Chapitre 11

Recherche multidimensionnelle

Soit d un entier positif. On se donne un ensemble S de n points de $\mathbb{R}^d$ et un ensemble $\mathcal{B}$ de parties de $\mathbb{R}^d$ appelées boîtes. Le problème de la recherche multidimensionnelle est de répondre à une requête du type :

Compter ou reporter les points de S contenus dans une boîte donnée de $\mathcal{B}$.

Pour répondre le plus rapidement possible à ce type de requête on construit généralement une structure de recherche qui aidera à répondre à toutes les requêtes possibles. Cette construction est donc considérée comme un pré-calcul dont le temps d'exécution peut raisonnablement être important par rapport au temps de réponse de chaque requête individuelle. La taille de cette structure doit par contre rester relativement faible dans la mesure où elle persiste pour toutes les requêtes. Dans la pratique un algorithme possédant un faible temps de requête nécessitera une structure plus volumineuse qu'un algorithme moins performant. Un exemple extrême consiste en une structure de taille linéaire se limitant à la liste des points de S et un algorithme de recherche au mieux linéaire obtenu en testant individuellement si chaque point de S est contenu dans la boîte de requête. Le problème général de la recherche multidimensionnelle suppose ainsi de faire des compromis entre l'espace requis pour la structure de données et la rapidité de réponse à une requête.

Références :

- [dBCvKO08, Chap. 5 and 16], [Mat94], [AE99], [GO04, Chap. 36].

11.1 Recherche orthogonale

La recherche orthogonale désigne le cas particulier de la recherche multidimensionnelle où l'ensemble des boîtes $\mathcal{B}$ est l'ensemble (infini) des pavés de $\mathbb{R}^d$.

Une approche directe consiste à pré-calculer toutes les réponses possibles ($\subset \mathcal{P}(S)$) et trouver un critère sur les boîtes pour savoir quand elles donnent la même réponse. Par ce moyen on obtient des temps de requêtes très performants mais des temps de pré-calculs très importants – ce qui n'est pas forcément rédhibitoire – et surtout des stockages très importants - ce qui est plus ennuyeux car ils persistent.

Exemple dans le plan : on trace une verticale et une horizontale par chaque point de S de manière à obtenir une grille. Deux boîtes donnent la même réponse si et seulement si leurs coins supérieurs gauches et inférieurs droits sont dans les mêmes cases de la grille. On a donc au plus $\binom{(n+1)^2}{2}$ réponses non-vides possibles (la grille possède $(n+1)^2$ cases). Comme chaque réponse peut contenir $O(n)$ points, on obtient naïvement une structure de stockage de taille $O(n^5)$ pour le problème du report des points.

On caractérise un algorithme de recherche par la paire (taille stockage, temps de requête).

11.1.1 Recherche unidimensionnelle

En dimension 1, le problème de la recherche orthogonale revient à compter ou reporter tous les points de S contenus dans un intervalle $[x, x']$ de requête.

Pour ce faire, on utilise un arbre binaire de recherche équilibré (la hauteur des sous-arbres gauche et droit de tout noeud diffère de un au plus) avec les points rangés aux feuilles, les noeuds internes contenant des valeurs de séparation entre les clés des sous-arbres gauche et droit. On suppose également que les feuilles sont chaînées dans l'ordre croissant des coordonnées des points. Pour le problème du report, on peut chercher dans l'arbre la feuille contenant x puis marcher jusqu'à x' en utilisant le chaînage. Ceci fournit une réponse en temps $(\log n + k)$, où k est le nombre de points à reporter. Cette méthode se généralise difficilement en dimension supérieure. Pour y remédier on commence par introduire la notion suivante :

Définition 11.1 *L'ensemble des clés des feuilles associées à un sous arbre de racine ν s'appelle l'*ensemble canonique *de ν et est noté $P(\nu)$.*

Plutôt que d'utiliser un chaînage aux feuilles on travaille avec la notion d'ensemble canonique. On commence ainsi par rechercher les feuilles contenant x et x' dans l'arbre de recherche. Les chemins de recherche de x et x' partent de la racine, restent confondus sur une certaine longueur, puis se séparent en deux sous-chemins γ_x et $\gamma_{x'}$ à partir d'un noeud ν_s. Il est facile de voir que l'ensemble cherché est l'union des ensembles canoniques des enfants droits des noeuds internes du chemin γ_x et des enfants gauches des noeuds internes du chemin $\gamma_{x'}$. À cette union, on ajoute éventuellement les feuilles de γ_x et γ'_x suivant les comparaisons de x ou x' relativement à ces feuilles. Notons que le calcul de l'ensemble canonique $P(\nu)$ d'un noeud ν peut s'obtenir en temps proportionnel à sa taille $|P(\nu)|$ puisque la taille d'un arbre binaire équilibré est proportionnelle à son nombre de feuilles. En résumé,

