R++-tree: an efficient spatial access method for highly redundant point data - Martin Šumák, Peter Gurský

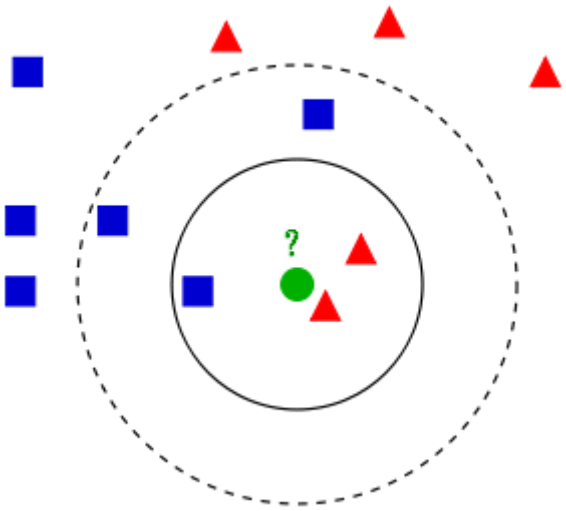
Recenzió: Németh Boldizsár

Térbeli indexelés

Az adatszerkezetek alapvetően fontos feladata, hogy a megadott feltételeknek megfelelő adatokat hatékonyan vissza lehessen nyerni. Egy nagyobb adathalmaz esetében nem használható az a triviális módszer, hogy az adatokat egyenként végignézzük.

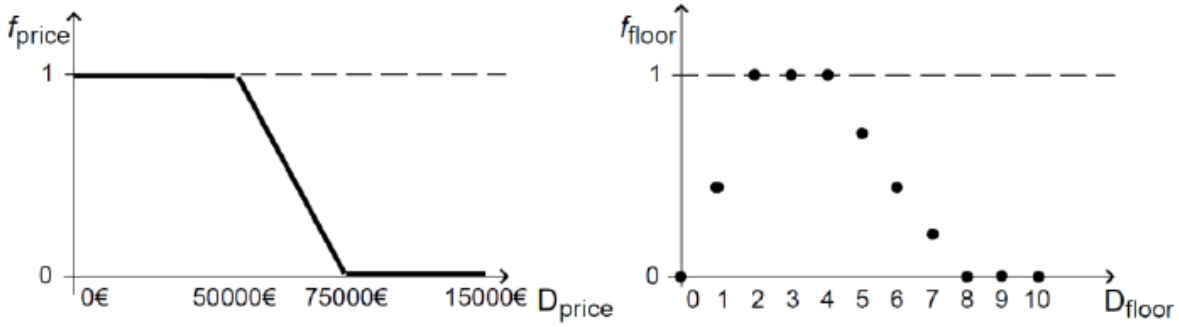
Adottak térbeli objektumok, amiket hatékonyan szeretnénk térbeli keresésekkel megtalálni. Például a következő keresések gyakoriak:

- A **tartomány**-keresés során egy adott (n-dimenziós) halmazon belül keresünk pontokat. Ez könnyebb feladat, ha az adott halmaz egy intervallum.
- A **k-NN** (k nearest neighbour) keresés során a kijelölt ponthoz legközelebbi k db objektumot keressük.



1. ábra – Egy k-NN keresés. A sima vonal k=3 esetét, a szaggatott a k=5 esetét mutatja

- **top-k**: Adott értékelési függvények alapján keressük a k legtöbbre értékelt pontot. Az értékelési függvények fuzzy függvények, vagyis 0 és 1 közötti értékre képeznek le lineáris módon. Ezeknek a súlyozott átlagát vesszük, és ez alapján szeretnénk megtalálni a k db legkedvezőbb objektumot.
  Példa: A felhasználó egy 50000 eurós ár alatti lakást szeretne, lehetőleg második, harmadik vagy negyedik emeleten. A lehetséges lakásokat most az ár és az emelet kétdimenziós terében tároljuk.



