

Cours 6: Branch and bound, complexité paramétrique

- Backtracking, branch and bound
- Vertex Cover et la complexité paramétrique
- Réduction au noyau (kernelisation)

Exploration arborescente et *backtracking*

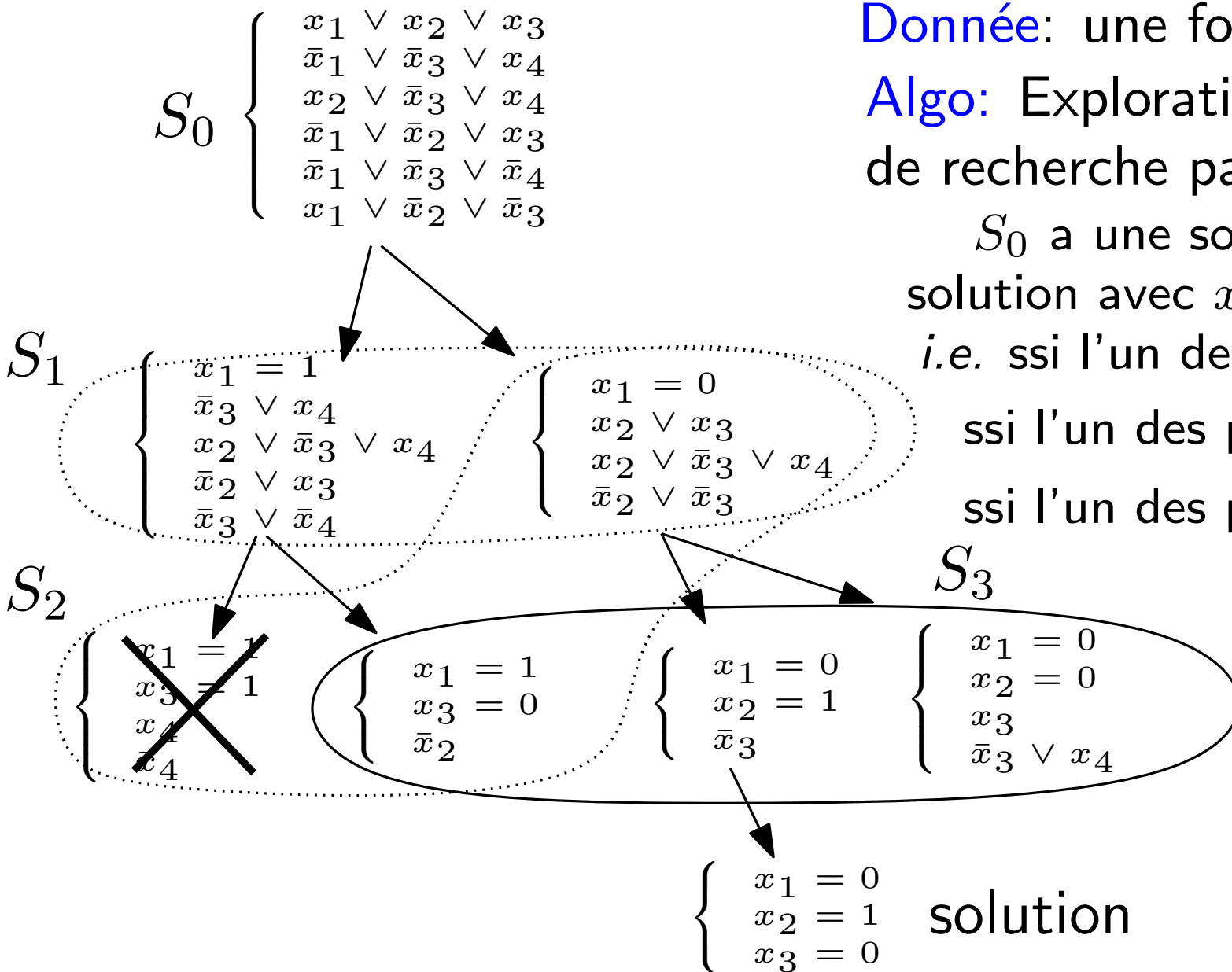
Donnée: une formule SAT

Algo: Exploration de l'espace de recherche par instantiation

S_0 a une solution ssi il y a une solution avec $x_1 = 1$ ou avec $x_1 = 0$
i.e. ssi l'un des pbs de S_1 en a une.

ssi l'un des pbs de S_2 en a une.

ssi l'un des pbs de S_3 en a une.



