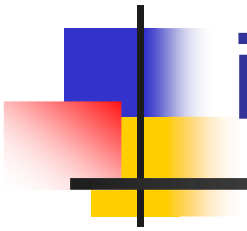
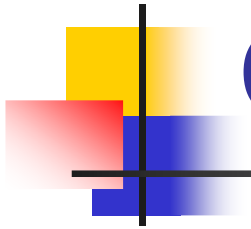


Multidimensionale Indexierung in ORDBMS

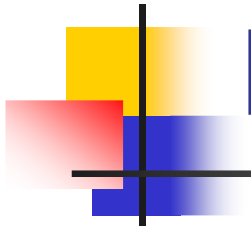
BTW 2007
Studierendenprogramm
Aachen, 6. März 2007

Studienarbeit Gunnar Söllig, Universität Rostock



Gliederung

- Hochdimensionale Indexierung
  - LSD^h-Baum
- Konzept für Relationale Indexierung
- Umsetzung im System IBM DB2
- Ausblick

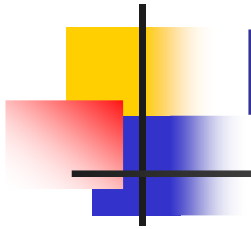


Hochdimensionale Indexierung

- Wahl geeigneter Verfahren erklärt sich aus Anwendungszweck
  - Medizin, CAD, Geographie, Molekularbiologie
  - Clusteringalgorithmen, OLAP, Multimediaobjekte
- Anfragearten
  - Punktanfrage, Bereichsanfragen, Partial match
  - Ähnlichkeitssuche
    - NNB, k-NNB, Epsilon-NNB, Ranking-Anfragen
    - Ähnlichkeitsmaß nicht generell definierbar
- Dead space indexing problem raumorganisierter Verfahren
- „Curse of Dimensionality“
  - Volumen wächst exponentiell in d
  - Fast alle Punkte in Randnähe
  - Indexpartitionen spannen oft gesamten Datenraum/Dimension auf
- Oft ab gewisser Dimensionszahl Sequentieller Scan schneller