Lemme 11.2 (Le problème de la recherche orthogonale en dimension 1) *En utilisant un arbre binaire de recherche on obtient :*
pré-calcul : $O(n \log n)$
espace : $O(n)$
temps requête : $O(\log n + k)$
où k est le nombre de points à reporter.

Exercice 11.3 *Modifier l'algorithme pour le problème du comptage seul et non du report des points eux-mêmes. Quelle est la complexité d'une requête ?*

Voir également : arbres d'intervalles, arbre de recherche de priorité et arbres de segments.

11.1.2 Kd-trees (arbres k-dimensionnels) (Bentley 1975)

Pour résoudre le problème de la recherche orthogonale en dimension $d > 1$, on construit récursivement un arbre binaire équilibré sur les points de S, stockés dans les feuilles de cet arbre, tel que pour chaque noeud ν de profondeur k, les ensembles canoniques de ses noeuds enfants constituent une partition de $P(\nu)$ en deux selon la valeur médiane de la (k mod d)-ème coordonnée. Ainsi, l'ensemble canonique de l'enfant gauche (resp. droit) de ν est le sous-ensemble des points de $P(\nu)$ dont la (k mod d)-ème coordonnée est majorée (resp. strictement minorée) par cette valeur médiane.

Notons que la valeur médiane des points ordonnés selon une coordonnée particulière peut se calculer en temps linéaire. On peut soit utiliser d listes – une par coordonnées – triées au départ et que l'on décime au fur et à mesure que l'on descend dans l'arbre, soit utiliser l'algorithme linéaire classique de calcul de médiane rappelé ci-dessous :

1. Choisir un pivot soit aléatoirement soit par la méthode suivante : couper la liste en $n/5$ groupes de 5 clés et calculer la médiane de chaque groupe, puis calculer récursivement la médiane de ces médianes,
2. scinder la liste en deux selon cette valeur,
3. itérer sur la sous-liste contenant la vraie médiane (i.e. la valeur de rang milieu).

Complexité : $T(n) = O(n) + T(n/5) + T(7n/10) = O(n)$. En effet la plus petite sous-liste contient au moins $3 \times 1/2 \times n/5$ éléments et du coup la plus grande en contient au plus $7n/10$.

Dans le plan, le temps de construction $C(n)$ d'un $2d$-arbre est donné par

$$C(n) = \begin{cases} O(1) & \text{si } n = 1 \\ O(n) + 2C(\lceil n/2 \rceil)) & \text{sinon} \end{cases} \tag{11.1}$$

On en déduit un temps de construction en $O(n \log n)$.

On peut associer à chaque noeud ν d'un $2d$-arbre arbre sur S une boîte $B(\nu)$ de la forme $[x_1, x_1'] \times [x_2, x_2']$ avec $x_1, x_1', x_2, x_2' \in \overline{\mathbb{R}}$, de sorte que $P(\nu) = S \cap B(\nu)$. Pour cela, on associe à la racine la boîte égale au plan tout entier et on associe aux enfants d'un noeud ν de profondeur k les deux boîtes obtenues en coupant la boîte $B(\nu)$ en deux par le plan d'équation $x_{k \bmod 2} = x_{med}$ où x_{med} est la valeur médiane de séparation associée à ce noeud. Clairement, les boîtes de tous les noeuds de même profondeur forment une partition de l'espace et de ce fait leurs ensembles canoniques constituent une partition de S.

Pour rechercher tous les points dans une boîte B donnée on descend récursivement à partir de la racine d'un $2d$-arbre arbre sur S :

1. Si B contient la boîte $B(\nu)$ du noeud courant ν on renvoie l'ensemble canonique $P(\nu)$ et on arête la récursion,
2. sinon, si B et $B(\nu)$ sont disjoints, on arête la récursion,

3. sinon on lance la recherche sur les deux enfants de ν.