2. ábra – Fuzzy függvények az ár és emelet értékének megállapítására

R-fák

Cél: Egy n dimenziós térben pontok megtalálása bizonyos feltételek alapján. Gyorsan lehet válaszolni olyan kérdésekre, mint „Keresd meg az összes múzeumot 2 km-es körzetben” (tartomány keresés), vagy „Hol van a legközelebbi benzinkút?” (1-NN keresés).

Az egymáshoz közeli objektumokat csoportokba szervezi és a minimális bennfoglaló intervallumával (2 dimenziós térben téglalapjával) reprezentálja. Ha a keresett tartomány nem metsz egy intervallumot, akkor az általa reprezentált csúcsok nem lehetnek benne a tartományban. A következő szinteken az egymáshoz közeli intervallumokat is csoportosítjuk, és így tovább. Ez a csoportosítási rendszer egy olyan fát alakít ki, amiben a szülő csúcsok szigorúan tartalmazzák a gyermek csúcsokat, de a gyermek csúcsoknak nem kell diszjunktaknak lennie.

A **keresés** egyszerű, a gyökértől indulunk, és figyelmen kívül hagyjuk azokat a farészeket, amikben biztosan nem lehet az eredmény.

A **beszúrás** során a gyökértől lefele haladva valamilyen heurisztika szerint minden szinten eldöntjük, hogy melyik részfába szúrjuk be az objektumot úgy, hogy a lehető legkevésbé kelljen megnagyobbítani az adott intervallumot. Az objektumokat csak levél csomópontokba szúrjuk be. Ha egy levél csúcs megtelik, ketté kell vágni. A kettévágások egészen a gyökérig felterjedhetnek.

A **törlés** során az őscsomópontok intervallumait szűkítjük. Ha egy csomópontnak túl kevés gyermeke lesz, akkor töröljük és az elemeit újra beszúrjuk.

A teljes R fát nem kell a memóriában tartani, emiatt különösen jó az adatbázisban történő tárolásra. Minden csomópontot egy lapon tart.

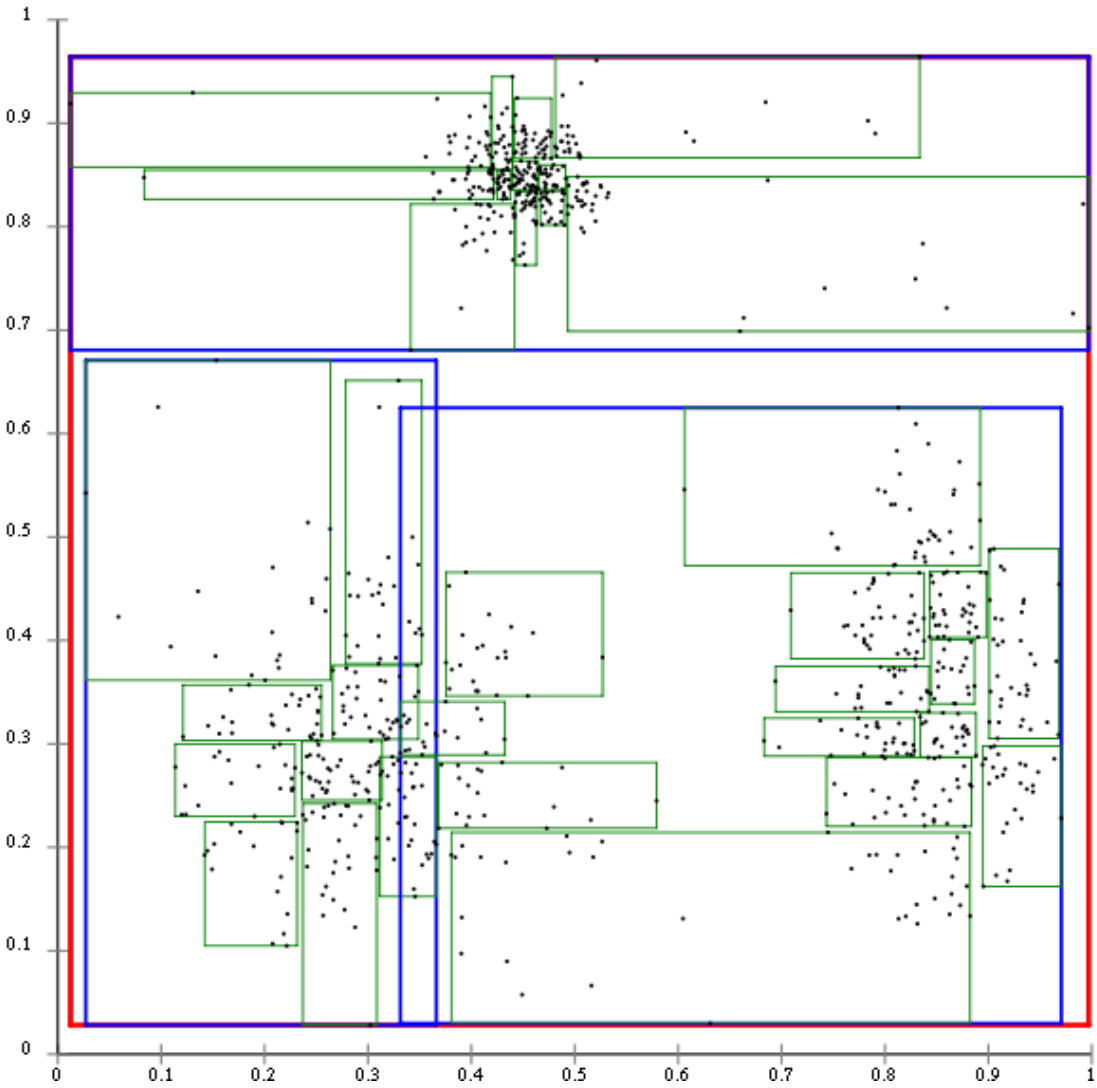
A kihívás az R fákkal kapcsolatban az, hogy úgy egyensúlyozzuk ki őket, hogy a fa levelei nagyjából egyforma mélységben legyenek, és az intervallumok ne tartalmazzanak túl sok üres teret, továbbá ne lógjanak túlságosan össze. Mivel ezek az alap R-fában nincsenek specifikálva, ezért ez nem garantál jó teljesítményt.