Exploration arborescente et *backtracking*

Donnée: un problème de décision à résoudre (par exemple une formule SAT)

Algo pour 3-SAT:

- dans S on prend une formule avec une clause à 1 ou 2 variables, ou sinon une formule avec un petit nombre de clauses
- on remplace la formule courante par 2 formules, obtenue en mettant l'une des variables à VRAI ou FAUX (choisir la variable "la plus" contrainte) et on résoud récursivement ces 2 problèmes contraints.

Algorithme générique:

- à chaque étape on a un ensemble S de pbs, il faut en résoudre un
- choisir un des problèmes P de S , le décomposer en union de plusieurs problèmes P_1, \dots, P_k tels que résoudre P est équivalent à résoudre les P_i
 - si l'un des problèmes P_i a une solution facile, la calculer et s'arrêter
 - éliminer les P_i trivialement sans solution et remplacer P par les autres
- recommencer jusqu'à ce que S soit vide ou qu'on ait trouvé une solution.

Remarque: la façon habituelle de générer les P_i est de partitionner l'espace de recherche en imposant des contraintes aux solutions.

⇒ programmation par contrainte

Exploration arborescente et *backtracking*

Donnée: un problème de décision à résoudre (par exemple une formule SAT)

Algo pour 3-SAT:

- dans S on prend une formule avec une clause à 1 ou 2 variables, ou sinon une formule avec un petit nombre de clauses
- on remplace la formule courante par 2 formules, obtenue en mettant l'une des variables à VRAI ou FAUX (choisir la variable "la plus" contrainte) et on résoud récursivement ces 2 problèmes contraints.

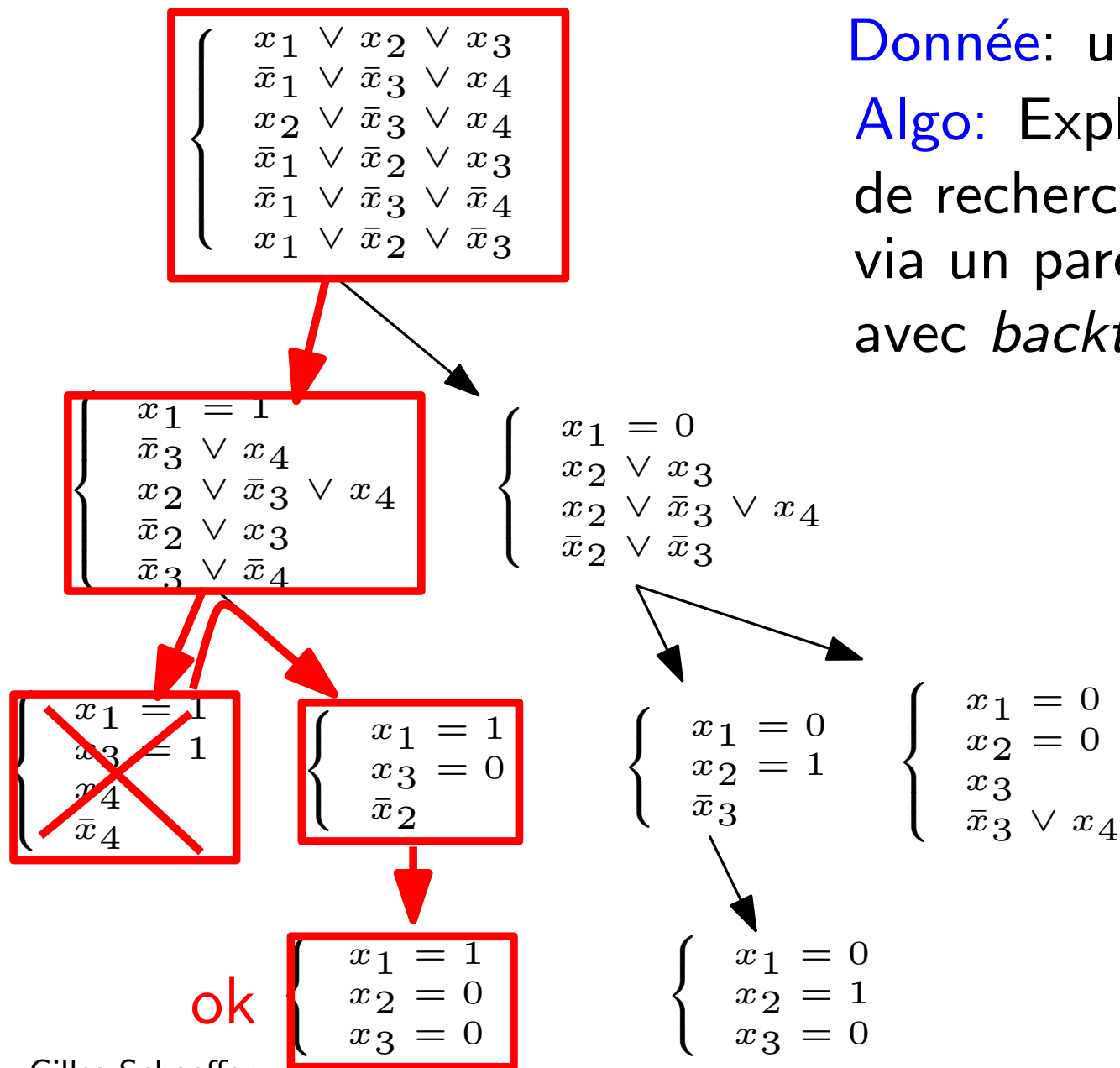
Backtracking, ou parcours en profondeur:

