ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1998
Seria: INFORMATYKA z. 35 Nr kol. 1402

Katarzyna STĄPOR
Politechnika Śląska, Instytut Informatyki

# OPTYMALIZACJA STRUKTURY R-DRZEWA - INDEKSU PRZESTRZENNEGO DLA OBIEKTÓW WIELOWYMIAROWYCH

**Streszczenie.** W artykule została opisana heurystyczna optymalizacja przestrzennej metody dostępu, R-drzewa. Opisano optymalizację tej struktury zarówno w przypadku danych o charakterze statycznym (ok. 100% upakowania), jak i dynamicznym. Opierając się na opisanych kryteriach optymalizacji dla dynamicznej struktury R-drzewa zaimplementowano algorytm wprowadzania danych do R-drzewa. Tak uzyskana struktura wykazuje lepsze właściwości od klasycznej.

# OPTIMIZATION OF THE R-TREE STRUCTURE - THE SPATIAL ACCESS METHOD FOR MULTIDIMENSIONAL OBJECTS

**Summary.** This article describes heuristic optimization of the R-tree, which is one of the most popular spatial access methods. Two kinds of such opimization are presented: static, giving packed (about 100%) R-tree and dynamic, for which criteria of optimization are described. Base on these criteria the algorithm for inserting data into R-tree dynamic structure is given. The obtained structure has better performance than the classic one.

## 1. Wstęp

W związku z intensywnym rozwojem geograficznych systemów informacyjnych (ang. Geographical Information System, GIS) pojawiło się zapotrzebowanie na systemy tzw. przestrzennych baz danych, umożliwiające efektywne zarządzanie danymi przestrzennymi. Termin **dana przestrzenna** odnosi się do danej związanej z przestrzenią, którą może być np. 2-wymiarowa abstrakcja części powierzchni Ziemi. Dane przestrzenne złożone są ze stosunkowo prostych obiektów geometrycznych typu punkt, linia, prostokąt, powierzchnia.

Każdy obiekt przestrzenny opisany jest poprzez minimalnie **jeden atrybut przestrzenny**, tzn. atrybut opisujący jego geometrię i położenie w przestrzeni. Atrybut taki zawiera 2- lub 3-wymiarowe dane typu punkt, linia, prostokąt, wielokąt, powierzchnia. **Zapytania przestrzenne** to zapytania dotyczące właśnie atrybutów przestrzennych obiektów. Zapytania przestrzenne można podzielić na dwie grupy:

a) **zapytania pojedynczego dostępu** (ang. single-scan), wymagające co najwyżej jednego dostępu do obiektu w trakcie wykonania zapytania, charakteryzujące się liniową zależnością czasu wykonania zapytania od ilości obiektów w relacji. Najbardziej znanymi przykładami takich zapytań są: 1) zapytanie **typu punkt** (ang. point query): mając dany punt P w d-wymiarowej przestrzeni znajdź wszystkie d-wymiarowe prostokąty R, zawierające P, 2) zapytanie **typu okno** (ang. window query): mając dany d-wymiarowy prostokąt S znajdź wszystkie d-wymiarowe prostokąty R przecinające (zawierające, należące do) S, 3) zapytanie **typu najbliższego sąsiada** (ang. nearest neighbor).

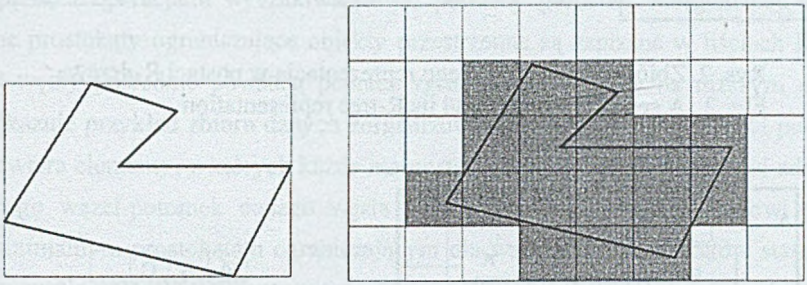
a) **zapytania wielokrotnego dostępu** (ang. multiplescan), wymagające wielokrotnego dostępu do obiektu w trakcie wykonania zapytania, charakteryzujące się subliniową raczej zależnością czasu wykonania zapytania od ilości obiektów. Najbardziej znanym przykładem takiego zapytania jest **złączenie przestrzenne** (ang. spatial join). Operacja złączenia przestrzennego polega na kombinacji obiektów z dwu zbiorów sterowanej pewnymi własnościami. Na przykład w przypadku relacji „lasy" i „miasta", gdzie atrybut przestrzenny reprezentuje w każdej z nich brzegi (granice) przykładem złączenia przestrzennego jest zapytanie: „znajdź wszystkie lasy w mieście".

Aby można było efektywnie przetwarzać wymienione zapytania przestrzenne, dane muszą zostać posortowane zgodnie z kluczami wyrażającymi ich własności przestrzenne, czyli położenie w przestrzeni. Takie techniki indeksacji są nazywane **indeksami przestrzennymi** lub **przestrzennymi metodami dostępu** (ang. spatial access method, SAM).

