Chapitre 5

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 temps et espace requis pour le pré-calcul et rapidité de réponse à une requête.

Références :

- Chap. 5 and 16 of Computational Geometry. Algorithms and applications. de Berg, van Kreveld, Overmars and Schwarzkopf. Springer 1997.

- Chap. 36 of Handbook of Discrete and Computational Geometry. Goodman and O'Rourke eds. CRC Press 2004.

- P.K. Agarwal, J. Erickson, Geometric range searching and its relatives, in : B. Chazelle, J.E. Goodman, R. Pollack (Eds.), Advances in Discrete and Computational Geometry, AMS Press, 1999, pp. 1-56.

5.1 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 $\mathbb{R}^d$.

On peut regarder 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 $\binom{n+1}{2}^2$ réponses non-vides possibles (une boîte est déterminée par deux verticales et 2 horizontales). 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).

5.1.1 Recherche unidimensionnelle

Problème : trouver tous les points (nombres) contenus dans un intervalle $[x, x']$ donné.

Solution : utiliser 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 peut chercher la feuille contenant x puis marcher avec un pointeur 'suivant' entre feuilles jusqu'à x'. Cette méthode se généralise difficilement en dimension supérieure. Pour y remédier on commence par introduire la notion suivante :

Définition 5.1 *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 pointeur 'suivant' entre 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 (resp. gauches) des noeuds internes du chemin γ_x (resp. $\gamma_{x'}$). 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 $P(\nu)$ puisque la taille d'un arbre binaire équilibré est proportionnelle à son nombre de feuilles.

Lemme 5.2 (Le problème de la recherche 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 trouvés.

On peut également utiliser des structures de recherches randomisées comme dans le chapitre précédent et obtenir des complexités équivalentes en moyenne.

Exercice 5.3 *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.

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

Pour résoudre le problème en dimension $d > 1$, on construit récursivement un arbre binaire équilibré tel que pour chaque noeud ν de profondeur k, les ensembles canoniques de ces noeuds enfants constituent une partition de $P(\nu)$ 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.

Remarque : pour trouver la valeur médiane des points ordonnés selon une coordonnée particulière 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 rappeler 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 $3 * 1/2 * n/5$ éléments et du coup la plus grande en contient au plus $7n/10$.

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

$$C(n) = \begin{cases} O(1) & \text{si } n = 1 \\ O(n) + 2C(\lceil n/2 \rceil)) & \text{sinon} \end{cases} \tag{5.1}$$

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

Pour effectuer la recherche on remarque que chaque noeud ν de l'arbre correspond à une boîte $B(\nu)$ de la forme $[x_1, x'_1] \times [x_2, x'_2]$ avec $x_1, x'_1, x_2, x'_2 \in \overline{\mathbb{R}}$. En effet la racine correspond à tout le plan et les enfants d'un noeud de profondeur k correspondent à la boîte de ce noeud coupée par le plan d'équation $x_{k \bmod 2} = x_{med}$ où x_{med} est la valeur médiane de séparation associée à ce noeud. De plus 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 on descend donc à partir de la racine, puis
- si la boîte B intersecte, sans la contenir, la boîte $B(\nu_g)$ (resp. $B(\nu_d)$) de l'enfant gauche (resp. droit) du noeud courant on continue la recherche sur cet enfant,
- sinon si $B(\nu_g)$ (resp. $B(\nu_d)$) est contenu dans B on renvoie l'ensemble canonique $P(\nu_g)$ (resp. $P(\nu_d)$).

Le temps de recherche est égal au nombre de noeuds visités plus le temps pour rendre les $P(\nu)$. Ce dernier est proportionnel au nombre k de clés retournées. Pour majorer le nombre de noeuds visités on remarque que les noeuds visités (i.e. dont on regarde si les deux sous-arbres enfants doivent être visités) 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 visités (dont on ne rend pas l'ensemble canonique).

On note $I(n)$ le nombre maximal de "noeuds intersectés" par une droite verticale dans un $2d$-arbre de taille n. Cette droite ne peut intersecter que 2 des 4 boîtes associées aux noeuds de profondeur 2. On peut donc écrire

$$I(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ 2 + 2I(\lceil n/4 \rceil)) & \text{sinon} \end{cases} \tag{5.2}$$

D'où $I(n) = O(\sqrt{n})$.

