Étude bibliographique : Le R-tree

# Sources

Guttman, A. (1984). ["R-Trees: A Dynamic Index Structure for Spatial Searching"](../bib/Gut84.Rtree.pdf). Proceedings of the 1984 ACM SIGMOD international conference on Management of data - SIGMOD '84. p. 47.

<https://en.wikipedia.org/wiki/R-tree>

Beckmann, N.; Kriegel, H. P.; Schneider, R.; Seeger, B. (1990). "The R\*-tree: an efficient and robust access method for points and rectangles". Proceedings of the 1990 ACM SIGMOD international conference on Management of data - SIGMOD '90. p. 322

[https://en.wikipedia.org/wiki/R\*\_tree](https://en.wikipedia.org/wiki/R*_tree)

T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: A dynamic index for multi-dimensional objects. In VLDB, 1987.

<https://en.wikipedia.org/wiki/R%2B_tree>

<https://en.wikipedia.org/wiki/Hilbert_R-tree>

<https://en.wikipedia.org/wiki/B%2B_tree#Bulk-loading>

https://en.wikipedia.org/wiki/B-tree#Algorithms

# Définition

Un R-tree est un genre de B-tree, adapté au stockage d’objets spatiaux. Les nœuds intermédiaires de l’arbre sont des pages de disque (gros tableau de pointeurs vers d’autres pages). Les feuilles sont des tableaux de pointeurs vers les données. La taille de ces tableaux est comprise entre un minimum *m* (typiquement supérieur à 2 mais pas trop gros) et un maximum *M*.

# Applications

Les R-tree sont utilisés principalement pour le stockage de données spatiales.

# Structure

## Définition

L’arbre est régi par six contraintes :

1. Tout nœud intermédiaire et différent de la racine possède entre *m* et *M* enfants.
2. Toute feuille contient entre *m* et *M* objets.
3. La racine a entre 2 et *M* enfants sauf si c’est une feuille.
4. Tout nœud intermédiaire est un tableau d’entrées de la forme (*I*, *id-enfant*), où *I* est le rectangle minimal englobant l’ensemble des rectangles englobants de l’enfant *id-enfant*.
5. Toute feuille contient un tableau d’entrées de la forme (*I*, *id-data*) où *I* est le rectangle minimal englobant l’objet *id-data*. Dans notre cas, la donnée est un point donc il pourra être envisageable d’utiliser une variante du R-tree : s’arrêter avant (quand ?)
6. Toutes les feuilles sont au même niveau.

Dans tout ce qui précède, les rectangles ont une dimension *n* (pour nous, *n* = 3, donc ce sont des pavés).

Étant donné qu’on ne stocke que des rectangles englobants minimaux, on peut dire que les R-tree ne stockent quasiment pas de vide.

## Propriétés

La hauteur d’un arbre contenant *N* objets n’excède pas .  
Le nombre maximal de nœuds est La complexité spatiale en pire cas pour chaque nœud est

## Algorithmes

### Constitution

On peut choisir une approche par insertion, ou une approche « *Bulk-loading* » plus rapide mais nécessitant de connaître la donnée.

### Recherche

La recherche de toutes les entités contenues dans un rectangle *S* donné est réalisée de manière descendante. Le critère de comparaison est : le *I* courant intersecte-t-il *S* ? Il peut être nécessaire de parcourir plusieurs enfants.

### Insertion

L’insertion est semblable au B-tree. On insère prioritairement dans le rectangle qui en sera le moins élargi.

### Suppression

La suppression, comme l’insertion peut nécessiter de réorganiser l’arbre. Dans notre cas, nous n’aurons pas à nous en servir.

## Création de l’arbre par « Bulk loading »

Lors d’une insertion standard, on parcourt tout l’arbre depuis la racine. Pour construire l’arbre, on peut utiliser une approche plus directe.

1. Trier le fichier source (!)
2. Insérer les entrées du fichier source dans la page la plus à droite juste au-dessus du niveau des feuilles.
3. Lorsque l’entrée la plus à droite est pleine, on la divise et on réorganise l’arbre.

## Conclusion

Que ce soit pour l’insertion ou la suppression, il peut être nécessaire de réorganiser l’arbre en coupant en deux un rectangle. La difficulté est de trouver une paire de rectangles stockant le moins possible de vide (aires minimales). L’algorithme naïf, de complexité exponentielle en *M*, explore toutes les paires possibles et choisit la meilleure. Guttman propose également un algorithme quadratique et un algorithme linéaire ; leurs performances après test sont semblables. Greene et Ang-Tan proposent d’autres algorithmes linéaires.

Pour notre sujet, il serait plus intéressant d’utiliser une des variantes du R-tree.

# Avantages et inconvénients

Le R-tree est mieux adapté que le B-tree au stockage d’objets multidimensionnels. La recherche a une complexité logarithmique par rapport au nombre d’objets mais plusieurs branches doivent parfois être explorées. L’insertion est quadratique ou linéaire, mais en améliorant la rapidité de l’insertion on détériore légèrement les performances de recherche et de stockage (plus de recouvrements).

# Autres méthodes

Méthodes cellulaires (?) cf. Guttman note 4

K-D-B-tree

## R\*-tree

Le R-tree est une variante du R-tree qui tend à améliorer la répartition des rectangles en minimisant les superpositions. Les algorithmes sont plus sophistiqués et de complexité équivalente au R-tree.

## R+-tree

Dans un R+-tree, on s’autorise à dupliquer les données qui appartiennent à plusieurs rectangles afin d’augmenter significativement les performances de recherche. En revanche, il nécessite davantage d’espace disque.

## R-tree de Hilbert

Dans ce R-tree, les rectangles sont découpés selon une courbe de Hilbert, ce qui diminue leur aire.