Cours 8: Complexité paramétrique

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

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.

$$\begin{cases} x_1 \vee x_2 \vee x_3 \\ \bar{x}_1 \vee \bar{x}_3 \vee x_4 \\ x_2 \vee \bar{x}_3 \vee x_4 \\ \bar{x}_1 \vee \bar{x}_2 \vee x_3 \\ x_1 \vee \bar{x}_2 \vee \bar{x}_3 \end{cases} \qquad \begin{cases} x_1 \vee \bar{x}_2 \\ x_1 \vee x_4 \\ \bar{x}_1 \vee \bar{x}_3 \\ \bar{x}_1 \vee x_5 \\ x_2 \vee \bar{x}_4 \\ x_3 \vee \bar{x}_5 \end{cases}$$

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 $P_1, \ldots, P_k$ 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'arêter
  – 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.

$\Rightarrow$ 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.

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 "instanciation"

- 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 !

$\Rightarrow$ 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 8: Complexité paramétrique

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

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'intéret 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	$\underline{10^9}$	10^5	4000	1500
40	10^{12}	10^7	10^4	10^4
50	10^{15}	$\underline{10^9}$	10^6	10^5
75	10^{22}	10^{13}	$\underline{10^9}$	10^8
100	10^{30}	10^{17}	10^{12}	$\underline{10^{10}}$

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 $f(1/\varepsilon) \times n^{O(1)}$) alors le problème de décision associé est FPT.

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 $f(2k) \times n^{O(1)}$ 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 $v(S^*) > v(S) - v(S^*)/2k > k - 1$ donc $v(S^*) \geq k$. la réponse est non.

Cours 8: Complexité paramétrique

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

Noyau et kernelisation

Donnée: un problème $\mathcal{P}$ de décision paramétré par k.
Problème: Donner un algo qui, à partir d'une instance X de $\mathcal{P}$ de taille n et paramètre k, construit, en temps polynomial en n, une instance $N(X)$ de $\mathcal{P}$ de taille au plus $g(k)$ (où g est une fonction indépt de n) de même réponse, appelée noyau de X.

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).

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 renvoit un noyau prédéfini.
- sinon $f(k) > n$, *i.e.* on a déjà une instance de taille au plus $f(k)$.

Noyau et kernelisation

Donnée: un problème $\mathcal{P}$ de décision paramétré par k.
Problème: Donner un algo qui, à partir d'une instance X de $\mathcal{P}$ de taille n et paramètre k, construit, en temps polynomial en n, une instance $N(X)$ de $\mathcal{P}$ de taille au plus $g(k)$ (où g est une fonction indépt de n) de même réponse, appelée noyau de X.

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).

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

en effet comme f est exponentielle, on a intérêt à ce que g ne croisse pas trop vite, mais aussi, l'existence d'un noyau linéaire peut donner un algo d'approximation

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.

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 le paramètre naturel ne donne pas de résultat, comme DOMINATING SET: non FPT en général
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)

Dans d'autres cas encore il faut faire appel à des paramètres cachés

- on va voir les paramètres structurels de type largeurs arborescente

Cours 8: Complexité paramétrique

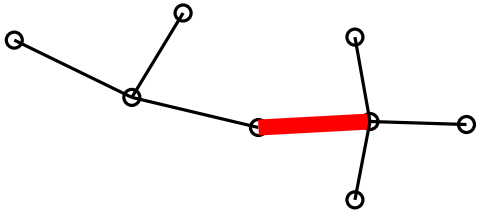
- Backtracking, branch and bound
- Vertex Cover et la complexité paramétrique
- Réduction au noyau (kernelisation)
- Largeur arborescente

Décomposition arborescente, largeur arborescente

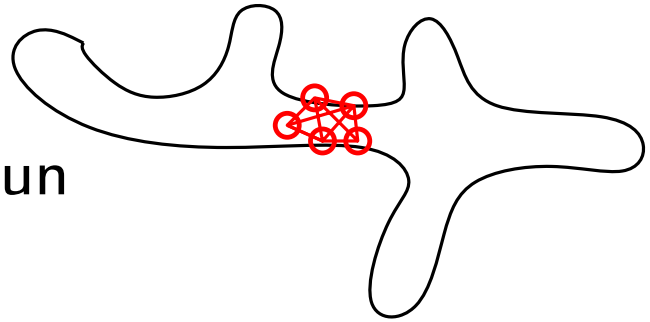
Mesurer si un graphe est plus ou moins loin d'être un arbre...