R* fa

Az R* fa egyszerre éri el a csomópontjai között az átfedés minimalizálását és a csomópontok területének csökkentését a gyors keresés érdekében. A keresésre nagyon hatékony, mert a csomópontok formája közel négyszögletes lesz, minimális területtel.

Beszúráskor, amennyiben egy csomópont telítődik, az elemei újra beszúrásra kerülnek a fába. Ez tekinthető egyfajta fokozatos optimalizációnak is, ennek köszönheti az R*-fa a kiegyensúlyozottságát. Azonban, emiatt a csomópontok beszúrása aránylag hosszú ideig tarthat.

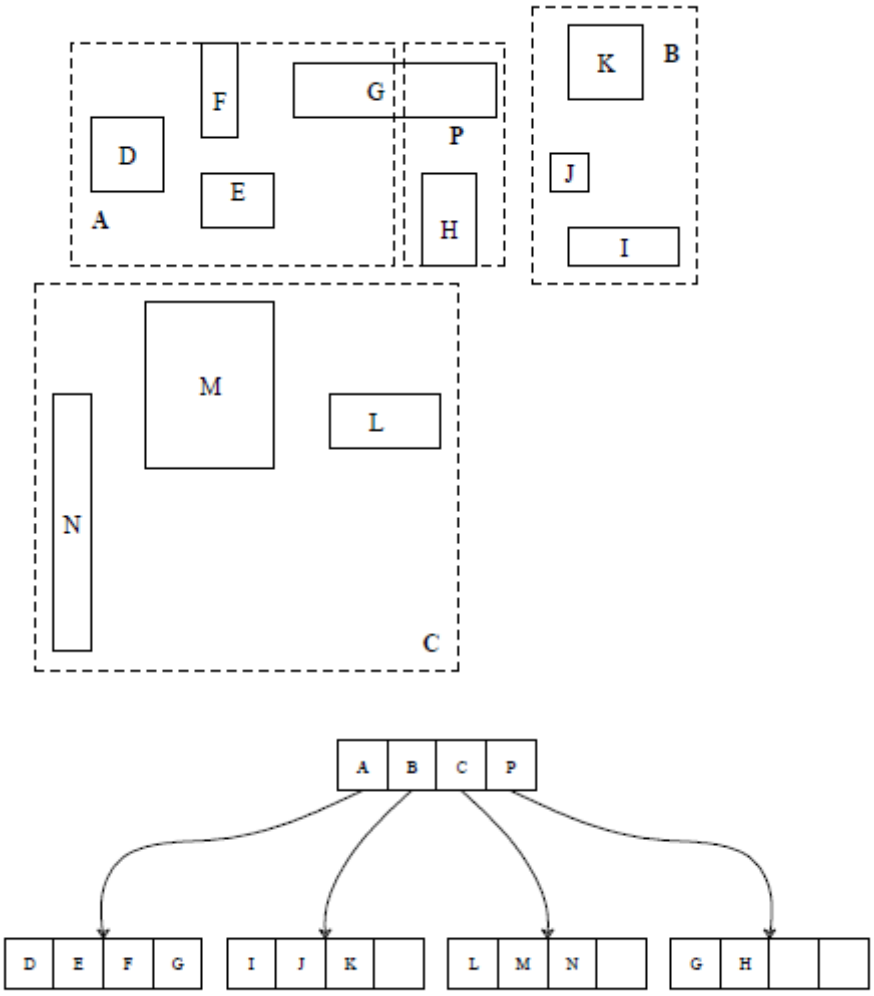
A keresésé és törlés az R-fához hasonló. Beszúráskor többféle heurisztikát is figyelembe veszünk.



3. ábra – Az ábrán jól látszik, hogy az R* fa csomópontjain jól követik az adatpontok sűrűségét.

R^+ fa

Nem tartalmaznak egymást átfedő intervallumokat. Ehelyett egy objektum több intervallumba is be lehet szúrva, ha szükséges.



4. ábra – Az ábrán jól látszik, hogy az R^+-fa csomópontjai között lehet átfedés. A G csomópont két másik csomópontnak is a gyermekévé válik.

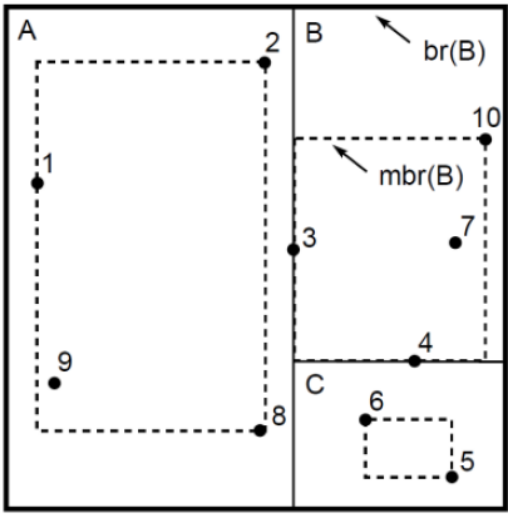
Az R^+ fa nem garantálja, hogy minden csomópont legalább félig teli lesz.

Mivel nincsenek egymást átfedő csúcsok, a kereséshez garantáltan csak egy utat kell végigjárni a gyökértől a tartalmazó levélig. Az objektumok duplikációja miatt azonban megnövekedhet az adatszerkezet mérete. A fa karbantartása is bonyolultabb, mint az egyszerű R fa esetében.