- le nombre de sous-problèmes à profondeur k est exponentiel en k
- pour éviter de devoir stocker un nombre exponentiel de sous-problèmes on peut effectuer un parcours en profondeur de l'arbre d'exploration
 - on descend dans l'arbre en imposant des contraintes à la solution (par exemple, pour SAT, décider la valeur d'une variable)
 - on remonte en revenant sur les dernières décisions prises, lorsqu'on s'aperçoit que les contraintes posées conduisent à l'absence de solution.

Exploration arborescente et *backtracking*

Donnée: une formule SAT

Algo: Exploration de l'espace de recherche par instantiation via un parcours en profondeur avec *backtracking*



backtracking pour VERTEX COVER

Donnée: un graphe $G = (V, E)$, $|V| = n$, $|E| = m$.

Problème: un ensemble de sommets couvrants de cardinalité minimale

Stratégie d'exploration ? plusieurs possibilités par instantiation

- A profondeur k de l'arbre on choisit le k ème sommet de la couverture parmi les sommets incidents aux arêtes non couvertes par les $k - 1$ premiers sommets choisis.

branchement entre n possibilités au premier niveau,
au plus $n - 1$ possibilités au second niveau...

L'ordre dans lequel on explore les branches a bcp d'influence:
commencer par les sommets de plus haut ou de plus bas degré ?

- A profondeur k de l'arbre on choisit de couvrir l'une ou l'autre des deux extrémités de la première arête non couverte (ordre arbitraire)
branchement entre 2 possibilités à chaque niveau:
arbre d'exploration binaire.

Optimisation: pour minimiser faut il explorer complètement l'arbre ?

Séparation-évaluation ou *branch and bound*

Donnée: un problème d'optimisation sous contraintes

Branch On constuit un arbre d'exploration des solutions faisables
c'est déjà ce qu'on fait avec le backtracking

Bound: Dès qu'on a une solution, ne plus explorer les sous-arbres dont on sait dire *a priori* qu'ils ne donneront pas mieux !

⇒ il faut savoir borner la valeur des solutions dans un sous-arbre avant de l'avoir exploré.

VERTEX COVER: une fois qu'on a un couvrant à k sommets, on n'explore plus au delà de la profondeur $k - 1$:

- stratégie par sommets: arbre d'exploration restant d'au plus n^k nœuds
- stratégie par arêtes: arbre d'exploration restant d'au plus 2^k nœuds

Cours 6: Complexité paramétrique

- Backtracking, branch and bound
- Vertex Cover et la complexité paramétrique
- Réduction au noyau (kernelisation)

Complexité paramétrique de VERTEX COVER

Donnée: un graphe $G = (V, E)$ et un entier k

Problème: un ensemble de sommets couvrants de cardinalité au plus k
L'objectif donne une borne *a priori* sur la profondeur d'exploration,
on explore que jusqu'à profondeur k

- Stratégie par sommets: complexité en $n^k \times O(n) = O(n^{k+1})$ **insuffisant**
- Stratégie par arêtes: complexité en $2^k \times O(n) = O(2^k n)$ **ok**

À k fixé le second algorithme est **linéaire** en la taille du graphe !

On dit que le **problème paramétré k -Vertex Cover** est

polynomial à paramètre fixé (FPT, fixed parameter tractable)

car il admet un algorithme de complexité $\leq f(k) \times n^{O(1)}$.

(f fonction qcq, k paramètre, n taille, $O(1)$ indépt de k et n .)

Méthode de l'arbre d'exploration borné: dans un algo branch and bound avec branchement de degré fini, si la forme de la fonction d'évaluation implique une borne sur la profondeur d'exploration, le problème est FPT.