On obtient finalement le

Lemme 5.4 *En utilisant un 2d-arbre on obtient :*
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 5.5 *Généraliser à $d > 2$ dimensions. Montrer en particulier que le temps de recherche est en $O(n^{1-1/d} + k)$.*

5.1.3 Range trees (Bentley 1979, ...)

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 $O(\log n)$ en voit que la taille d'un range-tree 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 linéaire).

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 5.6 *En utilisant un range-tree de dimension 2 on obtient :*
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 augmenter la taille de la structure.

Exercice 5.7 *Généraliser les range-trees à $d > 2$ dimensions.*

Exercice 5.8 *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 5.9 *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 $O(d2^d n)$.

5.1.4 Fractional cascading et layered range trees (Lueker 1978 et Willard 1978)

On s'intéresse d'abord à la dimension 2. Remarquons que dans un arbre binaire de recherche selon la première coordonnée on a pour tout noeud ν : $P(\nu) \subset P(\text{parent}(\nu))$. On remplace les arbres binaires de recherche en la deuxième coordonnée sur $P(\nu)$ utilisés par les range-trees par des listes triées (selon la deuxième coordonnée) où chaque élément pointe dans les deux listes des noeuds enfants sur la plus petite "valeur" qui lui est supérieure ou égale.

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

Pour effectuer une requête avec une boîte $B = [x, x'] \times [y, y']$ on procède comme pour un range-tree en déterminant les chemins de recherches γ_x et $\gamma_{x'}$. On recherche ensuite la plus petite valeur supérieure à y dans la liste du noeud de séparation de ces chemins et l'on propage en temps constant cette valeur à l'aide des pointeurs précédemment définis. Dans chaque noeud dont on doit reporter l'ensemble canonique on marche alors à partir de cette valeur jusqu'à la plus grande valeur inférieure ou égale à y'. On en déduit un temps de requête en

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

où ν et k_ν sont définis comme pour les range-trees.

Lemme 5.10 *En utilisant un layered range-tree de dimension 2 on obtient :*
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 5.11 *Généraliser les layered range-trees à $d > 2$ dimensions. Montrer en particulier qu'on obtient un temps de recherche en $O(\log^{d-1} n + k)$.*

Note : Chazelle obtient une structure optimale avec un espace $O(n \log n / \log\log n)$ et un temps de requête en $O(\log n + k)$. cf.
- Lower bounds for orthogonal range searching. Journal of the ACM. 1990.

5.2 Recherche simpliciale

Problème de la recherche simpliciale : étant donné un ensemble S de n points de $\mathbb{R}^d$ trouver, ou compter, tous les points contenus dans un simplexe de requête ou plus généralement dans un demi-espace.

Voir le survey paper de J. Matoušek. Geometric range searching. ACM Comput. Surv., 26 :421-461, 1994. http ://www.ms.mff.cuni.cz/acad/kam/matousek/oldpaps.html

5.2.1 Arbres de partition

L'idée est de partitionner les points en sous-ensembles inclus dans des domaines dont la description est simple (mais pouvant se recouper) 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 partition.

On regarde d'abord le problème dans le plan.

Définition 5.12 *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 sont des parties de S et les Δ_i des triangles du plan de sorte que les S_i forment une partition de S et 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 sur n points est bonne *(pour r) si chaque sous-ensemble de la partition contient entre n/r et $2n/r$ points. On appelle r le* paramètre *de la partition. Notons que la taille d'une bonne partition simpliciale de paramètre r est comprise entre $r/2$ et r.*

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.*

On admettra le théorème suivant.

Théorème 5.13 (de la partition simpliciale, Matoušek 1992) *Pour tout ensemble de n points et pour tout r ($1 \leq r \leq n/2$), il existe une bonne partition simpliciale de paramètre r et de nombre de croisements $O(\sqrt{r})$. De plus pour tout ϵ positif une telle partition peut être construite en temps $O(n^{1+\epsilon})$.*

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

Lemme 5.14 *Soit $r > 2$. Un arbre de partition de paramètre r sur un ensemble de n points a une taille linéaire (i.e. en $O(n)$) et peut être construit en temps $O(n^{1+\epsilon})$ pour tout ϵ positif.*

Preuve : Puisque chaque partition simpliciale utilisée est de paramètre r, le degré de chaque noeud interne de l'arbre est au moins $r/2$. Soit n_I et n_E les nombres respectifs de noeuds internes et externes de l'arbre de partition. En comptant de deux façons différentes le nombre d'arêtes de l'arbre de partition 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 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 $T(n) = O(n)$.