L'ensemble $S \cap B$ est l'union (disjointe) des $P(\nu)$ collectés lors de la recherche. Le temps de recherche est égal au nombre de noeuds visités plus le temps total pour rendre les $P(\nu)$. Ce dernier temps est proportionnel au nombre de points de $S \cap B$. En effet, l'ensemble canonique d'un noeud peut être collecté en temps proportionnel à sa taille puisque les 2d-arbres sont équilibrés. Pour majorer le nombre de noeuds internes visités (cas 3 dans la récurrence ci-dessus) on remarque que ceux-ci ont leur boîte intersectée par le bord de B et donc par les droites supports de B. En majorant le nombre de boîtes $B(\nu)$ intersectées par une droite donnée de direction horizontale ou verticale on obtient donc un majorant du nombre de noeuds internes visités (à un facteur 4 près) durant la recherche pour B.

Fixons une droite verticale D. On note $I(n)$ le nombre maximal de noeuds internes dont la boîte intersecte D dans un $2d$-arbre quelconque de taille n. Puisque la racine d'un $2d$-arbre est associée à une dichotomie verticale, la droite D ne peut intersecter que 2 des 4 boîtes associées aux noeuds de profondeur 2. Comme les noeuds de profondeur 2 sont également associés à des dichotomies verticales et eux-mêmes racines de $2d$-arbres de taille au plus $\lceil n/4 \rceil$, on peut écrire

$$I(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ 2 + 2I(\lceil n/4 \rceil)) & \text{sinon} \end{cases} \tag{11.2}$$

D'où, suivant le master theorem 1.34 ou suivant un calcul directe sur l'arbre de récursion, $I(n) = O(\sqrt{n})$. Un raisonnement analogue permet de borner le nombre de noeuds intersectés par une droite horizontale.

En résumé

Lemme 11.4 *Le problème de la recherche orthogonale dans le plan peut se résoudre en utilisant un $2d$-arbre avec :*
pré-calcul : $O(n \log n)$
espace : $O(n)$
temps requête : $O(\sqrt{n} + k)$ *où k est le nombre de points à reporter.*

Exercice 11.5 *Généraliser à $d > 2$ dimensions. Montrer en particulier que le temps de recherche est en $O(n^{1-1/d} + k)$.*

11.1.3 Arbres de domaines

Les arbres de domaines (*range trees*), introduits par Bentley en 1979, sont des structures de partitionnement de l'espace à plusieurs niveaux. En dimension 2, on commence par construire un arbre binaire de recherche équilibré selon la première coordonnée, puis pour chaque noeud ν on construit un arbre binaire de recherche équilibré sur l'ensemble canonique de ce noeud selon la deuxième coordonnée. En remarquant que les ensembles canoniques $P(\nu)$ correspondant à des noeuds de profondeur donnée forment une partition de l'ensemble des points, et que la hauteur de l'arbre de recherche "primaire" est en $O(\log n)$ on voit que la taille d'un arbre de domaines est en $O(n \log n)$. Pour la construction, on commence par trier les points selon la deuxième coordonnées y. Puis on crée une

racine en la faisant pointer sur un arbre binaire de recherche selon la deuxième coordonnée. Cet arbre binaire de recherche est construit en temps linéaire à partir de la liste triée de manière "bottom-up". On scinde alors la liste en deux selon x et on construit les deux sous-listes triées en y à partir de la liste triée de départ. Puis on continue récursivement. De cette manière le temps de construction est proportionnel à la taille de la structure finale (chaque arbre "secondaire" associé à un ensemble canonique est construit en temps proportionnel à sa taille).

Pour effectuer une requête avec une boîte $B = [x, x'] \times [y, y']$ on commence par rechercher les valeurs x et x' dans l'arbre primaire, puis pour chaque sous-arbre à droite (gauche) du chemin de recherche γ_x ($\gamma_{x'}$) situé après le noeud de séparation on fait une recherche en y dans la structure secondaire associée à sa racine. Le temps de la recherche est donc majoré par

$$\sum_{\nu} O(\log n + k_\nu) = O(\log^2 n + k)$$

où ν décrit les racines des sous-arbres sus-cités et k_ν le nombre de points reportés dans les ensembles canoniques associés.

Lemme 11.6 *Le problème de la recherche orthogonale dans le plan peut se résoudre en utilisant un arbre de domaines de dimension 2 avec :*
pré-calcul : $O(n \log n)$
espace : $O(n \log n)$
temps requête : $O(\log^2 n + k)$ *où k est le nombre de points à reporter.*

Remarque : Par rapport aux $2d$-arbres on a diminué le temps de requête mais augmenté la taille de la structure.