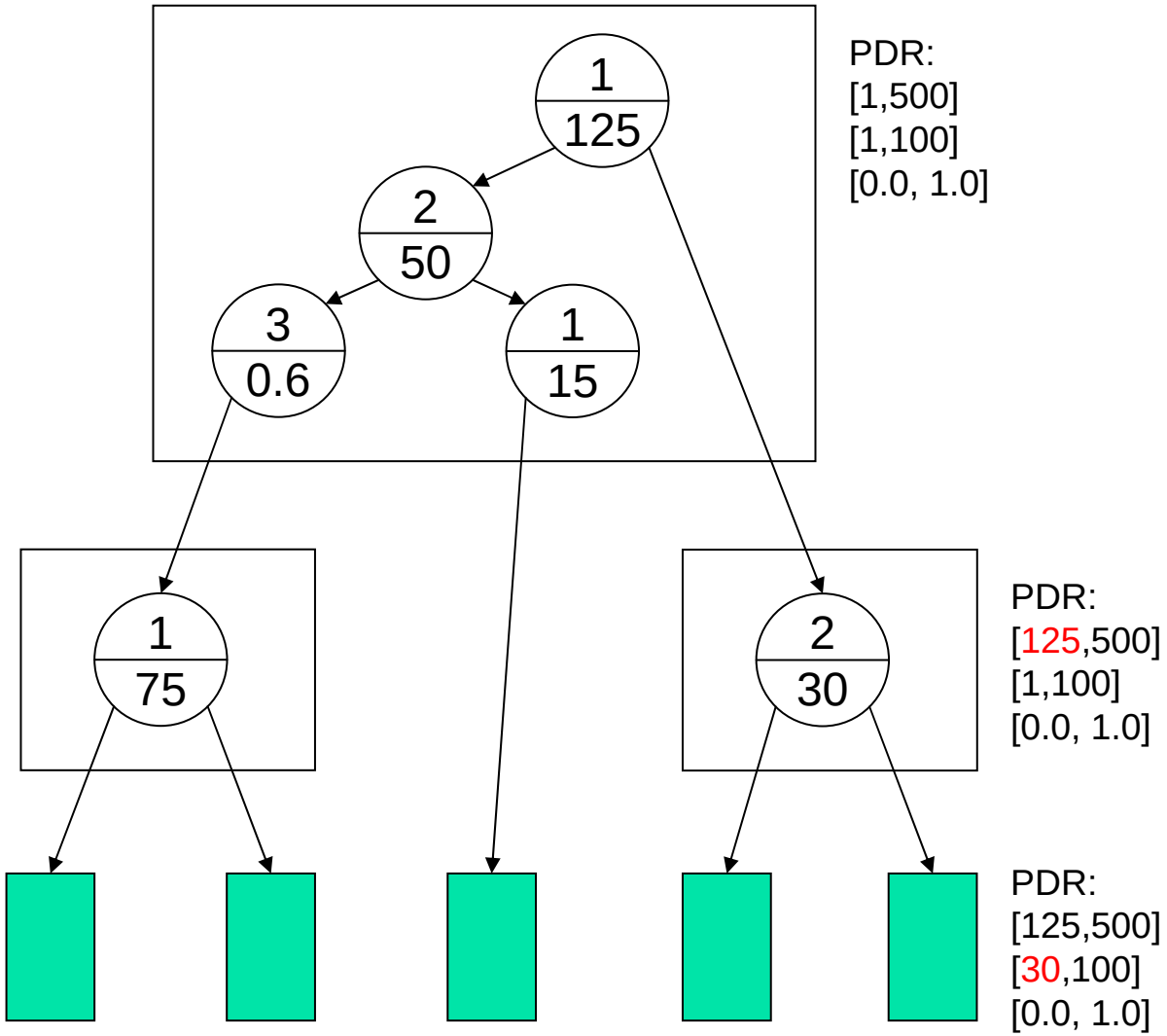
Name	**Probleme bei hohen Dimensionen**	**Anfragen**	**Speicher-ausnutzung**	**Fanout**
R-tree	Splitalg. ungeeignet für hohe Dimensionen	NN, Bereiche	schlecht	schlecht
R*-tree			mittel	
X-tree	Anfrageperformance wg. Überlappung		mittel	
LSD^h-tree	abh. von Verteilung in Splitdimensionen	NN, Bereiche	mittel	sehr gut
SS-tree	hohe Überlappung im Verzeichnis	NN	mittel	sehr gut
TV-tree	nur für spezielle Daten		mittel	schlecht
SR-tree	Verzeichnis sehr groß		mittel	sehr schlecht
SFCs	schlechte Datenraum-Partitionierung	NN, Bereiche	mittel	gut (vgl. B-tree)
Pyramid tree	asymmetrische Anfragen	Bereiche	mittel	gut (vgl. B-tree)

Quelle: Christian Böhm et al., Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases



LSD^h-tree (A. Henrich et al.)

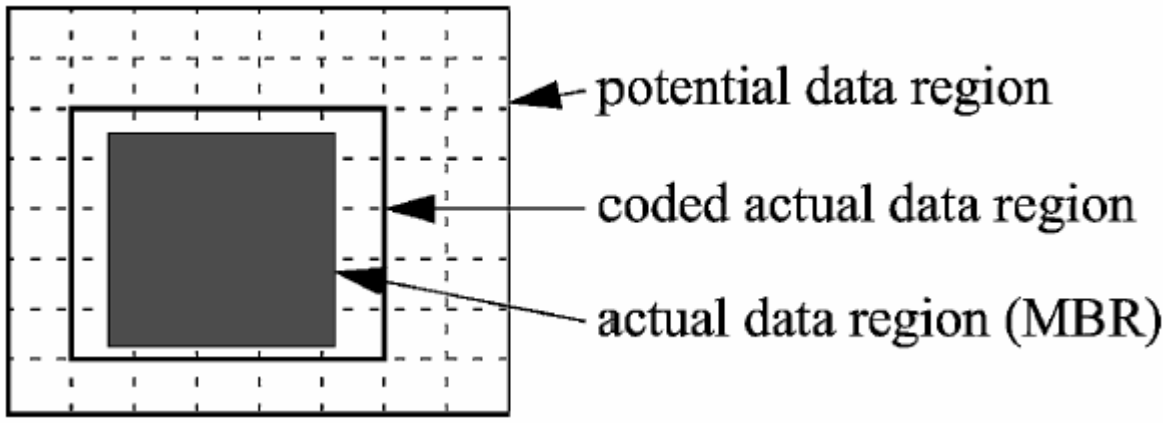
- LSD-tree: lokaler Splitbaum für multidimensionale Objekte
- LSD^h-tree ist definiert durch Vorgehen beim Splitting:
  - Wähle Dimensionen der Reihe nach
  - Überspringe dabei Dimensionen, in denen keine verschiedenen Datenwerte auftreten
  - Modifikation bei hoher Dimensionszahl möglich
  - Splitwert i.d.R. datenabhängig, z.B. arithmetisches Mittel
- Unterstützt Punkt-, Fenster-, Bereichsanfragen
- Ähnlichkeitssuche optimal bzgl. Seitenzugriffszahl für zugrunde liegende hierarchische Struktur (die aber leider nicht zwingend optimal ist)
- Ausgleichsverfahren ähnlich B-Baum-Varianten
  - Paging-Algorithmus
- Erweiterung mittels Coded Actual Data Regions (CADR)