W metodach indeksacji przestrzennej dokonuje się **aproksymacji obiektów**, tak aby uzyskać jako indeks przestrzenny znacznie prostsze geometrycznie obiekty w stosunku do rzeczywistych obiektów przestrzennych. Aproksymacja umożliwia znaczną oszczędność zapisu złożonych obiektów, a podstawowe charakterystyki obiektu: położenie oraz rozciągłość obiektu wzdłuż wszystkich osi zostają zachowane. Stosowane są dwa rodzaje takiej aproksymacji obiektów:

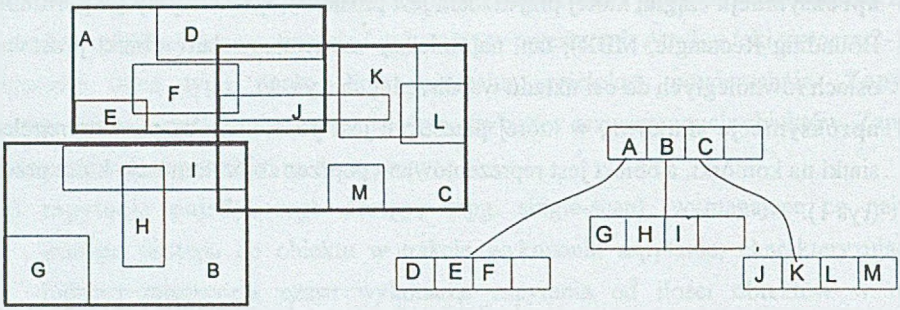
a) **aproksymacja ciągła,** której przykładem jest prostokąt ograniczający (ang. Minimum Bounding Rectangle, MBR), tzn. najmniejszy zawierający dany obiekt prostokąt, o osiach równoległych do osi układu współrzędnych (rys. 1),

b) **aproksymacja siatkowa,** w której przestrzeń jest podzielona za pomocą regularnej siatki na komórki, a obiekt jest reprezentowany poprzez zbiór komórek, które przecina (rys 1).

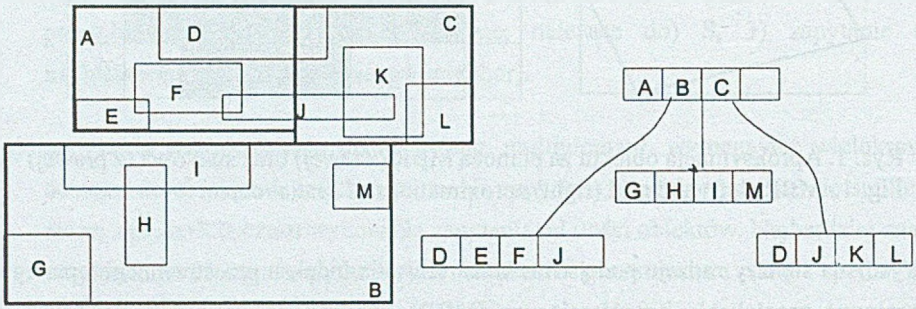


Rys. 1. Aproksymacja obiektu za pomocą MBR (z lewej) oraz siatkowa (z prawej)
Fig. 1. MBR (left) and grid (right) aproximations of spatial object

Wyróżnia się trzy rodzaje podejść do konstruowania indeksu przestrzennego opartego na minimalnym prostokącie ograniczającym (MBR) obiektu: 1) transformację danych, 2) obcinanie, 3) nakładanie się. Podejście oparte na **transformacji danych** (ang. transformation approach) polega na reprezentacji minimalnych prostokątów ograniczających obiekty przestrzenne poprzez punkty w przestrzeni o wyższym wymiarze, np. 2-wymiarowego prostokąta o osiach równoległych do osi układu współrzędnych za pomocą punktu 4-wymiarowego (**reprezentacja środkowa**): (c1, c2, e1, e2), gdzie (c1, c2) stanowią współrzędne środka prostokąta, zaś (e1, e2) odległości środka prostokąta od jego boków. Do takiego przekształconego zbioru prostokątów w postaci punktów 4-wymiarowych można zastosować np. strukturę indeksu przestrzennego typu grid file [6]. Podejście z **nakładającymi się obszarami** (ang. overlapping regions), w którym nie dokonuje się podziału przestrzeni danych, zaś obszary komór przestrzeni mogą zachodzić na siebie (np. struktura typu **R-drzewo** [4]). Na rys. 2 pokazany został przykład R-drzewa, korzeń zawiera prostokąt A, który jest prostokątem ograniczającym prostokąty D, E i F zapisane w potomku związanym z węzłem A. Prostokąty mogą zachodzić na siebie. Prostokąt może przecinać wiele obszarów komór, ale będzie reprezentowany tylko w jednym z nich. Problemem staje się jednak wyszukiwanie, które musi odbywać się wieloma ścieżkami, w przypadkach gdy interesujący obszar leży na przecięciu się dwóch obszarów potomków.




Rys. 2. Zbiór prostokątów i jego reprezentacja w postaci R-drzewa
Fig. 2. A set of rectangles and its R-tree representation




Rys. 3. Zbiór prostokątów i jego reprezentacja w postaci R+-drzewa
Fig. 3. A set of rectangles and its R+-tree representation

Podejście z **obcinaniem** (ang. clipping), w którym dokonuje się podziału przestrzeni danych. Jeśli prostokąt przecina granice podziału, to zostaje przycięty na kilka części i reprezentowany za pomocą wszystkich komórek, które przecina. Np. **R+-drzewo** [7], w którym unika się nakładania się obszarów związanych z komorami lub wewnętrznymi węzłami tego samego poziomu poprzez obcinanie prostokątów, jeśli jest to konieczne. Na rys. 3 pokazano R+-drzewo dla tego samego zbioru prostokątów jak na rys. 2. Prostokąty A, B i C w korzeniu zostały wybrane w trochę inny sposób, i stąd trzy obszary komór w potomku są rozłączne. Wadą podejścia jest konieczność powtórzenia zapisu prostokątów „obciętych" do różnych komór (J, D).