Exercice 11.7 *Généraliser les arbres de domaines à $d > 2$ dimensions.*

Exercice 11.8 *Jusqu'à maintenant, on a implicitement supposé que les k-ièmes coordonnées des points étaient deux à deux distinctes pour chaque k. Si ce n'est pas le cas on peut considérer la transformation*
$(x, y) \mapsto ((x, y), (y, x))$ *et utiliser l'ordre lexicographique sur les couples (x, y). La boîte de requête devient alors $[(x, -\infty), (x', \infty)] \times [(y, -\infty), (y', \infty)]$. Généraliser ce procédé à la dimension d.*
Remarque : cette astuce revient à perturber les données, c'est à dire à opérer une transformation du type $(x, y) \mapsto (x + \epsilon y, y + \epsilon x)$ où $\epsilon |y_i|_{max} \leq \min_{x_i \neq x_j} |x_i - x_j|$ et de même pour y.

Exercice 11.9 *Si on permet aux points d'avoir une partie de leurs coordonnées identiques on peut s'intéresser à tous les points ayant certaines de leurs coordonnées fixées par une requête. On parle alors de requête d'identification partielle (partial match query).*

1. Montrer qu'avec un $2d$-arbre on peut répondre à une requête d'identification partielle en temps $O(\sqrt{n} + k)$ où k est le nombre de points à reporter.

2. Trouver une structure de données utilisant un espace linéaire et qui répond à une requête d'identification partielle en temps $O(\log n + k)$.

3. Montrer qu'avec un kd-arbre de dimension d (i.e. un dd-arbre) on peut répondre à une requête d'identification partielle sur $s < d$ coordonnées en temps $O(n^{1-s/d} + k)$.

4. Trouver une structure de données pour répondre à une requête d'identification partielle en temps $O(d \log n + k)$ en dimension d, si on permet que la structure soit de taille $O(d2^d n)$.

11.1.4 Fractionnement en cascade

Le fractionnement en cascade (Lueker 1978 et Willard 1978) est une modification des arbres de domaines qui permet de gagner un facteur $\log n$ dans le temps de requête. On s'intéresse à la dimension 2. Dans les arbres de domaines les ensembles canoniques sont rangés dans des arbres binaires de recherche en la deuxième coordonnée. On remplace ces arbres par de simples listes triées (selon la deuxième coordonnée) en ajoutant à chaque élément d'une liste un pointeur vers le premier élément supérieur ou égal dans chacune des listes des deux noeuds enfants. On conserve de plus l'arbre binaire de recherche sur l'ensemble canonique de la racine.

La taille de la structure est la même que celle d'un arbre de domaines, $O(n \log n)$, et elle peut facilement être construite dans le même temps.

Pour effectuer une requête avec une boîte $B = [x, x'] \times [y, y']$ on procède comme pour un arbre de domaines en déterminant les chemins de recherches γ_x et $\gamma_{x'}$. On recherche ensuite dans l'ensemble canonique du noeud racine le premier sommet dont l'ordonnée est supérieure ou égale à y et l'on "propage" ce premier sommet le long des chemins de recherche à l'aide des pointeurs précédemment définis. Dans chaque noeud dont on doit reporter une partie de l'ensemble canonique on marche alors à partir de ce premier sommet jusqu'au dernier sommet d'ordonnée inférieure ou égale à y'. On en déduit un temps de requête en

$$\log n + \sum_{\nu} (O(1) + k_\nu) = O(\log n + k)$$

où ν et k_ν sont définis comme pour les arbres de domaines.

Lemme 11.10 *Le problème de la recherche orthogonale dans le plan peut se résoudre en utilisant le fractionnement en cascade avec :*
pré-calcul : $O(n \log n)$
espace : $O(n \log n)$
temps requête : $O(\log n + k)$ *où k est le nombre de points à reporter.*

Exercice 11.11 *Généraliser la technique du fractionnement en cascade en dimension $d > 2$. Montrer en particulier qu'on obtient un temps de recherche en $O(\log^{d-1} n + k)$.*

Note : Le problème de la recherche othogonale est l'un des plus étudiés en géométrie algorithmique. Étant donné l'interaction entre temps de requête et taille de la structure,

Chazelle [Cha90] a étudié la taille minimale requise si on impose un temps de requête avec report en temps $O(k + polylog(n))$. Autrement dit, on impose un temps de réponse proportionnel à cette dernière plus un polylog [1]. Chazelle étudie cette question dans le cadre d'une variante de machine à pointeurs et montre que toute structure de recherche en dimension d nécessite un espace $\Omega(n(\log n/\log\log n)^{d-1})$. En deux dimensions, et pour le modèle RAM en supposant que les points sont sur une grille entière de taille n^2 où la machine RAM traite des entiers de $\log n$ bits, Chan et al. [CLP11] obtiennent une première structure de recherche de taille $O(n\log\log n)$ pour un temps de requête $O(\log\log n(1+k))$ et une seconde structure de taille $O(n)$ pour un temps de requête $O((1+k)\log^{\varepsilon} n)$.