R^{++} fa

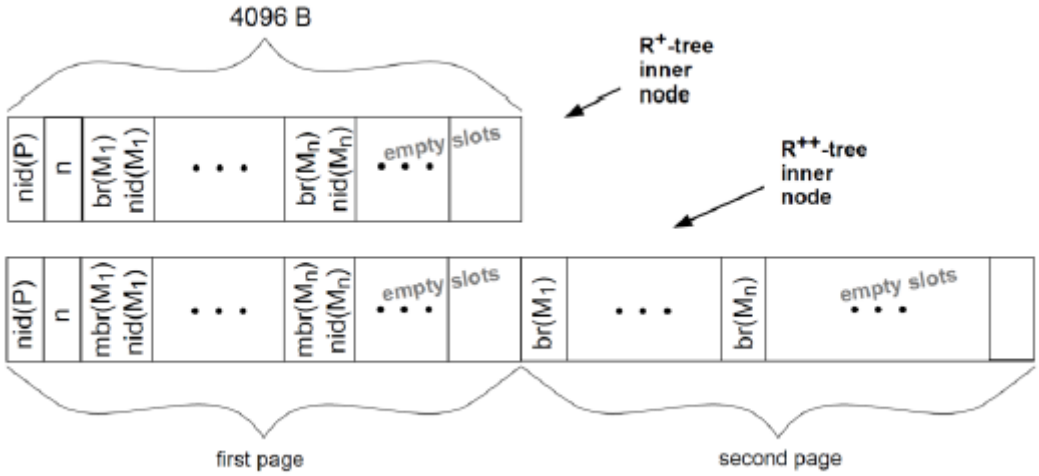
Az R^{++} fa, mint a nevéből sejteni lehet, az R^{+} fán alapul. A kutatók fő célja a hatékony top-k keresés támogatása volt, melyet az R^{+} fák csak rossz hatékonysággal biztosítják.

Az R^{++} fa csak pontszerű objektumokat tartalmazhat, szemben az R és R^{+} fákkal.



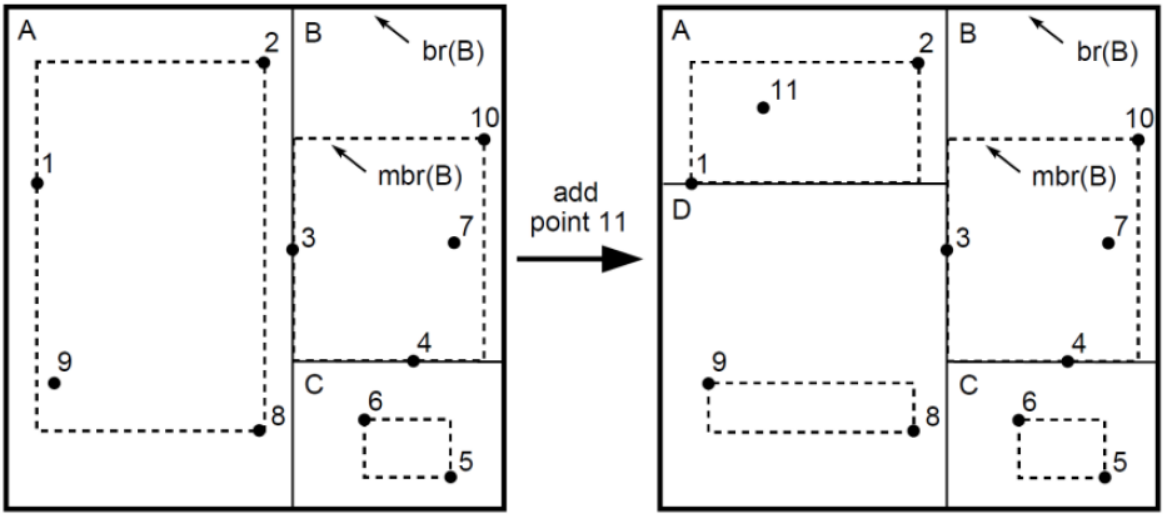
5. ábra - R^{++} fa

Egy csomópont egyszerre tárolja a gyermekei teljes kitöltő intervallumát (br, folytonos vonal) és minimális intervallumát (mbr, szaggatott vonal). Ettől lesz két lapnyi méretű (a lap alatt itt virtuális lapot értünk, vagyis az adattároló rendszer által meghatározott egységnyi adatmennyiséget). Kereséskor csak az első lapot kell olvasni a minimális bennfoglaló intervallumokkal (és a gyermek csúcsok azonosítóival).



A **keresés** úgy történik, mint az R és R^+ fák esetén.

Beszúráskor a kitöltő intervallumok alapján biztosan tudunk keresni egy olyan levelet, amibe az objektum beszúrható. Ezután megnöveljük a minimális bennfoglaló intervallumot úgy, hogy az új objektum benne legyen. (Mivel az objektum pontszerű, ez megtehető). Ha a levélben már nem fér el az új objektum, akkor a levél szét lesz osztva. Belső csúcsok is szétosztásra kerülhetnek, bár itt az algoritmus kissé komplikáltabb.