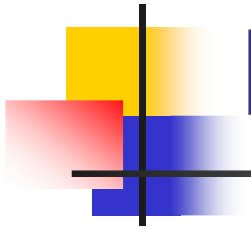
LSD^h-tree: Seitenstruktur und Potential Data Regions

LSD^h-tree: CADRs

- Zerlegung des gesamten Datenraumes einer Region in gleich große Hyperwürfel
- Pro Dimension Speicherung des kleinsten und des größten Indexes besetzter Teilwürfel
- Zusätzlicher Speicheraufwand 2*z*d Bit bei d Dimensionen und Auflösung z

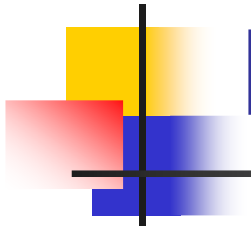


Entnommen aus: Christian Böhm et al., Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases



Relationale Indexierung

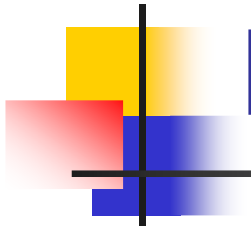
- Alternative Paradigmen
  - Integrierender Ansatz
    - „hartverdrahtet" im DB-Kern, DB-Hersteller
    - Extending Approach vs. Enhancing Approach
  - Generischer Ansatz
    - Nutzung von GiST-Frameworks: Methode der Wahl bei Eignung und Vorliegen des GiST (Generalized Search Tree)
- Vorteile Relationaler Indexierung
  - kein Eingriff in zugrundeliegende DBMS-Schichten
  - geringerer Implementationsaufwand
  - trotzdem konkurrenzfähige Performanz



Relationale Indexierung (2)

- Idee: Aufsetzen auf SQL-Schicht
- Relationale Zugriffsmethode (Kriegel et al.)
  - Indexbezogene Daten ausschließlich mittels Relationen speichern/abfragen
- Relationaler Index
  - User table
  - Index tables
  - Meta table
- Zielstellungen
  - Deklarative Integration
  - Extensible Indexing (OR-Indextyp implementieren)

LSD^h-tree: Konzept Relationaler Indexierung



- Row-IDs bezüglich Nutzertabelle zusammen mit indexierten Featurewerten in relationaler Tabelle speichern
- Indextabelle wird mit B-Baum-Standardindex organisiert
- Weitere Indextabellen simulieren Seitenverwaltung ursprünglicher Zugriffsstruktur (Updateoperationen)
- Implementation von Prädikaten unter Ausnutzung des Index
  - lsdh_match
  - lsdh_range
  - lsdh_nnb

Tabellendefinition

mytab (user table)

id INTEGER	feature_1 DOUBLE	…	feature_k DOUBLE	…

lsd_meta (meta table)

tabno INTEGER	cntfeatures INTEGER	tabname VARCHAR(128)
1	k	mytab

lsd_dirpages1 (index table)