Intérêt des arbres: décomposer n'importe où et recommencer récursivement

Remplacer les paires séparatrices
par des k-séparateurs



$\Rightarrow$ graphe inscrit dans un
"arbre de largeur k"



Arbre de décomposition d'un graphe G:

– les nœuds sont des sacs formés de copies des sommets de G: tout sommet est dupliqué dans un ensemble de sacs formant un sous-arbre connexe.

– toute arête $\{x, y\}$ de G doit être incluse dans au moins un sac

la largeur de la décomposition est la taille du plus gros sac moins 1

la largeur arborescente du graphe G est la largeur de la décomposition arborescente la moins large.

Treewidth=1 $\Leftrightarrow$ arbres Treewidth=2 $\leftrightarrow$ graphes séries-parallèles

Décomposition arborescente sympatique

On peut montrer qu'il est toujours possible de se ramener à une décomposition arborescente dite "sympatique":

– Des noeuds binaires, pour l'union de deux branches de même sac racine.
– Des noeuds unaires d'ajout: pour ajouter un sommet au sac fils
– Des noeuds unaires d'élimination: pour enlever un sommet du sac fils
– Des feuilles: avec un sac arbitraire.

Dominating set FPT pour la treewidth

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

Problème: existe-t-il un ensemble D de ℓ sommets dominants (tq tout sommet de G a un voisin dans D) ?

NP-complet et pas de FPT connu pour le paramètre ℓ.

Supposons qu'on ait une décomposition sympatique de G de largeur k.

On fait un calcul bottom-up sur l'arbre (dynamic programming):

– pour chaque sous-arbre de l'arbre calculer les configurations d'étiquettes admissibles pour les sommets du sac racine: dominant, dominé ou libre
  – un sommet introduit est dominé s'il y a un dominant dans le sac
  – un sommet exclus ne doit pas être libre
  – lors d'une union les étiquettes doivent être compatibles.

Le calcul est mené de bas en haut, pour chaque nœuds de l'arbre on calcule pour chaque configuration admissible le nombre de dominants nécessaires.

SAT

Donnée: un ensemble de clauses $C = \{C_i\}$ sur les variables $X = \{x_j\}$.

Problème: $\exists$ une assignation des variables qui satisfasse toutes les clauses ?

Largeur arborescente d'une formule ?

la largeur arborescente du graphe d'incidence variable/clause !

Théorème: SAT est FPT par rapport à la largeur arborescente.

Même stratégie que pour dominant: programmation dynamique sur un arbre de décomposition sympatique

Largeur arborescente, conclusions

Exemples:

Le probleme k-DOMINANT SET est FPT par rapport à la largeur arborescente du graphe

Le problème SAT est FPT par rapport à la largeur arborescente du graphe d'incidence variables/clauses

Les programmes écrits dans un langage impératif sans GOTO ont un graphe de dépendance des flots de largeur arborescente au plus 6
$+$ le k-coloriage des graphes est FPT pour la treewidth
$\Rightarrow$ un algorithme efficace pour l'allocation de registres.

Tout problème expressible en logique monadique du second ordre sur les structures de graphes est FPT pour la treewidth.

(un mot de la logique monadique du 2nd ordre sur les structures de graphes ?)

Remarque. En général le calcul de la largeur arborescente est NP-complet mais il est FPT pour le paramêtre largeur arborescente lui-même.

Cours 8: Complexité paramétrique

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

Retenir: Heuristique branch-and-bound et problème FPT
La kernelisation donne des algo pratiques pour k petits !
Utilisation de paramètres structuraux

Aller plus loin: une théorie de la complexité paramétrique
Treewidth, edgewidth et théorie des mineurs exclus de Robertson et Seymour

- Annexe : heuristique de type branch and bound pour le problème MAX-EQUI-PARTITION

Partition équitable d'un graphe

Donnée: un graphe G, une valuation v des arêtes

Problème: trouver une partition X_1, X_2 des sommets tq

$$|X_1| = |X_2| \pm 1, \text{ et } \sum_{x \in X_1, y \in X_2} v(x,y) \text{ minimal}$$

Modélisation quasi-linéaire: une variable x_i par sommet, à valeur $\{0,1\}$

$$n/2 - 1 \leq \sum_i x_i \leq n/2 + 1, \quad \text{minimiser } \sum_{(i,j) \in E} v_{i,j} |x_i - x_j|$$

Branch and bound ? il faut les ingrédients suivants...