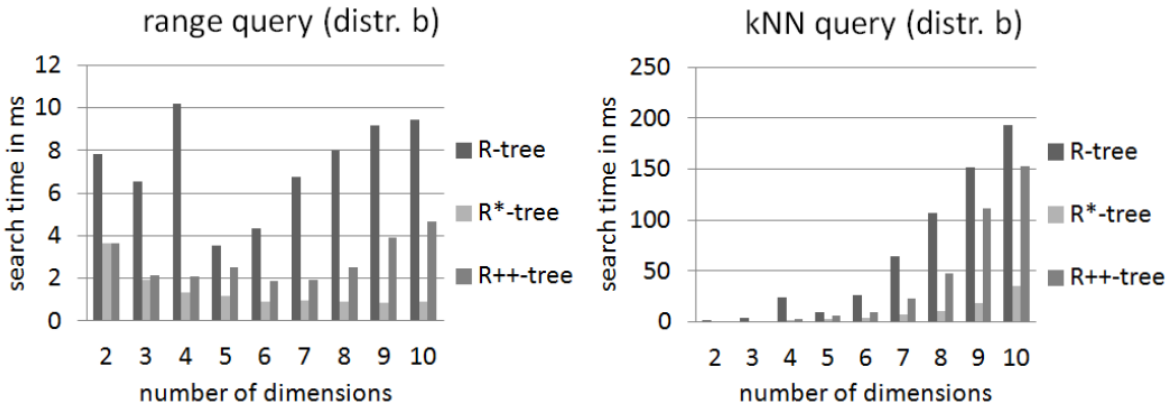


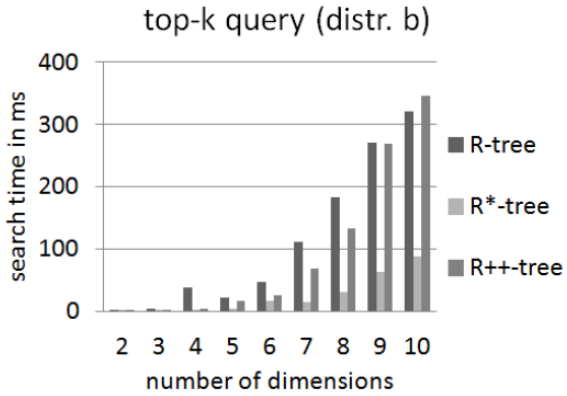
Az R++ fa hatékonysága

Az R^{++} fa lényegesen jobb az R^{*} fánál tartománykeresésekre, kNN és top-k keresésekre, ha az adatok redundánsak és a dimenziók száma alacsony. Négy dimenzió fölött azonban ezt az előnyét elvesztette.

Pszeudo-valós adatok esetén az R^{++} fa jobb teljesítményt nyújtott top-k keresésekre és ez az előny nem csökkent akkor sem, ha növeltük a vizsgált dimenziók számát.

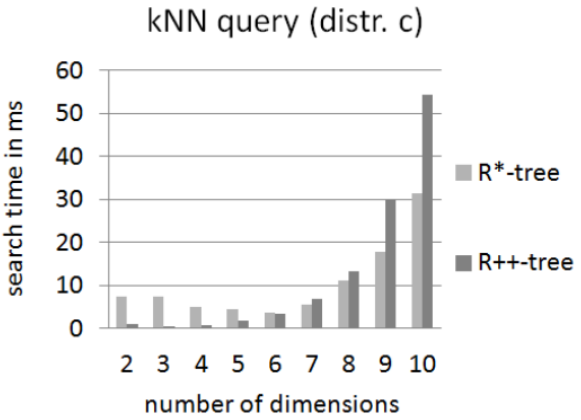
A következő ábrák bemutatják az R, R* és R^{++} fák teljesítményét egy olyan mesterséges adathalmazra, ami alacsony dimenziókban magas redundanciával rendelkezik, viszont magas dimenzióban már kevésbé redundáns.