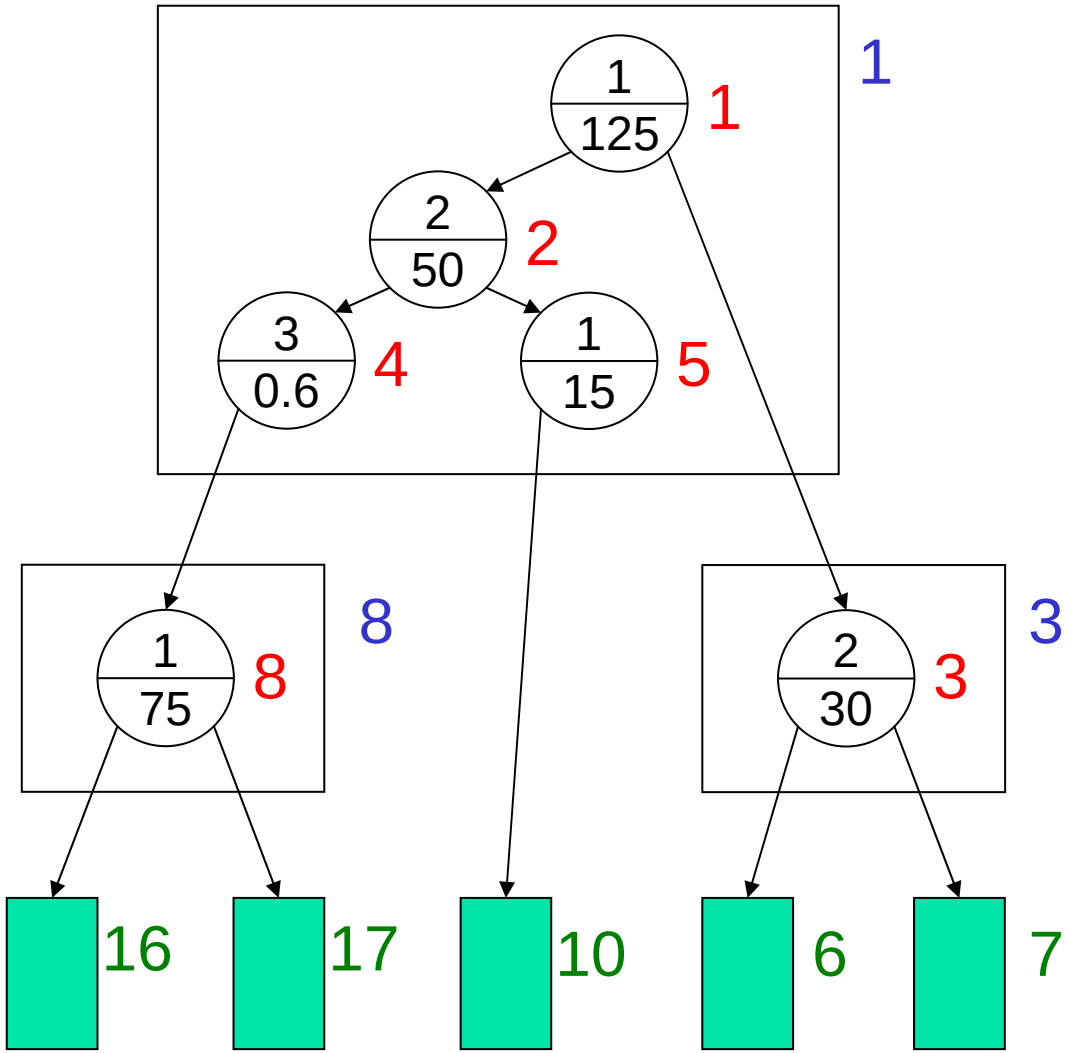
pageno INTEGER	node INTEGER	splitdim SMALLINT	splitval DOUBLE

lsd_dirinfo1 (index table)

pageno INTEGER	leafdist SMALLINT	nodecnt SMALLINT

lsd_leaves1 (index table)

pageno INTEGER	id INTEGER	feature_1 DOUBLE	…	feature_k DOUBLE



lsd_dirpages

pgno	node	sdim	sval
1	1	1	125.0
1	2	2	50.0
1	4	3	0.6
1	5	1	15.0
3	3	2	30.0
8	8	1	75.0