- une solution initiale
- une stratégie de parcours de l'arbre (choix du P à décomposer)
- un moyen de diviser le problème P en sous problèmes $P_1, \ldots, P_k$ tels que l'ensemble des solutions de P soit couvert
- une fonction $g(P_i)$ qui donne une bonne borne sur l'optimum dans P_i.

idée: à chaque étape on élimine les P_i tels que $g(P_i)$ est moins bon que la meilleure solution trouvée jusqu'ici.

Partition équitable d'un graphe

Donnée: un graphe G, une valuation v des arêtes

Problème: trouver une partition X_1, X_2 des sommets tq

$$|X_1| = |X_2| \pm 1, \text{ et } \sum_{x \in X_1, y \in X_2} v(x,y) \text{ minimal}$$

Modélisation quasi-linéaire: une variable x_i par sommet, à valeur $\{0,1\}$

$$n/2 - 1 \leq \sum_i x_i \leq n/2 + 1, \quad \text{minimiser } \sum_{(i,j) \in E} v_{i,j} |x_i - x_j|$$

Branch and bound ? il faut les ingrédients suivants...

- une solution initiale heuristique ! influe beaucoup sur l'efficacité

pour PartitionÉquitable: démarrer avec $X_1 := \{x_1\}$

Pour i de 1 à $n/2$, ajouter à X_1 le sommet y dont la somme des distances aux éléments de X_1 déjà choisis est maximale.

On obtient une partition valide mais pas forcément optimale.

Partition équitable d'un graphe

Donnée: un graphe G, une valuation v des arêtes

Problème: trouver une partition X_1, X_2 des sommets tq

$$|X_1| = |X_2| \pm 1, \text{ et } \sum_{x \in X_1, y \in X_2} v(x,y) \text{ minimal}$$

Modélisation quasi-linéaire: une variable x_i par sommet, à valeur $\{0,1\}$

$$n/2 - 1 \leq \sum_i x_i \leq n/2 + 1, \quad \text{minimiser } \sum_{(i,j) \in E} v_{i,j} |x_i - x_j|$$

Branch and bound ? il faut les ingrédients suivants...

- un moyen de diviser le problème P en sous problèmes $P_1, \ldots, P_k$ tels que l'ensemble des solutions de P soit couvert

On divise l'espace des solutions en ajoutant des contraites: le plus simple est d'instancier des variables *bien choisies*

PartitionÉquitable: on choisit une variable x_j encore non instanciée et on divise le problème en deux, suivant que $x_j \in X_1$ ou $x_j \in X_2$.

Partition équitable d'un graphe

Donnée: un graphe G, une valuation v des arêtes

Problème: trouver une partition X_1, X_2 des sommets tq

$$|X_1| = |X_2| \pm 1, \text{ et } \sum_{x \in X_1, y \in X_2} v(x,y) \text{ minimal}$$

Modélisation quasi-linéaire: une variable x_i par sommet, à valeur $\{0,1\}$

$$n/2 - 1 \leq \sum_i x_i \leq n/2 + 1, \quad \text{minimiser } \sum_{(i,j) \in E} v_{i,j} |x_i - x_j|$$

Branch and bound ? il faut les ingrédients suivants...

• une fonction $g(P_i)$ qui donne une bonne borne sur l'optimum dans P_i.

c'est le problème le plus délicat dans la méthode...

pour PartitionÉquitable: On a affecté partiellement X_1 et X_2, et les sommets de Y restent, et on pose $p = |X_1|$, $q = |X_2|$.

Alors le coût minimal d'une solution de ce type sera au moins la somme du coût des arêtes suivantes: entre X_1 et X_2; les $\frac{n}{2} - q$ plus petites entre chaque élément de X_1 et Y; les $\frac{n}{2} - p$ plus petites entre chaque élément de X_2 et Y, les $(\frac{n}{2} - p)(\frac{n}{2} - q)$ plus petites entre les éléments de Y.