A Hybrid Split Strategy for k-d-Tree Based Access Structures

Andreas Henrich
Universität Siegen, Praktische Informatik, D-57068 Siegen, Germany
henrich@informatik.uni-siegen.de
http://www.informatik.uni-siegen.de/pi/user/henrich

Abstract

With spatial access structures the split strategy determining how the objects stored in an overflowing bucket are distributed over two buckets is crucial for the performance and robustness. Two categories of split strategies can be distinguished: *Data dependent split strategies* depend only on the objects stored in the bucket to be split. *Distribution dependent split strategies* choose the split dimension and the split position independently of the objects actually stored in the bucket to be split based on an hypothesis about the object distribution. Unfortunately both types of split strategies have specific drawbacks. For a *distribution dependent split strategy* a sound hypothesis for the spatial distribution of the objects is needed in advance and with *data dependent split strategies* the directory tree becomes extremely unbalanced for insertions in a geometrically presorted order.

In this paper we present a hybrid split strategy which tries to combine the advantages of both types of split strategies. Normally it adapts the behavior of a data dependent split strategy, but when the risk of a degeneration of the access structure is detected, it mutates to a distribution dependent split strategy. We will show that this hybrid split strategy is robust with respect to skew distributions and with respect to insertions in presorted order.

1 Introduction

Various efficient multi-dimensional access structures have been proposed. The focus of most of these structures has been on the maintenance of spatial objects, but they can be used as multi-attribute access structures as well [HM95]. Besides the well-known R-tree family [Gut84, SRF87, BKSS90] and various hash-based access structures (like the grid file [NHS84], the bang file [Fre87] or the buddy-tree [SK90]) a lot of these access structures use a variant of the k-d-tree [Ben75] as directory (see e.g. [Rob81, OMSD87, LS89, HSW89, KO91]). A survey of these access structures is given in [GG95].

As usual for secondary storage access structures these structures store the objects in buckets of fixed size. Each bucket is associated with a dynamically defined part of the data space called its bucket region. Initially there is only one bucket, for which the bucket region corresponds to the whole data space.

If a new object has to be inserted into a bucket which is already filled, the bucket has to be split. To this end a split line is determined, which is used as the criterion to decide for each object in the overflowing bucket whether it should remain in the bucket or moved into its newly created sibling. Thereafter, the new split line has to be inserted into the directory maintaining the partition of the data space induced by the split lines.

Obviously the choice of the split line is crucial for the performance and robustness of the access structure. To determine the split line a split strategy has to compute the split dimension and the split position.

For spatial objects it is usual to use the dimensions of the data space on each path in the directory in cyclic order as split dimensions. The computation of the split position is a little bit more complicated. Two categories of split strategies can be distinguished:

Data dependent split strategies depend only on the objects stored in the bucket to be split. An example is to use the average over all object coordinates with respect to the split dimension as split position.

Distribution dependent split strategies choose the split dimension and the split position independently of the actual objects stored in the bucket to be split. An example based on the assumption of a uniform distribution of the objects is to split a cell into two cells of equal areas.

Since data dependent split strategies calculate the split line based on the objects actually stored in the access structure, on the one hand, they yield a good adaptation of the data space partition. On the other hand, the data space partition depends on the order of insertion and degenerates especially if the objects are inserted in a geometrically presorted order.

In contrast, the data space partition induced by a distribution dependent split strategy is independent of the order of insertion. The drawback of distribution dependent split strategies is, that a hypothesis about the object distribution must be found in advance [HS91].

However, since the worst case situations for both types of split strategies are different (skew object distributions in one case and presorted insertion in the other) a natural improvement might try to combine the advantages of both split strategies. To this end, a hybrid split strategy would be desirable which behaves like a distribution dependent split strategy if the objects are inserted in presorted order and like a data dependent split strategy if the objects are skewly distributed. The most important question in this respect is: How can the split strategy determine which behavior is best suited to the actual situation?

In this paper we present two approaches in the depicted direction. Both split strategies normally behave like a data dependent split strategy to adapt well for skew object distributions. Whenever the split strategy detects a significant imbalance in the directory, it smoothly mutates into a distribution dependent split strategy to avoid the risk of a degeneration of the directory tree caused by insertions in presorted order.