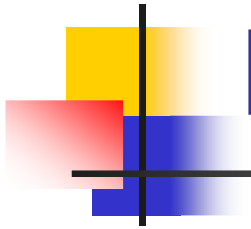
lsd_dirinfo

pgno	leafdist	nodecnt
1	2	4
3	1	1
8	1	1

lsd_leaves

pgno	id	F1	F2	F3
16				
…				

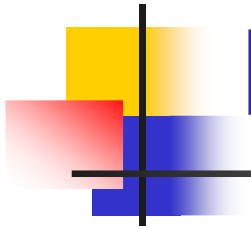
LSD^h-tree: Zusammenhang Knoten und Seiten



Punktanfrage

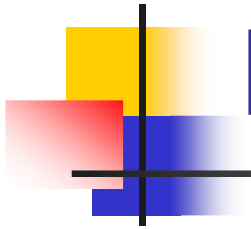
- **function** exact_match (feature: **array**[1..k] **of** featuretype) **returns table**(idtype)

```
1 curPage = 1; curNode = 1;
2 repeat
3       SELECT splitdim, splitval FROM lsddirpages
4       WHERE pageno = curPage AND node = curNode;
5       if foundInfo then
6                   if feature[splitdim] <= splitval then
7                       curNode = 2*curNode
8                   else curNode = 2*curNode + 1
9       elseif curPage <> curNode then begin
10                  curPage = curNode;
11                  continue end
12 until not foundInfo;
13 return SELECT id FROM lsdleaves WHERE pageno = curPage
14 AND feature_1 = feature[1] AND ... AND feature_k = feature[k]
```



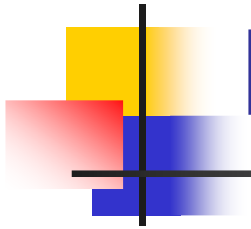
LSD^h-tree: Umsetzung

- DBMS bieten aufbauend auf nutzerdef. Datentypen und Prädikaten Erweiterbare Indexing Frameworks
  - Informix: Data blades
  - Oracle: Data cartridges
- DB2: Extensible Indexing Framework
  - B-Baum-basiert (row-id, key)
  - Nutzer implementiert Schlüsselgenerator zum Einfügen und Rangeproducer für Abfragen
  - Für LSD^h-tree nicht geeignet, da Zugriff keyProducer auf Indexdaten des neuen Datensatzes beschränkt
- Implementation mittels SPs aus C++-basierter DLL



LSD^h-tree: Umsetzung

- Indexupdate
  - Triggerung aller Updateoperationen der Nutzertabelle
  - *lsdh_insert, lsdh_delete*
- Abfragen
  - Nutzung des UDTF-Konzeptes von DB2
  - *lsdh_range, lsdh_nnb,* ...
  - Producer-Consumer-Prinzip
  - Nutzerdefinierte Ähnlichkeitsmaße
    - Euklidische Distanz
    - Maximum-Metrik
    - Gewichtete Distanzfunktionen
- Indexerstellung
  - Erstellung sämtlicher Trigger, Registrierung der Funktionen, Aufbau der Indextabellen gemäß Nutzerdaten
  - *lsdh_create_extension, lsdh_create_index*



Nutzerinterface Abfragen

- FUNCTION matches(tabname VARCHAR,
  f1 DOUBLE, ..., fk DOUBLE)
  RETURNS TABLE(id IDTYPE);
- FUNCTION windowrange(tabname VARCHAR,
  lbound1, hbound1, ..., lboundk, hboundk)
  RETURNS TABLE(id IDTYPE);
- FUNCTION nnb(tabname VARCHAR,
  f1 DOUBLE, ..., fk DOUBLE)
  RETURNS TABLE(id IDTYPE, dist DOUBLE);

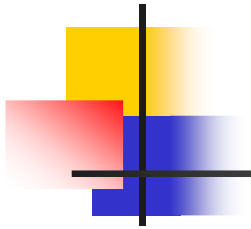
  FUNCTION nnb(k INTEGER, tabname VARCHAR,
  f1 DOUBLE, ..., fk DOUBLE) …

  FUNCTION nnb(tabname VARCHAR, dist VARCHAR, ...) ...

  FUNCTION nnb(k INTEGER, tabname VARCHAR, dist VARCHAR, ...) ...

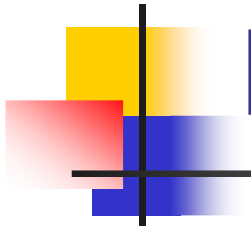
- FUNCTION range(maxd DOUBLE, tabname VARCHAR, ...) ...