## 2. Indeksacja przestrzenna wykorzystująca R-drzewo

### 2.1. Opis struktury i własności

R-drzewo [4] stanowi strukturę podobną do B+-drzewa [3], która umożliwia zapis wielowymiarowych prostokątów stanowiących aproksymacje obiektów (Mbr). Jest to w pełni dynamiczna struktura danych: operacje wprowadzania nowego węzła, jego usuwania przeplatają się z operacjami wyszukiwania bez potrzeby globalnej reorganizacji struktury. Minimalne prostokąty ograniczające obiekty przestrzenne są zapisane w liściach R-drzewa, natomiast węzły pośrednie powstają poprzez zgrupowanie węzłów na niższym poziomie. Rys.4 pokazuje przykład zbioru danych zorganizowanego w R-drzewo. Węzeł pośredni R-drzewa zawiera elementy, z których każde ma postać: (p, mbr), gdzie p stanowi adres węzła stanowiącego węzeł-potomek danego węzła w tym drzewie, mbr zaś stanowi prostokąt będący minimalnym prostokątem ograniczającym dla wszystkich prostokątów stanowiących wejścia w węźle-potomku wskazywanym przez p. Węzły stanowiące liście drzewa mają wejścia o postaci: (oid, mbr), gdzie oid wskazuje na rekord w bazie danych opisujący dany obiekt przestrzenny, zaś mbr stanowi jego minimalny prostokąt ograniczający. Na oznaczenie prostokąta stanowiącego minimalny prostokąt ograniczający (mbr) dla zbioru podlegających mu prostokątów używa się również nazwy katalog (ang. directory rectangle). Niech M oznacza maksymalną liczbę elementów, jaką może pomieścić jeden węzeł R-drzewa, zaś m stanowi parametr opisujący minimalną liczbę elementów w jednym węźle ($2 <= m <= |M/2|$).

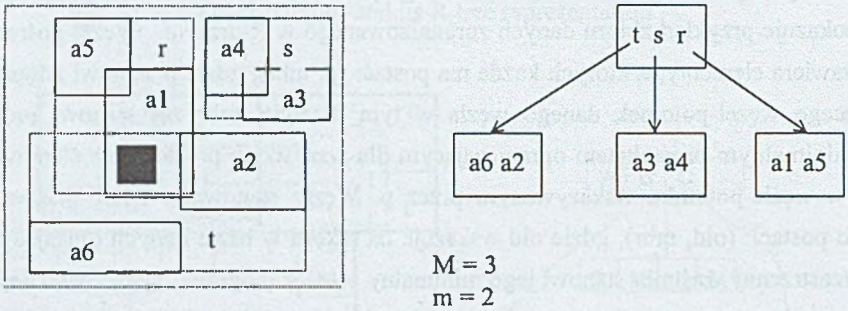
R-drzewo stanowi strukturę o następujących własnościach:

a) korzeń drzewa posiada przynajmniej dwoje potomków,
b) każdy węzeł (oprócz korzenia) posiada ilość potomków w przedziale od m do M,
c) drzewo jest zrównoważone (wszystkie węzły-liście znajdują się na tym samym poziomie),
d) każdy prostokąt węzła pośredniego pokrywa wszystkie prostokąty w odpowiadającym mu węźle-potomku, najczęściej jest to minimalny prostokąt ograniczający.

### 2.2. Wyszukiwanie elementu danych

Niech S stanowi prostokąt w zapytaniu typu window. Realizacja zapytania rozpoczyna się w korzeniu drzewa, gdzie obliczane są wszystkie wejścia, które przecinają się z prostokątem S. Dla tych wejść pobierane są do pamięci operacyjnej węzły potomne i realizacja zapytania (jeśli węzeł nie jest liściem) przebiega dalej jak dla korzenia.

Rys. 4 pokazuje R-drzewo składające się z trzech stron danych i jednej strony katalogu (jeden węzeł drzewa odpowiada jednej stronie w pamięci zewnętrznej). Prostokąty strony katalogu stanowią minimalne prostokąty ograniczające (MBR) prostokątów, które są zapisane w odpowiadającym węźle-potomku. Okno zapytania jest wyróżnione szarym kolorem. Najpierw zapytanie jest wykonywane na korzeniu, w którym prostokąty t i r przecinają się z prostokątem zapytań S. Odpowiednie dwie strony są wczytywane do pamięci i ich wejścia są sprawdzane na przecięcie się z prostokątem zapytań S. W końcu znaleziona zostaje odpowiedź: prostokąt a1 jako jedyny przecinający prostokąt zapytań.