11.2 Recherche simpliciale et par demi-espaces

S désigne à nouveau un ensemble de n points de $\mathbb{R}^d$ fixé une fois pour toute. Le problème de la recherche simpliciale (resp. par demi-espace) consiste à reporter, ou compter, tous les points de S contenus dans un simplexe (resp. un demi-espace) de requête. Si une requête de type demi-espace peut sembler plus simple que pour un pavé parallèle aux axes, elle est en réalité bien plus difficile et bien plus générale du fait de l'orientation a priori quelconque de l'hyperplan support du demi-espace de requête. En particulier, la recherche par demi-espaces permet de répondre à des requêtes portant sur des boîtes non-linéaires par un procédé classique de linéarisation. Supposons par exemple vouloir effectuer une requête de type disque dans le plan. Par la transformation $(x, y) \mapsto (x, y, x^2 + y^2)$, cette requête se traduit aisément en une requête de type demi-espace dans $\mathbb{R}^3$. La recherche simpliciale, que l'on peut voir comme une application de la recherche par demi-espaces, permet quant-à elle de répondre à des requêtes portant sur des polyèdres de tailles bornées en effectuant une triangulation préalable des polyèdres.

11.2.1 Arbre de partitions

L'idée est de partitionner les points en sous-ensembles de tailles approximativement égales inclus dans des domaines dont la description est simple de sorte que la boîte de requête intersecte une faible proportion de ces domaines. On procède alors récursivement sur les points inclus dans chaque domaine pour définir un arbre de partitions.

On regarde le problème dans le plan.

Définition 11.12 *Une* partition simpliciale *d'un ensemble S de n points du plan est une famille de couples $\{(S_i, \Delta_i)\}_{i\in I}$ où les S_i forment une partition de S et les Δ_i sont des triangles du plan de sorte que pour chaque i on a $S_i \subset \Delta_i$. Notons que les triangles Δ_i peuvent se chevaucher et qu'un point de S peut appartenir à plusieurs Δ_i. Le nombre $|I|$ de parties est la* taille *de la partition.*

Soit $r \geq 1$. Une partition simpliciale de paramètre r, *ou r-partition, sur S est une partition simpliciale de S dont chaque sous-ensemble S_i contient entre n/r et $2n/r$ points. Notons que la taille d'une r-partition est comprise entre $r/2$ et r.*

1. *polylog*(n) désigne une fonction de la forme $P(\log n)$ pour un polynôme P fixé.

Le nombre de croisements *d'une droite avec une partition simpliciale est le nombre de triangles de la partition intersectés par cette droite. Le* nombre de croisements *de la partition est le maximum de ce nombre sur toutes les droites du plan.*

Le théorème 12.9 de la partition simpliciale affirme que pour tout r fixé on peut construire en temps $O(n)$ une r-partition de S de nombre de croisements $O(\sqrt{r})$.

On construit récursivement un *arbre de partitions* de paramètre r sur un ensemble S de n points en associant à la racine un nombre d'enfants en correspondance avec les sous-ensembles d'une partition simpliciale de S de paramètre r fixé. On stocke les descriptions des triangles Δ_i de la partition au niveau des enfants puis on continue récursivement sur chaque enfant avec son sous-ensemble S_i associé tant que ce sous-ensemble compte au moins $2r$ points.