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

$$C(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ O(n^{1+\epsilon}) + \sum_\nu C(n_\nu) & \text{sinon} \end{cases} \tag{5.3}$$

où ν parcourt les enfants de la racine. Comme $n_\nu \leq 2n/r$, l'arbre de partition a une hauteur $O(\log n)$. Par ailleurs, l'ensemble des noeuds de profondeur donnée forme une partition des n points. Comme la fonction $x \mapsto x^{1+\epsilon}$ est convexe, on en déduit que le coût total de traitement de ces noeuds est un $O(n^{1+\epsilon})$, d'où $C(n) = O(n^{1+\epsilon} \log n) = O(n^{1+2\epsilon})$. □

Pour répondre à une requête du type demi-plan l^+ défini 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. La complexité $R(n)$ de la recherche est donnée par

$$R(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ O(r) + \sum_\nu R(n_\nu) & \text{sinon} \end{cases} \tag{5.4}$$

Soit, compte tenu de la bonne partition simpliciale :

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

On en déduit (par le master theorem) $R(n) = O(n^{\log_{r/2} c\sqrt{r}})$. En choisissant $r = \lceil 2(c\sqrt{2})^{1/\epsilon} \rceil$ on a

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

D'où $R(n) = O(n^{1/2+\epsilon})$.

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

Finalement :

Lemme 5.15 *En utilisant un arbre de partition de dimension 2 on obtient :*
pré-calcul : $O(n^{1+2\epsilon})$
espace : $O(n)$
temps requête : $O(n^{1/2+\epsilon} + k)$
où k est le nombre de points à reporter.

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 $O(3\sqrt{r})$.

5.2.2 Cutting trees

la dualité point/droite $(a, b) \mapsto \{y = ax - b\}$, notée *, 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 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 5.16 *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.*

Théorème 5.17 (admis) *Pour tout ensemble L de n droites du plan et tout $r \leq n$ il existe un $(1/r)$-cutting de taille $O(r^2)$ qui peut être construit en temps $O(nr)$ (en collectant pour chaque triangle les droites qui le coupent).*

On construit récursivement un *cutting-tree* pour L
- si L contient une unique droite on crée un feuille qui stocke cette droite
- sinon on crée un enfant de la racine par triangle d'un $(1/r)$-cutting de L et on construit récursivement un cutting-tree pour les droites intersectant le triangle associé à un enfant. On stocke également pour chaque enfant le nombre de droites (de son parent) L^- (resp. L^+) au dessus (resp. au dessous) de son triangle.

La taille $T(n)$ d'un cutting tree construit à l'aide de $(1/r)$-cutting de taille $O(r^2)$ vérifie

$$T(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ O(r^2) + \sum_\nu T(n_\nu) & \text{sinon} \end{cases} \tag{5.5}$$

D'après le cutting utilisé 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 $r = \lceil (2c)^{1/\epsilon} \rceil$, $T(n) = O(n^{2+\epsilon})$.

Le temps $C(n)$ de construction de cette structure vérifie

$$C(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ O(nr) + \sum_\nu C(n_\nu) & \text{sinon} \end{cases} \tag{5.6}$$

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 $r = \lceil (2c)^{1/\epsilon} \rceil$.

Pour trouver le nombre de droites sous un point p de requête on descend récursivement dans le cutting-tree 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.

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

$$R(n) \leq \begin{cases} O(1) & \text{si } n = 1 \\ O(r^2) + R(n/r) & \text{sinon} \end{cases} \tag{5.7}$$

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

Pour répondre à une requête de type triangle au lieu de demi-plan on peut utiliser un structure de cutting tree à trois niveaux : le premier niveau est un cutting tree simple auquel on associe à chaque noeud un cutting tree sur son ensemble canonique et on itère pour les noeuds de la structure secondaire. La recherche se fait en utilisant les trois droites supports d'un triangle pour la recherche dans les structures primaire, secondaire et tertiaire respectivement. Le temps de sélection des ensembles canoniques contenant la réponse est $O(\log^3 n)$.