Rys. 4. Przykład R-drzewa z zaznaczonym oknem zapytania
Fig. 4. An example of R-tree with a query window

Ideą algorytmu jest zstępujący podział obszaru przeszukiwania aż do osiągnięcia danych obiektów znajdujących się w liściach struktury. Przebieg procedury przedstawia się następująco:

Search(R,W);

*Input:* R-tree dane korzeniem R oraz okno poszukiwań W

*Output:* Wszystkie obiekty mające część wspólną z W

1. [Przeszukiwanie węzłów pośrednich]

   Jeśli R nie jest liściem, wtedy dla każdego elementu-wejścia R (p,mbr)

   Jeśli mbr $\cap$ W to

   Search(CHILD, mbr $\cap$ W) , gdzie CHILD jest potomkiem wskazywanym przez p

2. [Przeszukiwanie liści]

   Jeśli R jest liściem, wtedy dla każdego elementu-wejścia R (p,mbr)

   Jeśli mbr $\cap$ W to Return (OBJ) , gdzie OBJ jest obiektem wskazywanym przez p.

### 2.3. Usuwanie elementu danych

Usuwanie zbędnego prostokąta odbywa się poprzez zlokalizowanie go (ich) w drzewie i usunięcie go (ich) z liści. Powodem brania pod uwagę więcej niż jednego elementu jest sytuacja, w której niepotrzebny prostokąt znajduje się w kilku węzłach. Funkcja kasująca przedstawia się następująco:

**Delete(R,W);**
*Input:* R-tree dane korzeniem R oraz okno do usunięcia W
*Output:* Nowe R-tree po usunięciu W
1. [Przeszukiwanie węzłów pośrednich]
Jeśli R nie jest liściem, wtedy
dla każdego elementu R (p,mbr)
Jeśli mbr ∩ W to
Delete(CHILD, mbr ∩ W) , gdzie CHILD jest potomkiem wskazywanym przez p
2. [Usuwanie z liści]
Jeśli R jest liściem, wtedy
usuń W z R oraz dostosuj prostokąt rodzica tak by zawierał pozostałe prostokąty-dzieci;

### 2.4. Wprowadzanie nowego elementu danych

Algorytm wprowadzania nowego prostokąta danych (**Wprowadź**) wywołuje dwie procedury: **WybierzWejście**, wybierającą odpowiednią ścieżkę wstawienia, oraz procedurę **Podział**, która dokonuje podziału przepełnionego węzła. Głównym zadaniem procedury **WybierzWejście** jest wybór na każdym poziomie drzewa optymalnej struktury danych uwzględniającej nowo wprowadzany prostokąt danych. Optymalizacja w R-drzewie opiera się na następującym kryterium (**kryterium K1**): pole powierzchni przestrzeni martwej węzła (tzn. nie pokrywającej powierzchni prostokąta danych ) powinno zostać zminimalizowane.

Procedura **Podział**, której zadaniem jest podział M+1 prostokątów pomiędzy dwa węzły w sposób jak najbardziej optymalny, jest wywoływana w przypadku, gdy wprowadzenie nowej danej spowodowałoby przepełnienie węzła (ilość wejść >M). Procedura WybierzCentrum znajduje jądra obu grup, tzn. dwa prostokąty, które przydzielone do jednej grupy spowodowałyby powstanie największej powierzchni martwej węzła, w tym sensie te dwa prostokąty są najbardziej odległe.

**WybierzWejście;**
/wybiera na każdym poziomie poddrzewo - kontynuację ścieżki wstawienia, minimalizującą czynnik K1/
1. N = korzeń drzewa
2. Jeśli N = liść return N

```
        else
                Wej = element-wejście E(p, mbr) ⊂ N  and
                        area(mbr') - area(mbr) = min    and  // mbr' - prostokąt po
                        area(mbr) = min                      // wprowadzeniu nowej danej
        end
3.      N = Child                          // Child - potomek wskazywany przez p
        goto 2.
```

**Podział**

/dokonuje podziału zbioru M+1 prostokątów na dwie grupy g1 i g2/

```
1.      WybierzCentrum
2.      repeat
        2.1     Wej = następny element-wejście
        2.2     if ( area(g1.mbr') - area(g1.mbr) < area(g2.mbr') - area(g2.mbr)
                        Wej ⊂ g1
                else    Wej ⊂ g2
        until
        ( wszystkie wejścia są rozdzielone lub jedna z dwu grup posiada (M-m+1) wejść)
3.      jeśli pozostają jakieś wejścia, to przydziel je do innej grupy, tak aby miała ona
        minimalną liczbę m wejść.
```

**WybierzCentrum**

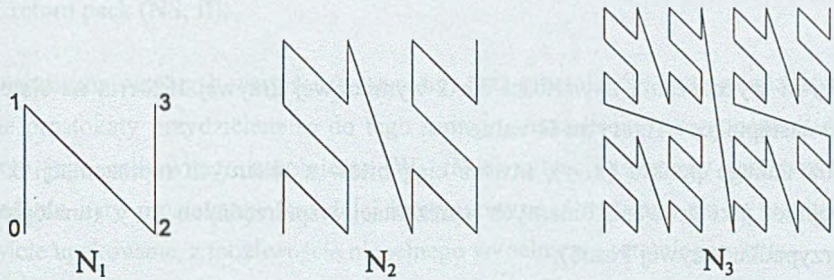
```
1.      Dla każdej pary wejść E1 i E2 utwórz prostokąt R ⊃ (E1.prostokąt ∪ E2.prostokąt).
        d = area(R) - area(E1.prostokąt) - area(E2.prostokąt)
2.      wybierz parę z największym d.
```