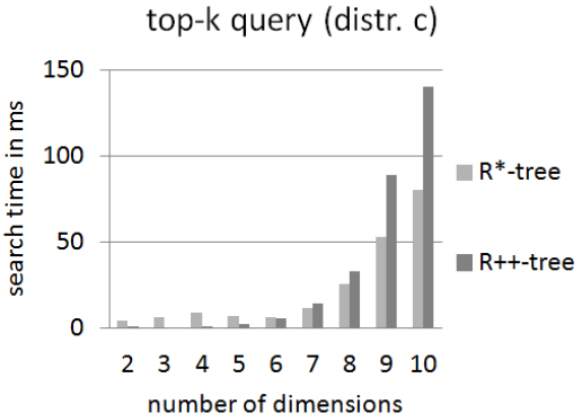




A következő ábrák bemutatják az R, R* és R^{++} fák teljesítményét egy olyan mesterséges adathalmazra, ami alacsony és magas dimenziószám esetén egyaránt redundáns.

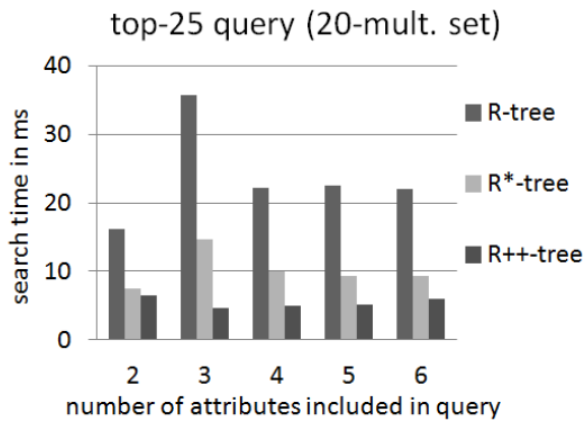
Ilyen esetekben az R^{++}-fa egyértelműen jobb teljesítményt mutat még az R*-fánál is négy dimenzióig. Afölött a tartomány-keresésekben továbbra is hatékonyabb, míg a többi keresésben a hatékonysága összehasonlítható az R*-fával.

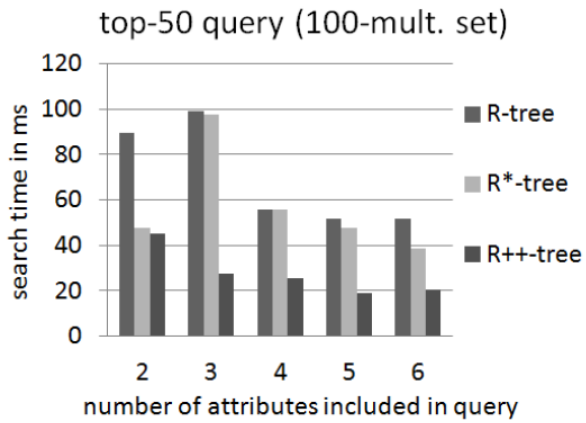




Az alábbi ábrák pszeudo-valós adathalmazokon hasonlítják össze az R, R* és R^{++} fák teljesítményét. Az adatok egy ingatlan adatbázisból származnak, és az adott 2-6 tulajdonság alapján súlyozva keressük a 25, illetve 50 legjobb találatot.

Az ábrákon látszik, hogy az R^{++}-fa minden esetben jobb volt az R*-fánál.





Források (a recenzióhoz):

- Az eredeti cikk: Martin Šumák, Peter Gurský - R++-tree: an efficient spatial access method for highly redundant point data
- Az R+ fákat bemutató eredeti cikk: Timos Sellis, Nick Roussopoulos, Christos Faloutsos - The R+-Tree: A Dynamic Index for Multi-Dimensional Objects
- R fák (wikipédia): http://en.wikipedia.org/wiki/R-tree
- R^+ fák (wikipédia): http://en.wikipedia.org/wiki/R%2B_tree
- R* fák (wikipédia): http://en.wikipedia.org/wiki/R*-tree
- K-legközelebbi szomszéd keresés (wikipédia): http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm