q_i \subseteq q_j$ pour i,j in $[0..4]$ (justifier)?