## 3. Optymalizacja struktury statycznego R-drzewa

W tej sekcji prezentowany jest algorytm utworzenia R-drzewa o maksymalnym stopniu wykorzystania powierzchni węzłów (ok. 100%), czyli **upakowanego R-drzewa.** Struktura indeksu opartego na R-drzewie, który stanowi w pełni dynamiczną strukturę, gwarantuje minimalne 50% wykorzystanie powierzchni. W przypadku danych o charakterze statycznym (czyli niezmiernie rzadko aktualizowanych), można jeszcze podwyższyć ten parametr. Zwiększenie wykorzystania przestrzeni węzła poprzez dokładne upakowanie daje podwójną korzyść: nie tylko redukuje całkowity rozmiar struktury R-drzewa, ale także zwiększa efektywność wyszukiwania, ponieważ przy zmniejszeniu wysokości drzewa czas realizacji zapytania maleje. Dane statyczne są charakterystyczne dla wielu aplikacji. Na przykład w systemach kartograficznych modyfikacje są bardzo rzadkie, a w bazach danych statystycznych oraz tych publikowanych na CD-ROM'ach nie występują one wcale.

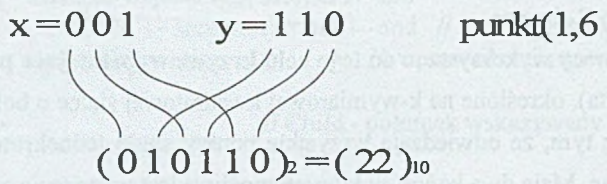
Natomiast rozmiary tych baz danych są olbrzymie i decydującego znaczenia nabiera minimalizacja wielkości struktury indeksu. W takim przypadku pożądane jest utworzenie skompresowanego R-drzewa.

W niniejszej pracy wykorzystano do tego celu **krzywe wypełniające przestrzeń** (krzywe Peano oraz Hilberta), określone na k-wymiarowej kwadratowej siatce o boku 2. Krzywe takie charakteryzują się tym, że odwiedzają wszystkie punkty siatki jednokrotnie, nigdy się przy tym nie przecinając. Mają dwa końce, w których możliwe jest połączenie z innymi krzywymi. Podstawowa krzywa jest rzędu 1. By wyprowadzić krzywą rzędu i, każdy wierzchołek krzywej bazowej jest zastępowany krzywą rzędu i-1, która może być odpowiednio obrócona i/lub odbita. Krzywe wypełniające przestrzeń narzucają liniowe uporządkowanie elementów siatki, które można obliczyć startując z jednego końca i poruszając się wzdłuż krzywej do drugiego końca. Rys. 5 przedstawia krzywą Peano rzędu 1, 2 i 3 dla siatki 2*2. Krzywe wyższych rzędów otrzymuje się poprzez zastąpienie wierzchołków krzywej bazowej ($N_1$) krzywą poprzedniego rzędu bez dodatkowych operacji. Uporządkowanie elementów przestrzeni według tej krzywej nazywa się **z-uporządkowaniem** [17]. Operacja wyznaczenia wartości odpowiadających wierzchołkom krzywej (tzw. **Z-wartość**) zwana jest „bit interleaving" i polega na „przetasowaniu" bitów w binarnej reprezentacji współrzędnych punktu, co pokazuje rys. 6.

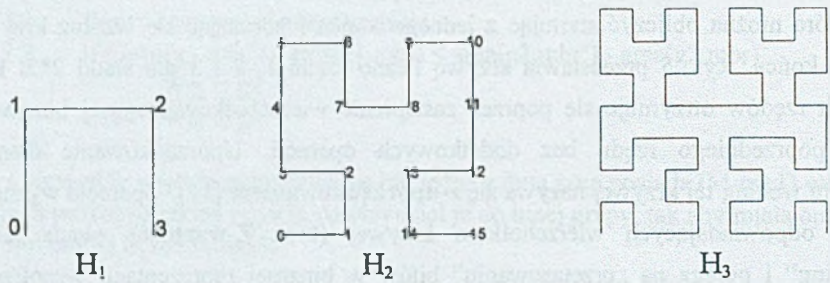



Rys. 5. Krzywe Peano rzędu 1, 2, 3
Fig. 5. Peano curves of order 1, 2, 3

Rys. 7 przedstawia krzywą Hilberta pierwszych trzech rzędów dla siatki 2*2. Procedura generująca krzywe wyższych rzędów dokonuje obrotu krzywej z poprzedniego poziomu w wierzchołkach 0 i 3 krzywej bazowej ($H_1$). Uporządkowanie elementów przestrzeni według tej krzywej nazywa się **h-uporządkowaniem**.