Problème paramétré et complexité paramétrique

Problème paramétré = problème + paramètre.

On dit qu'un problème paramétré est

polynomial à paramètre fixé (FPT, fixed parameter tractable)
s'il existe une fonction f et un algorithme de complexité $\leq f(k) \times n^{O(1)}$
sur les instances de taille n et paramètre k .

- si le problème non paramétré est NP-complet, la fonction f est exponentielle en k (sauf si $P=NP$)
- Il peut y avoir plusieurs paramètres d'intérêt pour un même problème
Exemple: **MIN VERTEX COVER** pour les graphes d'excès k est FPT
($G = (V, E)$ a excès k si G connexe et $|E| = |V| - 1 + k$,
 G a excès 0 $\Leftrightarrow G$ est un arbre)

preuve?

Problème paramétré et complexité paramétrique

Problème paramétré = problème + paramètre.

On dit qu'un problème paramétré est

polynomial à paramètre fixé (FPT, fixed parameter tractable)

s'il existe une fonction f et un algorithme de complexité $\leq f(k) \times n^{O(1)}$ sur les instances de taille n et paramètre k .

Tous les problèmes NP-complet ne sont pas connus FPT:

ainsi le meilleur algo connu pour k DOMINATING SET est essentiellement l'exploration des $\Theta(n^k)$ ensembles de k sommets.

Le gain en complexité des algorithmes polynomiaux à paramètre fixé est significatif en pratique:

$\frac{n^{k+1}}{2^k n}$	$n = 50$	$n = 100$	$n = 150$
$k = 2$	625	2.500	5.625
$k = 3$	15.625	125.000	421.875
$k = 5$	390.625	6.250.000	31.640.625
$k = 10$	1.9×10^{12}	9.8×10^{14}	3.7×10^{16}
$k = 20$	1.8×10^{26}	9.5×10^{31}	2.1×10^{35}

Problème paramétré et complexité paramétrique

Problème paramétré = problème + paramètre.

On dit qu'un problème paramétré est

polynomial à paramètre fixé (FPT, fixed parameter tractable)

s'il existe une fonction f et un algorithme de complexité $\leq f(k) \times n^{O(1)}$ sur les instances de taille n et paramètre k .

Tous les problèmes NP-complet ne sont pas connus FPT:

ainsi le meilleur algo connu pour k DOMINATING SET est essentiellement l'exploration des $\Theta(n^k)$ ensembles de k sommets.

Il est aussi utile en pratique de se battre sur le comportement exponentiel: pour VERTEX COVER, jusqu'à $f(k) = 1.28^k$

k	$f(k) = 2^k$	$f(k) = 1.49^k$	$f(k) = 1.32^k$	$f(k) = 1.28^k$
10	1000	54	16	12
20	10^6	3000	258	140
30	10^9	10^5	4000	1500
40	10^{12}	10^7	10^4	10^4
50	10^{15}	10^9	10^6	10^5
75	10^{22}	10^{13}	10^9	10^8
100	10^{30}	10^{17}	10^{12}	10^{10}

Cours 6: Complexité paramétrique

- Backtracking, branch and bound
- Vertex Cover et la complexité paramétrique
- Réduction au noyau (kernelisation)

Noyau et kernelisation

Donnée: un problème \mathcal{P} de décision paramétré par k .

Problème: Donner un algorithme qui à une instance (X, k) de \mathcal{P} de taille n associe en temps polynomial en n une instance $(N(X), k')$ de \mathcal{P} de taille au plus $g(k)$ (où g est une fonction indépt de n) appelée **noyau** de X , telle que la réponse à $(N(X), k')$ soit identique à la réponse à (X, k) .

Si on trouve un tel algorithme alors on sait résoudre le problème de départ en temps $f(g(k)) + n^{O(1)}$ où f est le coût d'un algorithme de résolution du problème non paramétré et $n^{O(1)}$ correspond au coût de la réduction.

Théorème: Tout problème FPT admet une réduction à un noyau (*a priori* de taille exponentielle en k).

Noyau et kernelisation

Théorème: Tout problème FPT admet une réduction à un noyau (*a priori* de taille exponentielle en k).

Preuve: supposons qu'on ait un algo en $f(k) \times n^\alpha$, on va voir qu'on peut trouver un noyau de taille au plus $f(k)$ en temps $O(n^{\alpha+1})$

- si on a une instance avec $f(k) \leq n$: on utilise l'algo pour décider en temps $f(k) \times n^\alpha = O(n^{\alpha+1})$ et on renvoie un noyau prédéfini donnant la bonne réponse.
- sinon $f(k) > n$, *i.e.* on a déjà une instance de taille au plus $f(k)$.

La question est de trouver des noyaux de taille polynomiale, voire linéaire

Noyau pour VERTEX COVER

Donnée: un graphe $G = (V, E)$ et un entier k .

Problème: existe-t-il un ensemble de k sommets couvrants ?

Supprimer les boucles et arêtes multiples

Remarquer qu'un sommet de degré k est nécessairement dans le couvrant.

Supprimer ces sommets itérativement. L'objectif k diminue en conséquence.

Le graphe restant n'a que des sommets de degré $\leq k$.

Il a donc $\leq k^2$ arêtes, sinon pas couvert par k sommets de deg $\leq k$.

Il reste donc au plus $2k^2$ sommets non isolés:

on a extrait un **noyau de taille $O(k^2)$** en temps $O(n + m)$.

Avec la stratégie des arêtes on obtient un algo en $O(n + m + 2^k k^2)$.

Noyau et approximation: VERTEX COVER

Donnée: un graphe $G = (V, E)$ et un entier k .

Problème: existe-t-il un ensemble de k sommets couvrants ?

Supposons qu'on ait une réduction avec noyaux linéaires (de taille $\leq c \cdot k$).

Supposons de plus que cette réduction construise un noyau N qui est un sous-graphe du graphe G de départ et que la réduction fixe le status des sommets hors du noyau.

Lors de la réduction, deux cas peuvent se présenter:

- soit on constate au cours de la réduction qu'il faut plus de k sommets;
- soit on place $i \leq k$ sommets dans la couverture au cours de la réduction et on obtient un graphe réduit restant avec $j \leq c(k - i)$ sommets;

On peut donc dire soit qu'il n'existe pas de couvertures à k sommets, soit produire une couverture à $\leq ck$ sommets, en temps polynomial.

Par dychotomie à partir de $k = n$ on trouve k tel qu'on ait une couverture de taille ck et on sache qu'il n'existe pas de couverture de taille $k - 1$:

on a construit une approximation à un facteur c en temps polynomial.

Algorithmes d'approximation et complexité paramétrée

Un problème d'optimisation: trouver S^* tq $v(S^*) \leq v(S)$ pour tout S .

Le problème de décision associé: existe-t-il S tq $v(S) < k$?

\Rightarrow naturellement paramétré par k

Théorème: si le problème d'optimisation admet un schéma d'approximation polynomial efficace (il trouve une solution ε approchée en $f(1/\varepsilon) \times n^c$) alors le problème de décision associé est FPT (de complexité $f(2k) \times n^c$)

même idée que pour montrer l'inapproximabilité: utiliser le fait que k entier

pour savoir s'il existe S avec $v(S) < k$, on cherche une solution $\varepsilon = 1/(2k)$ approchée: en temps $f(2k) \times n^c$ on trouve S avec $v(S) < (1 + \frac{1}{2k})v(S^*)$

- si $v(S) < k$ alors la réponse au pb de décision est **oui**.
- si $v(S) \geq k$ alors $v(S^*) > v(S) \cdot (1 + \frac{1}{2k}) > k - 1$ donc $v(S^*) \geq k$, la réponse au problème de décision est **non**.

Paramètres explicites, paramètres structurels

Dans la "vie courante", certains algos *a priori* exponentiels dans le pire cas ont des comportements expérimentaux polynomiaux

- inférence de type en CAML: un problème EXP-complet...

mais ça marche en pratique car le problème est FPT par rapport à la profondeur d'imbrication des types.

dans ce cas la complexité paramétrique fournit une explication plus convaincante que ne pourrait le faire la complexité moyenne par exemple

Il y a des cas où les paramètres naturels ne donnent pas de résultat, comme DOMINATING SET: non FPT par rapport à la taille du dominant

mais il le devient pour de "bonnes" sous-classes de graphes: FPT pour les graphes planaires (ou de genre fixé, ou VSLI à k couches)

ou lorsqu'on paramétrise le problème par un invariant structurel du graphes (comme la largeur arborescente, cf la semaine prochaine), plutôt que par la taille du dominant.

Cours 6: Complexité paramétrique

- Backtracking, branch and bound
- Vertex Cover et la complexité paramétrique
- Réduction au noyau (kernelisation)

Retenir: Heuristique branch-and-bound et problème FPT

La kernelisation donne des algo pratiques pour k petits !

La suite la semaine prochaine...