Lemme 11.13 *Soit $r > 2$. Un arbre de partitions de paramètre r sur un ensemble de n points a une taille linéaire (i.e. en $O(n)$) et peut être construit en temps $O(n \log n)$.*

Preuve : Puisque chaque partition simpliciale utilisée est de paramètre r, le degré de chaque noeud interne de l'arbre de partitions est au moins $r/2$. Soit n_I et n_E les nombres respectifs de noeuds internes et externes (feuilles) de l'arbre de partitions. En comptant de deux façons différentes le nombre d'arêtes de l'arbre de partitions à partir de l'extrémité respectivement inférieure ou supérieure de chaque arête, on obtient :

$$(r/2)n_I \leq n_I + n_E - 1.$$

Soit

$$n_I \leq \frac{2n_E - 2}{r - 2}.$$

Or chaque feuille de l'arbre de partitions est associée à au moins un point et les ensembles associés aux feuilles sont disjoints. On en déduit $n_E \leq n$ et par suite la taille de l'arbre de partitions est linéaire.

Le temps de construction $C(n)$ vérifie d'après le théorème 12.9 de la partition simpliciale

$$C(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ O(n) + \sum_\nu C(n_\nu) & \text{sinon} \end{cases} \tag{11.3}$$

où ν parcourt les enfants de la racine. Comme $n_\nu \leq 2n/r$, l'arbre de partitions a une hauteur $O(\log n)$. Par ailleurs, l'ensemble des noeuds de profondeur donnée forme une partition des n points et peut donc être traité en temps $O(n)$. On en déduit $C(n) = O(n \log n)$. □

Pour répondre à une requête du type demi-plan l^+ bordé par une droite l on part de la racine puis on teste si chacun des triangles des (au plus r) enfants est contenu dans l^+, disjoint de l^+, ou intersecte l. Dans les deux premiers cas on sait quoi faire sinon on "récurre" sur les triangles enfants intersectés. Ceci permet de sélectionner un ensemble de

noeuds dont les ensembles canoniques forment une partition des points contenus dans l^+. La complexité $R(n)$ de cette sélection est donnée par

$$R(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ O(r) + \sum_\nu R(n_\nu) & \text{sinon} \end{cases} \tag{11.4}$$

Soit, compte tenu du paramètre r de la partition simpliciale :

$$R(n) \leq cr + c\sqrt{r}R(2n/r).$$

pour une certaine constante c que l'on peut supposer $> 1/\sqrt{2}$. En choisissant $r = \lceil 2(c\sqrt{2})^{1/\epsilon} \rceil$ on a

$$0 < \log_{r/2} c\sqrt{r} = \frac{\log(c\sqrt{r})}{\log(r/2)} = \frac{\log \sqrt{r/2}\log(c\sqrt{2})}{\log(r/2)} = \frac{1}{2} + \frac{\log(c\sqrt{2})}{\log(r/2)} \leq \frac{1}{2} + \frac{\log(c\sqrt{2})}{\log((c\sqrt{2})^{1/\epsilon})} \leq \frac{1}{2} + \epsilon.$$

On en déduit, par le master theorem 1.34, $R(n) = O(n^{\log_{r/2} c\sqrt{r}}) = O(n^{1/2+\epsilon})$. Dans le cas du report, il faut rendre les ensembles canoniques des noeuds sélectionnés. Pour chaque noeud, cet ensemble est l'union des points stockés aux feuilles du sous-arbre dont il est racine. Comme ce sous-arbre à une taille linéaire, on en déduit que le problème du report peut-être résolu avec un temps supplémentaire proportionnel à la taille de la réponse.

Note : En prenant $r = \sqrt{n}$ et en résolvant $R(n) = O(r) + O(\sqrt{r})R(2n/r)$ on trouve $R(n) = O(\sqrt{n}\, 2^{O(\log\log n)}) = O(\sqrt{n}\, polylog\, n)$.

Pour répondre à une requête de type triangle au lieu de demi-plan on peut utiliser la même structure et obtenir la même complexité asymptotique de recherche en remarquant que, à chaque niveau de la recherche, le nombre de triangles d'une partition intersectés par les 3 droites supports d'un triangle de requête est en $O(3\sqrt{r})$. Finalement :

Lemme 11.14 *Le problème de la recherche simpliciale dans le plan peut se résoudre en utilisant un arbre de partitions avec :*
pré-calcul : $O(n \log n)$
espace : $O(n)$
temps requête : $O(n^{1/2+\epsilon})$ *pour le décompte et* $O(n^{1/2+\epsilon} + k)$ *pour le report où* k *est le nombre de points à reporter.*

11.2.2 Arbres de cuttings et recherche par demi-plan

Les arbres de partitions ont évidemment une taille optimale mais un temps de requête important. Quitte à augmenter la taille de la structure de recherche, il est possible d'obtenir des temps de recherche logarithmiques. C'est l'objet de ce qui suit. La méthode repose ici sur une transformation préalable des données par la dualité point/droite du plan. Celle ci, notée $*$, est donnée par la transformation $(a, b) \mapsto \{y = ax - b\}$, et son inverse, et définit une bijection entre le plan et l'ensemble des droites non verticales du plan.

Propriété : le point p est au dessus de la droite d si et seulement si d^* est au dessus de p^*.

Étant donné une droite d, trouver tous les points s_i de S au dessus de d revient ainsi à trouver l'ensemble des droites de $\{s_i^* | s_i \in S\}$ au dessous de d^*. Clairement cet ensemble

ne dépend que de la cellule de l'arrangement des droites s_i^* contenant d^*. Il y a $O(n^2)$ telles cellules. On peut donc espérer un algorithme utilisant une place $O(n^2)$ avec un temps de requête en $O(\log n)$.

Définition 11.15 *Soit L un ensemble de n droites du plan et soit $r \in [1, n]$. Un* $(1/r)$-cutting *pour L est une partition du plan en triangles (possiblement non bornés) telle que chaque triangle est intersecté par au plus n/r droites de L. La* taille *de ce cutting est son nombre de triangles.*

Le théorème 12.1 affirme que l'on peut construire en temps $O(nr)$ un $(1/r)$-cutting de taille $O(r^2)$ en collectant pour chaque triangle du cutting les droites de L qui le coupent.

On s'intéresse tout d'abord au problème du décompte. On définit pour cela récursivement un *arbre de cuttings* pour L :
- si L contient une unique droite alors son arbre est réduit à une feuille qui stocke cette droite,
- sinon on crée autant d'enfants de la racine que de triangles dans un $(1/r)$-cutting de L. Pour chaque enfant ν, associé au triangle $\Delta(\nu)$, on construit récursivement un arbre de cuttings pour les droites $L(\nu) \subset L$ intersectant $\Delta(\nu)$. L'ensemble de droites $L(\nu)$ est appelé l'*ensemble canonique* de ν. Pour le problème du comptage on stocke également pour chaque enfant les nombres $\ell^-(\nu)$ et $\ell^+(\nu)$ de droites de L respectivement au dessus et au dessous de $\Delta(\nu)$.

La taille $T(n)$ d'un arbre de cuttings construit à l'aide de $(1/r)$-cuttings, chacun de taille $O(r^2)$, vérifie

$$T(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ O(r^2) + \sum_\nu T(n_\nu) & \text{sinon} \end{cases} \tag{11.5}$$

Où ν décrit les enfants d'un arbre de cuttings sur n droites et n_ν la taille de son ensemble canonique. Par définition d'un $1/r$-cutting on a encore pour une certaine constante c :

$$T(n) \leq cr^2 + cr^2 T(n/r)$$

D'où $T(n) = O(n^{\log_r cr^2})$. Soit en prenant $r = \lceil c^{1/\epsilon} \rceil$, $T(n) = O(n^{2+\epsilon})$.

Compte tenu du temps de calcul d'un $(1/r)$-cutting, le temps $C(n)$ de construction de cette structure vérifie

$$C(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ O(nr) + \sum_\nu C(n_\nu) & \text{sinon} \end{cases} \tag{11.6}$$

Soit encore $C(n) \leq cnr + cr^2 C(n/r)$ pour une certaine constante c, ce qui donne à nouveau $C(n) = O(n^{2+\epsilon})$ pour $r = \lceil c^{1/\epsilon} \rceil$.

Pour trouver le nombre de droites sous un point p de requête on descend récursivement dans l'arbre de cuttings depuis la racine jusqu'aux feuilles en s'orientant pour chaque noeud visité vers l'unique enfant dont le triangle associé contient p. La somme des nombres ℓ^- associés aux noeuds de ce parcours est le nombre cherché. Bien entendu, en accumulant les valeurs ℓ^+ au lieu de ℓ^- la même structure permet de calculer le nombre de droites au dessus d'un point de requête.

Le temps de recherche $R(n)$ vérifie ainsi :

$$R(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ O(r^2) + R(n/r) & \text{sinon} \end{cases} \tag{11.7}$$

D'où $R(n) = O(\log n)$ si r est constant.

Pour le problème du report, pour chaque noeud ν de l'arbre de cuttings, si μ est son parent, on stocke explicitement au niveau de ν les sous-ensembles $L^+(\nu), L^-(\nu) \subset L(\mu)$ de droites respectivement au dessus et au dessous du triangle $\Delta(\nu)$. L'ensemble des droites au dessous (resp. au dessus) d'un point de requête est obtenu récursivement comme dans le cas du décompte en accumulant cette fois les ensembles L^- (resp. L^+) le long du parcours. Le temps de requête se décompose alors en un temps de parcours $O(\log n)$ comme pour le décompte plus un terme proportionnel à la taille de la réponse. La taille $T'(n)$ de la structure de recherche vérifie maintenant

$$T'(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ O(nr^2) + \sum_\nu T'(n_\nu) & \text{sinon} \end{cases} \tag{11.8}$$

D'où $T'(n) = O(nr^2 + r^2 T'(n/r))$. Par application du master theorem 1.34 on déduit à nouveau $T'(n) = O(n^{2+\epsilon})$. En résumé,

Lemme 11.16 *Le problème de la recherche par demi-plan peut se résoudre en utilisant un arbre de cuttings avec :*
pré-calcul : $O(n^{2+\epsilon})$
espace : $O(n^{2+\epsilon})$
temps requête : $O(\log n)$ *pour compter et* $O(\log n + k)$ *pour reporter où* k *est le nombre de points à reporter.*

11.2.3 Application à la recherche simpliciale

Soit Δ un triangle, intersection de trois demi-plans H_1, H_2, H_3 bordés respectivement par les droites h_1, h_2, h_3. On suppose que H_1 est au dessous de h_1 tandis que H_2 et H_3 sont au dessus de leur droite bordante respective. Les autres cas se traitent de manière analogue. Le problème de la sélection des points de S inclus dans Δ se dualise alors en un problème du type : trouver les droites de S^* simultanément au dessus de h_1^* et au dessous de h_2^* et de h_3^*.

Pour répondre à ce type de requête – et donc par dualité à une requête de type triangle – on peut utiliser un structure d'arbre de cuttings à trois niveaux : le premier niveau est un arbre de cuttings simple. On associe de plus à chaque noeud ν deux arbres de cuttings (à deux niveaux) : l'un sur son ensemble $L^+(\nu)$ et l'autre sur son ensemble $L^-(\nu)$. Finalement on associe à chaque noeud de la structure secondaire deux arbres de cuttings simples sur ses ensembles L^+ et L^-. La recherche procède comme suit. On effectue une requête avec h_1^* sur l'arbre primaire pour sélectionner les droites au dessus de h_1^*. Pour chaque noeud ν sélectionné dans cette recherche, plutôt que de rendre (ou comptabiliser) $L^+(\nu)$, on effectue une requête avec h_2^* sur sa structure secondaire afin de sélectionner les droites de $L^+(\nu)$ au dessous de h_2^*. Finalement on effectue une requête avec h_3^* sur les structures

associées aux noeuds secondaires sélectionnés pour rechercher les droites au dessous de h_3^*. Clairement, cette procédure permet de sélectionner les droites simultanément au dessus de h_1^* et au dessous de h_2^* et de h_3^*.

Le temps de recherche dans une structure secondaire vérifie :

$$R'(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ O(\log n + r^2) + R'(n/r) & \text{sinon} \end{cases} \tag{11.9}$$

D'où $R'(n) = O(\log^2 n)$ si r est constant. Le temps de recherche dans la structure primaire vérifie donc :

$$R(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ O(\log n^2 + r^2) + R(n/r) & \text{sinon} \end{cases} \tag{11.10}$$

D'où $R(n) = O(\log^3 n)$ si r est constant. La taille de la structure secondaire vérifie :

$$T'(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ O(r^2 + n^{2+\epsilon}) + \sum_\nu T'(n_\nu) & \text{sinon} \end{cases} \tag{11.11}$$

Soit (cf. master theorem) $T'(n) = O(n^{2+\epsilon})$ si r est constant et suffisamment grand. On en déduit de même que la taille de la structure primaire est un $O(n^{2+\epsilon})$. On montre selon les mêmes lignes que celle-ci peut-être construite dans le même temps. Finalement,

Lemme 11.17 *Le problème de la recherche simpliciale dans le plan peut se résoudre en utilisant un arbre de cuttings à trois niveaux avec :*
pré-calcul : $O(n^{2+\epsilon})$
espace : $O(n^{2+\epsilon})$
temps requête : $O(\log^3 n)$ *pour compter et* $O(\log^3 n + k)$ *pour reporter où* k *est le nombre de points à reporter.*