Rys. 6. Wyznaczanie Z-wartości dla punktu krzywej Peano
Fig. 6. Computing of z-value for a point on Peano curve




Rys. 7. Krzywe Hilberta rzędu 1, 2, 3
Fig. 7. Hilbert curves of order 1, 2, 3

Algorytm wyznaczania H-wartości dla 2-wymiarowej krzywej Hilberta na siatce $2^n*2^n$ przebiega następująco /**Algorytm H-value**/

a) Dla danego punktu (x, y), utwórz ciąg bitów z binarnych reprezentacji x oraz y poprzez przetasowanie binarnych reprezentacji współrzędnych x , y (analogicznie do przypadku krzywej Peano);

b) Poczynając od lewej podziel uzyskany ciąg bitów na 2-bitowe sekwencje $s_i$, i = 1,.., N;

c) Dla każdej $s_i$ wyznacz dziesiętne wartości $d_i$ według następującego schematu: '00' - 0, '01' - 1, '10' - 3, '11' - 2, i umieść je w tablicy w kolejności $s_i$ ;

d) Dla każdego $d_i$ w tablicy: Jeśli i= 0, to każde następne wystąpienie 1 zamień na 3, a każde wystąpienie 3 zastąp 1; Jeśli $i_i$ = 3 to każde następne wystąpienie 0 zamień na 2, a każde wystąpienie 2 zastąp 0;

e) Dokonaj konwersji każdej wartości $d_i$ na dwubitową reprezentację, a następnie scal je w jeden ciąg. Dla takiego ciągu bitów wyznacz wartość dziesiętną będącą wartością krzywej Hilberta dla danego punktu.

Efektywność R-drzewa zależy głównie od algorytmu grupującego prostokąty w węzły. Ścieżka krzywej wypełniającej przestrzeń narzuca liniowe uporządkowanie punktów siatki. Na przykład punkt (0,0) krzywej $H_2$ ma h-wartość 0, a punkt (1,1) na tej samej krzywej wartość 2.

Definicja wartości krzywej wypełniającej przestrzeń dla prostokąta jest następująca: „Wartość krzywej wypełniającej przestrzeń dla prostokąta odpowiada wartości krzywej dla jego środka". Najważniejszym założeniem przy tworzeniu struktury upakowanego R-drzewa w niniejszej pracy jest zachowanie własności R-drzewa.. Struktura węzła została rozszerzona o parametr przechowujący wartość krzywej (value): (p, mbr, value).
Procedura budowy indeksu przebiega następująco:

**PACK (S, ff)** // S - zbiór prostokątów
// ff - FillFactor (rozmiar węzła)

a) Wyznacz wartość value krzywej dla każdego prostokąta;
b) sort (S) względem value;
c) Utworzenie węzłów-liści (poziom l = 0): while (| ilość prostokątów w S| > 0) utwórz nowy węzeł i przydziel do niego ff prostokątów;
d) Utworzenie węzłów na poziomie (l+1): while (| ilość prostokątów na poziomie l | > 1) utwórz nowy węzeł na poziomie l przydziel do niego ff prostokątów;
e) return pack (NS, ff);

Posortowana według h-wartości (z-wartości) lista prostokątów jest przeglądana liniowo. Kolejne prostokąty przydzielane są do tego samego węzła-liścia tworzonego R-drzewa do momentu jego całkowitego zapełnienia. W takiej sytuacji tworzony jest nowy węzeł i przeglądanie listy prostokątów jest dalej kontynuowane. Otrzymuje się w rezultacie węzły całkowicie upakowane, z możliwością niepełnego wypełnienia ostatniego węzła.

Porównując dwa zaimplementowane schematy - R-drzewo pakowane krzywą Peano i Hilberta, lepsze rezultaty daje krzywa Hilberta ze względu na bardziej równomierne wypełnianie przestrzeni, mimo iż h-wartość charakteryzuje większa złożoność algorytmu wyznaczania. Słabość Z-uporządkowania uwidacznia się wraz ze wzrostem rzędu krzywej - występują pary kolejnych punktów znacznie od siebie oddalonych. Statyczne R-drzewo pozwala osiągnąć blisko 100% wykorzystanie powierzchni węzłów, co znacznie wpływa również na poprawę efektywności wyszukiwania dzięki zmniejszeniu wysokości drzewa.

## 4. Optymalizacja struktury dynamicznego R-drzewa

### 4.1. Kryteria optymalizacji

Podstawowy problem w R-drzewach jest następujący: „Mając arbitralny zbiór prostokątów Z zbudować dynamicznie z podzbiorów zbioru Z minimalne prostokąty ograniczające (zawierające od m do M prostokątów), tak aby arbitralne zapytania z oknem o rozmiarze arbitralnym mogły zostać zrealizowane efektywnie". Efektywność wykonywania zapytań z wykorzystaniem R-drzewa jako indeksu zależy od tego jak „dobry" jest algorytm przydziału prostokątów do węzłów. Dlatego największy wpływ na efektywność realizacji zapytań ma procedura wprowadzania nowego węzła. Niniejsza sekcja opisuje optymalizację klasycznej struktury R-drzewa [1]. Opiera się ona na dodatkowych kryteriach K1 i K2.

**Kryterium K2.** Część wspólna powierzchni pomiędzy prostokątami w danym katalogu (tzw. współczynnik nakładania się) powinno zostać zminimalizowane.

**KryteriumK3.** Suma długości boków prostokąta katalogu powinna zostać minimalizowana, co prowadzi do kształtu katalogu w postaci jak najbardziej zbliżonej do kwadratu.

Ponieważ R-drzewo stanowi w pełni dynamiczną strukturę, to optymalizacja wymienionych czynników powinna zachodzić przy każdej operacji wstawiania nowego prostokąta danych. Minimalizacja parametrów K1 - K2 wymaga więcej swobody w rozmieszczeniu prostokątów w węźle oraz doboru kształtu prostokątów, a co za tym idzie powoduje gorsze wykorzystanie przestrzeni. Z kolei optymalizacja czynnika K3 powoduje lepsze wykorzystanie przestrzeni, gdyż algorytm pakowania dla elementów o kształcie kwadratu jest bardziej optymalny.