  FUNCTION range(maxd DOUBLE, tabname VARCHAR,
  dist VARCHAR, ...) ...



Konzept Extensible Indexing

Extensible Indexing	Implementation LSD^h-Baum
index_create()	lsdh_create_index()
index_drop()	lsdh_drop_index()
index_insert()	lsdh_insert()
insert_delete()	lsdh_delete()
index_update()	lsdh_insert() und lsdh_delete()
index_open()	-
index_fetch()	lsdh_match(), lsdh_window(), lsdh_nnb()
index_close()	-



Beispiel TRIANGLE (1)

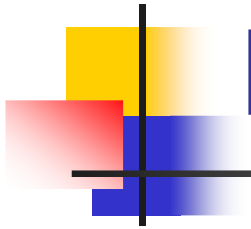
- CREATE TYPE POINT AS (
  x DOUBLE,
  y DOUBLE) MODE DB2SQL@
- CREATE TYPE TRIANGLE AS (
  name VARCHAR(30),
  A POINT,
  B POINT,
  C POINT) MODE DB2SQL@
- CREATE TABLE fig_data (
  nr INTEGER NOT NULL,
  figure TRIANGLE,
  area DOUBLE,
  PRIMARY KEY (nr))@

Beispiel TRIANGLE (2)

- CREATE PROCEDURE
  CREATE_INDEX_EXTENSION_LSDH ()
  EXTERNAL NAME
  'lsdh_01_140.dll!lsdh_create_extension'
  LANGUAGE C
  PARAMETER STYLE SQL
  FENCED
  NOT THREADSAFE
  MODIFIES SQL DATA@

- CALL CREATE_INDEX_EXTENSION_LSDH@

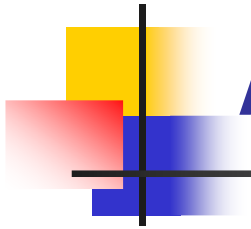
- CALL CREATE_INDEX_LSDH_ON
  ('fig_data(nr, figure(A(x), B(x), C(x)), area)')@



Beispiel TRIANGLE (3)

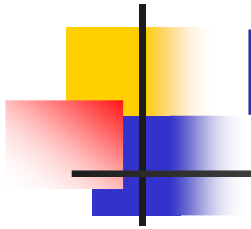
- CREATE FUNCTION mydist (
  Ax1 DOUBLE, Bx1 DOUBLE, Cx1 DOUBLE, Area1 DOUBLE,
  Ax2 DOUBLE, Bx2 DOUBLE, Cx2 DOUBLE, Area2 DOUBLE)
  RETURNS DOUBLE
  RETURN Abs(Sqrt(Area1)-Sqrt(Area2))@

- SELECT result.dist, figure..name, area
  FROM fig_data, TABLE(nnb(3, 'fig_data', 'mydist',
  1.0, 3.0, 2.0, 2.0)) AS result
  WHERE fig_data.nr = result.id ORDER BY 1@



Ausblick

- Mögliche Erweiterungen der LSD^h-tree-Implementation
  - Erweitertes Ausgleichsverfahren zur Verhinderung der Baumdegeneration
  - Nutzung von CADRs statt PDRs
  - Abschätzungsfunktionen Seitenzugriff
- Diplomarbeit (laufend)
  - Überführung LSD^h–tree in eine Oracle Data Cartridge
  - Weitere Zugriffsstrukturen (Pyramid tree)
  - Untersuchung von GiST-Konzepten



Literatur

- Christian Böhm, Stefan Berchtold, Daniel A. Keim: Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases; In ACM Computing Surveys, Vol. 33, No. 3, 2001
- J. M. Hellerstein, J. F. Naughton, A. Pfeffer: Generalized Search Trees for Database Systems; In Proceedings 21st International Conference on VLDB, 1995
- Andreas Henrich: The LSD^h-tree: An Access Structure for Feature Vectors; In Proceedings 14th International Conference on Data Engineering, Orlando 1998
- Hans-Peter Kriegel, Martin Pfeifle, Marco Pötke, Thomas Seidl: The Paradigm of Relational Indexing: A Survey; In LNI – Tagungsband der 10. BTW-Konferenz, Leipzig, 2003