Struktura R-drzewa jest niedeterministyczna, jeśli chodzi o alokację wejść do węzłów (różne sekwencje operacji insert budują różne drzewa). Dlatego dane wprowadzone wcześniej mogą nie być optymalnie ulokowane w sytuacji bieżącej. Aby uniknąć wymienionej „historyczności" struktury zastosowano kasowanie części starej struktury podczas każdej operacji wprowadzania nowej danej i następnie ponowne wprowadzenie usuniętych wejść, umożliwiając tym samym adaptację wszystkich elementów struktury do nowej sytuacji.

Ponieważ wymienione parametry wpływające na efektywność realizacji zapytania oddziałują na siebie wzajemnie i to w sposób bardzo złożony, nie jest raczej możliwa optymalizacja jednego parametru bez wpływu na inny. Ponieważ prostokąty danych mogą być różnej wielkości i kształtu oraz prostokąty katalogowe mogą dynamicznie zmieniać

swoje rozmiary, to optymalizacja jednego parametru jest raczej niepewna. Wynika stąd konieczność stosowania podejścia heurystycznego, opartego na eksperymencie.

### 4.2. Wprowadzanie nowego węzła w zmodyfikowanym R-drzewie

Aby rozwiązać problem wyboru właściwej ścieżki wstawiania w R-drzewie, brane są pod uwagę wszystkie wymienione (K1- K3) kryteria. W trakcie operacji wprowadzenia wejścia, któremu odpowiada prostokąt R,może zaistnieć konieczność wyboru węzła pośredniego, nie zawierającego wejścia, którego odpowiadający mu prostokąt pokrywa całkowicie wprowadzany prostokąt R. W opisywanym dalej algorytmie wybierane jest to wejście, dla którego współczynnik nakładania się po wprowadzeniu R będzie minimalny. Dalej, aby zapobiec zależności od kolejności wprowadzania, nie dokonuje się podziału węzła przepełnionego, lecz pewna część jego elementów jest usuwana i powtórnie wprowadzana (re-insert) na tym samym poziomie w drzewie. Dla algorytmu re-insert wybierane są takie elementy, które są najbardziej odległe od środka oryginalnego MBR-a węzła. Poniżej opisano tę zmodyfikowaną wersję algorytmu.

**WybierzWejście**

1. N = korzeń drzewa
2. Jeśli N jest liściem
   return
   else
   if (wskaźniki potomków w N wskazują na liście)
   **/określ minimalny K1/**
   Wej = element-wejście E(p, mbr) ⊂ N and
   K1' - K1 = min and // K1' - wsp. wyypełn. po
   area(mbr) = min // wprowadzeniu nowej
   then
   Wej = element-wejście E(p, mbr) ⊂ N and
   area(mbr) = min
   if wskaźniki potomków w N nie wskazują na liście
   **/określ minimalny K2/**
   Wej = element-wejście E(p, mbr) ⊂ N and
   area(mbr') - area(mbr) = min and // mbr' - prostokąt po
   area(mbr) = min // wprowadzeniu nowej
   end
3. ustaw N - Child // Child - potomek wskazywany przez p
   repeat 2

**WprowadźWęzeł**

1. WybierzWejście
2. if ( N <= M wejść)
   dostosuj N, aby zawierało nowe wejście E,

if N = M wejść Przepełnienie (N )

3. if Przepełnienie została wywołana i podział został wykonany
propagacja do góry zmian, jeśli konieczne
if Przepełnienie spowodował podział korzenia
to utwórz nowy korzeń,
4. dostosuj wszystkie pokrywające prostokąty na ścieżce wstawiania, tak aby stanowiły mbr-y swoich potomków.

**Przepełnienie**

if (poziom nie jest poziomem korzenia i jest to pierwsze wywołanie Przepełnienie na danym poziomie w czasie operacji insert wprowadzania jednego prostokąta danych)
WprowadźPonownie
Els Przepełnienie
end

**WprowadźPonownie**

1. dla wszystkich M+1 wejść w węźle N oblicz odległość między środkami ich prostokątów i środkiem mbr-a prostokąta N,
2. posortuj wejścia w malejącym porządku odległości obliczonej w 1,
3. usuń pierwsze p wejść w węźle N i dostosuj mbr dla węzła N,
4. w uporządkowaniu zdefiniowanym w 2 startując od minimalnej wywołaj insert, aby dokonać ponownego wprowadzenia wejść.

W pracy przeprowadzono eksperyment polegający na porównaniu efektywności klasycznej struktury R-drzewa (**algorytm 1**) oraz opisanego powyżej wariantu algorytmu (**algorytm 2**). Testy przeprowadzono na przygotowanych kilku zbiorach danych zawierających około 200 000 prostokątów o kilku różnych rozkładach i stosunku długości do szerokości, dla których zrealizowano budowę indeksu i aktualizację według obu wymienionych algorytmów. Jako zapytanie wybrano jedną z najbardziej złożonych obliczeniowo operacji przestrzennych - złączenie. Miernikiem wydajności złączenia pomiędzy dwoma zbiorami prostokątów jest zarówno czas obliczeń mierzony ilością porównań (porównania te występują podczas sprawdzania warunku na przecinanie się prostokątów ograniczających) jak i czas odczytu danych mierzony ilością odwołań do dysku.

Uzyskane wyniki dla operacji złączenia przestrzennego wykazały, że wykorzystanie indeksu w postaci klasycznego R-drzewa (algorytm1) wymaga użycia ok. 147% dostępów do dysku w porównaniu do indeksu zbudowanego według algorytmu zmodyfikowanego (algorytm 2). Tak więc w przypadku stosowania algorytmu 2 maleje znacznie ilość odczytów dyskowych. Uwidacznia to wyższość algorytmu zmodyfikowanego, w którego to przypadku zostaje zachowane uporządkowanie: prostokąty blisko siebie leżące w przestrzeni danych występują z dużym prawdopodobieństwem na jednej stronie dyskowej.

Wymuszona operacja ponownego wprowadzania wymienia wejścia w sąsiednich węzłach zmniejszając współczynnik nakładania się, poprawiane jest wykorzystanie pamięci jako efekt uboczny oraz dzięki lepszej strukturalizacji występuje mniej podziałów. Ponieważ zewnętrzne prostokąty węzła są ponownie wprowadzane, kształt prostokątów katalogu będzie bardziej kwadratowy. Koszt wykorzystania cpu dla zmodyfikowanego algorytmu jest większy, gdyż procedura insert wołana jest więcej razy, ale występuje za to mniej podziałów.

Dzięki zastosowaniu procedury ponownego wprowadzania zwiększa się wykorzystanie przestrzeni, zmniejsza nakładanie się prostokątów, co powoduje, że wydajność praktycznie nie zależy od kolejności wstawień (usunięcie „historyczności" struktury). Również dzięki zastosowaniu wszystkich trzech kryteriów struktura R-drzewa zbudowana zmodyfikowanym algorytmem jest bardziej odporna na różnorodność rozkładów danych.

## 5. Podsumowanie

W artykule opisano przykład optymalizacji heurystycznej klasycznej struktury R-drzewa. Wyniki uzyskano na wygenerowanych danych testowych dla jednej z najbardziej złożonych obliczeniowo operacji przestrzennych - złączenia. W przypadku danych statycznych zastosowanie krzywych wypełniających przestrzeń umożliwia uzyskanie prawie 100% wykorzystania powierzchni węzłów. Maleje dzięki temu wysokość R-drzewa, co w konsekwencji skraca konieczną ścieżkę poszukiwań. W przypadku dynamicznej struktury R-drzewa opisane, dodatkowe kryteria optymalizacji: współczynnik nakładania się oraz sumaryczna długość boków prostokąta umożliwiają uzyskanie struktury R-drzewa dającej w rezultacie lepszą efektywność realizacji zapytań przestrzennych (mniejsza ilość dostępów do dysku). Dzięki zastosowaniu procedury ponownego wprowadzania zwiększa się wykorzystanie przestrzeni, zmniejsza nakładanie się prostokątów, co powoduje, że wydajność praktycznie nie zależy od kolejności wstawień (usunięcie „historyczności" struktury).

## LITERATURA

1. R.H. Guting. An introduction to spatial database systems. The VLDB Journal, 3(4), October, 1994.
2. H. Samet. The design and analysis of spatial data structures. Addison-Wesley, Reading, Massachusets, 1992.

3. J.L. Bentley. Multidimensional binary search trees used for associative searching. CACM, 18(9), Sept. 1975, pp. 509-517.
4. A. Guttman. R-trees: a dynamic index structure for spatial searching. Proc. ACM SIGMOD Int. Conf. on Management of Data, Boston, MA., 1984.
5. J. A Orenstein. Spatial query processing in an object-oriented database system. Proc. ACM SIGMOD Int. Conf. on Management of Data, Washington DC, 1986.
6. J. Nievergelt, H. Hinterberger, K. Sevcik. The grid file: an adaptable, symmetric multikey file structure. ACM Trans. on Database Systems, 9, 1984.
7. T. Sellis, N. Roussopoulos, C. Faloutsos. The R+-tree: a dynamic index for multidimensional objects. Proc. of the 13th Conf. on VLDB, Los Angeles, 1988.
8. B. Seeger, H. Kriegel. Techniques for design and implementation of efficient spatial access methods, Proc. of the 14th VLDB Conf., Los Angeles, California, 1988.

Recenzent: Prof. dr hab. inż. Alicja Wakulicz-Deja

Wpłynęło do Redakcji 20 listopada 1997 r.

**Abstract**

This article describes heuristic optimization of the R-tree structure, which is the popular spatial access method for rectangles. Two kinds of such opimization are presented: static, and dynamic. Presented in the paper static optimization is based on space filling curves. Two packing algorithms for R-tree were given: one based on Hilbert curve (so called h-ordering, see Fig. 7, procedure H-value on page 11) and the other on Peano curve (z-ordering, see Fig. 6). The compactness is very good (about 100% utilisation of node's area) resulting in better performance in query processing due to decreasing a height of a tree.

The proposed optimization for dynamic R-tree is based on two new formulated criteria: overlapping factor (K1) and margin factor (K2) decreasing overlap and overall length of the directory rectangles. The new algoritm for inserting a new data into R-tree was also given. The algorithm is based on forcing insertion of all data recangles after the new one is introduced resulting in better utilisation of node area, independence of the structure of insertion process history. Storage utilisation and average cost insertion for the proposed dynamic R-tree is higher than for classic R-tree structure.