Although all methods proposed in this paper can be applied to all multi-dimensional access structures which use a variant of the *k-d*-tree as directory, we will use the LSD-tree (*LSD* stands for *local split decision* in this context) as an example for such an access structure in the following in order to be as precise as possible. Furthermore the experimental results which will be given in section 4 have been achieved using an LSD-tree.

2 Basic concepts of the LSD-Tree

As usual for geometric access structures which support spatial access to point objects the LSD-tree divides the data space into pairwise disjoint data cells. With every data cell a bucket of fixed size is associated, which stores all objects contained in the cell. In this context a data cell is often called *bucket region.*

Initially, the whole data space corresponds to one bucket. After a certain number of insertions the initial bucket has been filled, and an attempt to insert an additional object causes the need for a bucket split. To this purpose, a *split line* is determined and the objects on one side of the split line are stored in one bucket, while those on the other side are stored in another bucket. After some further insertions, the capacity of another bucket will be exceeded. In this case, a split line in the corresponding bucket region is determined, thus splitting this region into two subregions. This process is repeated each time the capacity of a bucket is exceeded.

For example, consider the 2-dimensional data space

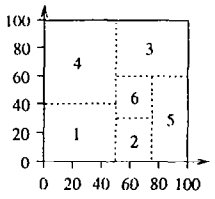

Figure 1: Possible data space partition for an LSD-tree

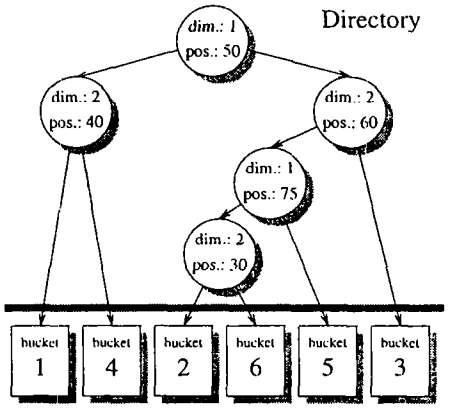

Figure 2: The LSD-tree associated with the data space partition of figure 1

in figure 1. The first split was made according to coordinate 50 in the first dimension, giving buckets 1 and 2. Then bucket 2 was split according to position 60 of the second dimension, giving buckets 2 and 3, and so on.

The split lines of the LSD-tree are maintained in a directory which is a generalized *k-d*-tree [Ben75]. For each split, a new node containing the position and the dimension of the split line is inserted into the directory tree. The leaves of the directory tree reference the buckets in which the actual objects are stored. For example, the partitioning displayed in figure 1 results in the directory tree displayed in figure 2.

Usually, the directory grows up to a point where it cannot be kept in main memory any longer. In this case, subtrees of the directory are stored on secondary memory, whereas the part of the directory near the root remains in main memory. For the details of the paging algorithm we refer to [HSW89].

To describe the **processing of a range query** on an LSD-tree, we have to introduce the term of a *data region.* The *data region* of a bucket is its bucket region, and the *data region* of a directory node is defined as the union of the two data regions of its sons. As a consequence, a data region in an LSD-tree is always an axis-parallel rectangle.

During a range query searching all points geometrically located in a given query region q, the directory of the LSD-tree is traversed from the root to the buckets. To start with, the whole data space is considered to be the data region of the root itself. Thereafter, the data regions of the sons of a directory node can be calculated from the data region of their father and the split line stored with the father. Hence, we can decide at each node if the data region of the left and/or right son intersects q. If the data region of exactly one son intersects q, we follow this son. If the data regions of both sons intersect q, we choose an arbitrary son to continue and store the other son on an auxiliary stack. When the search reaches a bucket, all found objects geometrically located in q are reported. If there are directory nodes or buckets in the auxiliary stack, query processing is continued taking the first directory node or bucket from the stack. Otherwise query processing is finished.

3 Hybrid Split Strategy

In this chapter we present two split strategies which try to detect imbalances in the directory tree in order to change their behavior in this case.

3.1 Weighted Average

With the first split strategy the split position is calculated as the weighted average of the split positions calculated using a data dependent split strategy and a distribution dependent split strategy assuming a uniform distribution of the objects.

In the following we assume a bucket capacity of b objects. Furthermore we assume that a bucket w has to be split and that the selected split dimension is dimension d. Let $R(w) = (w_{\lambda_1}, w_{v_1}] \times \ldots \times (w_{\lambda_k}, w_{v_k}]$ denote the bucket region of w and let B denote the set containing the b objects stored in w and the object which has to be inserted – i.e. the object which caused the bucket split. Furthermore let $o[i]$ denote the coordinate of an object $o \in B$ in dimension i. Then the split position p_{data} for the data dependent split strategy can be calculated as

$$p_{data} = \frac{1}{b+1} \cdot \sum_{o \in B} o[d] \qquad (1)$$

and the split position p_{dis} for the distribution dependent split strategy can be calculated as

$$p_{dis} = \frac{w_{\lambda_d} + w_{v_d}}{2} \qquad (2)$$

As mentioned the weight for p_{dis} in our hybrid split strategy should be high if there is a risk for a degeneration of the directory tree because of presorted insertions. Otherwise the weight for p_{data} should be high.

The obvious question is now: How can we detect situations where the directory tree becomes unbalanced due to presorted insertions? We decided to use a simple heuristics for this purpose: If the path from the root of the directory to the bucket w which has to be split is relatively long compared to the other paths in the directory, the weight for p_{dis} in our hybrid split strategy is increased, because the path to w might have become relatively long because of presorted insertions.

To achieve this effect, let β denote the number of buckets in the access structure and let l denote the length of the path from the root of the directory to w. Then the height of an optimally balanced directory tree would be $\lceil \log_2 \beta \rceil$.

Using this formula we calculate

$$p_{hybid} = \alpha \cdot p_{data} + (1 - \alpha) \cdot p_{dis} \qquad (3)$$

with

$$\alpha = \begin{cases} 1 & \text{if } 2 \geq l - \lceil \log_2 \beta \rceil \\ \frac{l - \lceil \log_2 \beta \rceil - 2}{5} & \text{if } 2 < l - \lceil \log_2 \beta \rceil < 7 \\ 0 & \text{if } 7 \leq l - \lceil \log_2 \beta \rceil \end{cases}$$

This yields a split strategy which behaves like a data dependent split strategy as long as the path to the bucket to be split is at most two references longer than the paths of an optimally balanced tree would be. For longer paths the split strategy smoothly mutates into a distribution dependent split strategy and if the path to the bucket to be split is seven or more references longer than the paths of an optimally balanced tree, the split strategy behaves like a distribution dependent split strategy.

Of course the parameters of this split strategy – i.e. the bounds of two and seven are relatively arbitrary, but we performed tests with other settings and found that the influence of these parameters on the shape of the access structure is relatively low as long as they remain in a reasonable range.

3.2 Limited Redistributions

In [Hen96] we proposed a redistribution schema for k-d-tree based access structures to improve the storage utilization and the range query performance of the access structures. This redistribution schema works as follows:

If a bucket cannot accommodate an additional entry, we try to fix this situation by the means of a local redistribution to avoid a bucket split. To this end, we consider the sibling of an overflowing bucket, i.e. the other son of the node in the directory tree referencing the overflowing bucket. If this sibling is a bucket which can take an additional element without a bucket split, a local redistribution is performed shifting the object which is located closest to the split line from

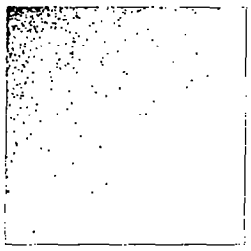

Figure 3: Corner distribution

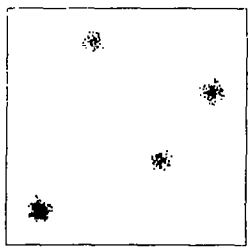

Figure 4: Multi-heap distribution

Figure 5: Zipcodes in the United States

the overflowing bucket into its sibling and adjusting the split line in the corresponding directory node.

Using this redistribution schema the split line originally determined by the split strategy is only preliminary. With each redistribution the split line is adjusted. Hence, the final position depends on the data stored in the overflowing bucket and in its sibling. As a consequence an access structure using this redistribution schema is sensitive against presorted insertions even if the split lines are originally determined using a distribution dependent split strategy.

Hence, it seems to be a good idea to stop the redistributions for those paths which are in danger to degenerate due to presorted insertions. To this end we implemented a limited redistribution technique which performs redistributions only if the path from the root of the directory to the bucket w which has to be split is short enough.

Again, let β denote the number of buckets in the access structure and let l denote the length of the path from the root of the directory to w. Then redistributions are performed for the limited redistribution technique only if $3 < l - \log_2 \beta$.

4 Experimental Results

In order to cover a wide range of application profiles, we used five data sets for the experiments:

1. 250,000 uniformly distributed 2-dimensional points inserted in random order.
2. 250,000 uniformly distributed 2-dimensional points presorted with respect to their distance to the point $(0, 0)$.
3. 250,000 2-dimensional points generated according to a *corner distribution* which is illustrated for a random sample of 500 objects in figure 3. For this distribution all points tend to be located in the upper left corner. We inserted these points in random order.
4. 250,000 2-dimensional points generated according to a *multi-heap distribution* which is illustrated for a random sample of 500 objects in figure 4. For each heap the points have been inserted in random order, but the heaps have been inserted in sequence, i.e. we started inserting the objects in the lower left heap, thereafter we inserted the objects in the lower right heap, ...
5. 34,265 2-dimensional points representing the locations of the zipcodes in the United States given in latitude and longitude (downloaded from `http://tiger.census.gov/places.html`). The points have been inserted in the given order where on the top level the states have been presorted in alphabetical order and for each state the cites in alphabetical order. The distribution of the points in this data set is illustrated in figure 5.

For our experiments the bucket capacity has been 5 and the maximum number of internal directory nodes has been 500. We used 512 bytes per directory page such that external directory pages could accommodate subtrees up to a height of 6.

Table 1 states the storage utilization, the directory size and the construction time for the structures under concern. The first column has been achieved using a distribution dependent split strategy assuming a uniform distribution of the objects as given in equation (2). The second column has been achieved using a data dependent split strategy as given in equation (1). The basic redistribution technique described in section 3.2 has been used for column three and the redistribution schema with limited redistributions for column four. Finally the hybrid split strategy defined in equation (3) yielded the results in column five.

	distribution dependent	data dependent	redistribution of objects	limited redistribution	hybrid split strat.
uniform distribution (250000 objects)					
bucket utilization	69.2 %	73.4 %	77.5 %	75.2 %	73.2 %
height of directory	20	22	22	22	22
external directory pages	3421	3480	3023	3530	3436
external height of directory	2	2	2	2	2
construction time in sec.	591	632	724	759	590
insertion in presorted order (250000 objects)					
bucket utilization	69.2 %	67.0 %	86.9 %	69.6 %	69.4 %
height of directory	20	838	708	34	37
external directory pages	3421	**8601**	**6189**	3463	3607
external height of directory	2	**56**	**34**	2	2
construction time in sec.	445	3654	3118	621	585
multi-heap distribution (250000 objects)					
bucket utilization	69.1 %	73.3 %	77.4 %	71.7 %	70.9 %
height of directory	28	46	46	41	41
external directory pages	3661	3592	3065	3853	3722
external height of directory	2	2	2	2	2
construction time in sec.	746	763	983	814	794
corner distribution (250000 objects)					
bucket utilization	67.9 %	73.3 %	77.0 %	74.6 %	72.7 %
height of directory	45	24	23	24	36
external directory pages	5074	3536	3070	3746	3674
external height of directory	2	2	2	2	2
construction time in sec.	671	595	729	735	669
zipcodes (34265 objects)					
bucket utilization	63.8 %	71.9 %	76.3 %	62.9 %	64.1 %
height of directory	48	58	60	54	52
external directory pages	1066	1100	997	1080	1063
external height of directory	2	2	2	2	2
construction time in sec.	65	68	97	77	69

Table 1: Storage utilization and directory size

To gain a visual impression of the shape of the directory tree and of the data space partition achieved with the compared access structures, we generated figures 6, 7, 8 and 9 depicting the resulting data space partitions and directory trees for 500 inserted objects created according to the first four of our test data distributions.

Especially the problems arising from insertions in presorted order with the data dependent split strategy and with the original redistribution schema become obvious from figure 7 and the corresponding numbers in table 1 which show a height of 838, resp. 708, for the directories of these access structures. Another indicator for these problems is the external height of the directory, i.e. the maximal number of directory pages passed by a path from the root of the directory to a bucket. Whereas this external height is 2 for all other access structures, with presorted insertions it is 56 for the data dependent split strategy and 34 for the original redistribution schema.

A comparison with the numbers given for the redistribution schema with limited redistributions and for the hybrid split strategy in table 1 shows that the access structures based on these split strategies have no significant problems with presorted insertions. Hence, we have achieved the objective to prevent degenerations with presorted insertions at least for this data set.

To depict how the hybrid split strategy adapts to the different data sets, we have stated the influence of the data dependent split position on p_{hybrid} in table 2. A

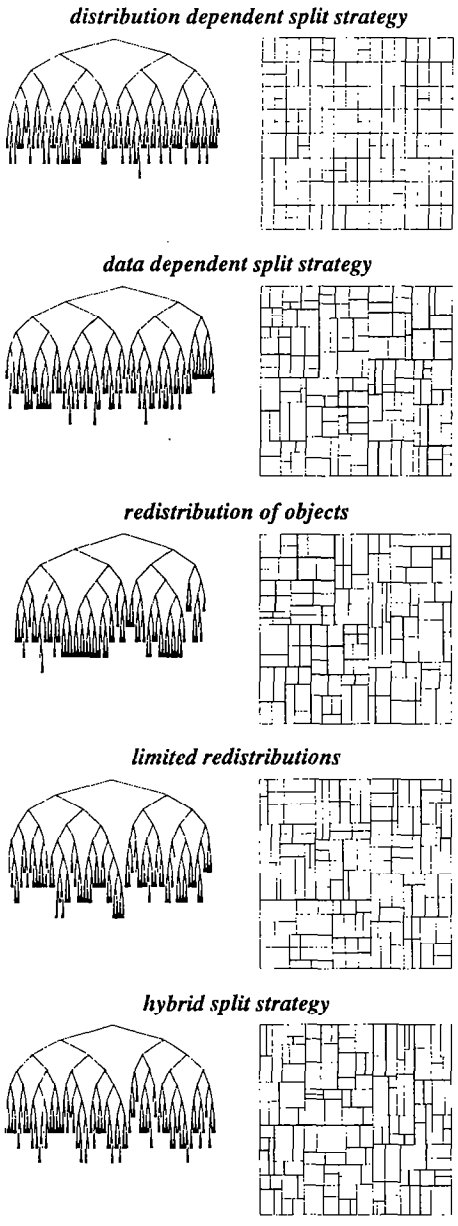

Figure 6: Resulting data space partitions and LSD-trees for the uniform distribution

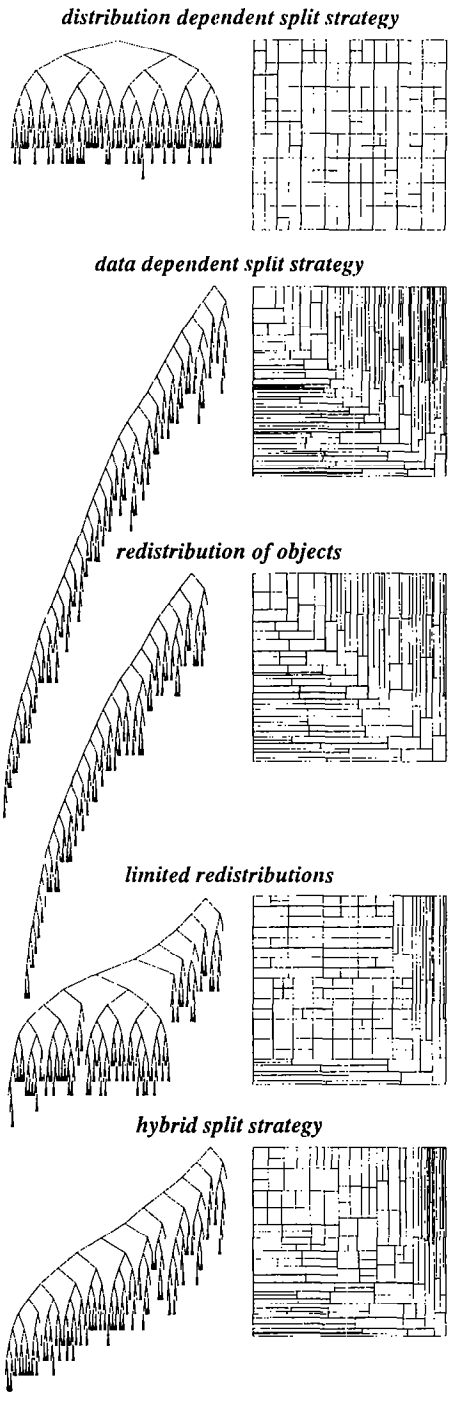

Figure 7: Resulting data space partitions and LSD-trees for presorted insertions

comparison of the rows for the uniform distribution and for the sorted insertions stresses that the hybrid split strategy works as intended for these data sets. If the objects are not presorted, the split strategy behaves like a data dependent split strategy for 53420 of all 68272 splits in the access structure, and with presorted insertions the split strategy behaves like a distribution dependent split strategy for over 90 % of all splits.

Furthermore the numbers given for the corner distribution in table 1 and the access structures shown in figure 8 point out that limited redistributions and our hybrid split strategy perform better than a distribution dependent split strategy assuming a uniform distribution if the points are skewly distributed.

But these are the clear cut data sets, which are relatively easy to handle. What about the multi-heap distribution and the zipcodes, which are both partly

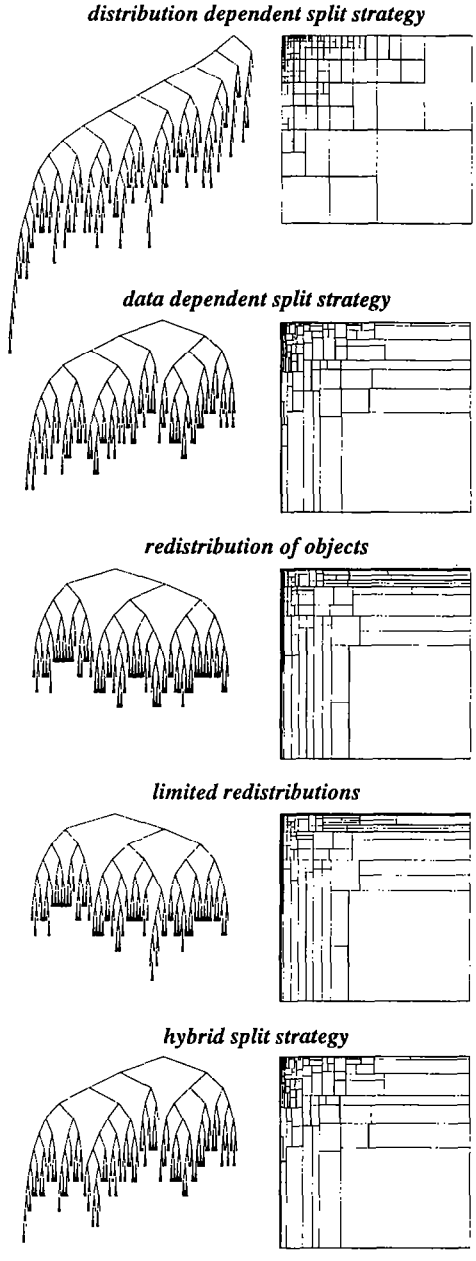

Figure 8: Resulting data space partitions and LSD-trees for the corner distribution

presorted data sets? Obviously the situation is more difficult here. The split strategy might decide for a long path, which might have become long because of presorted insertions, to behave like a distribution dependent split strategy. However, the points for each heap as well as the zipcodes of each state are inserted in a random order with respect to there geometrical location. As a consequence, the hybrid split strategy and the limited redistributions often make erroneous

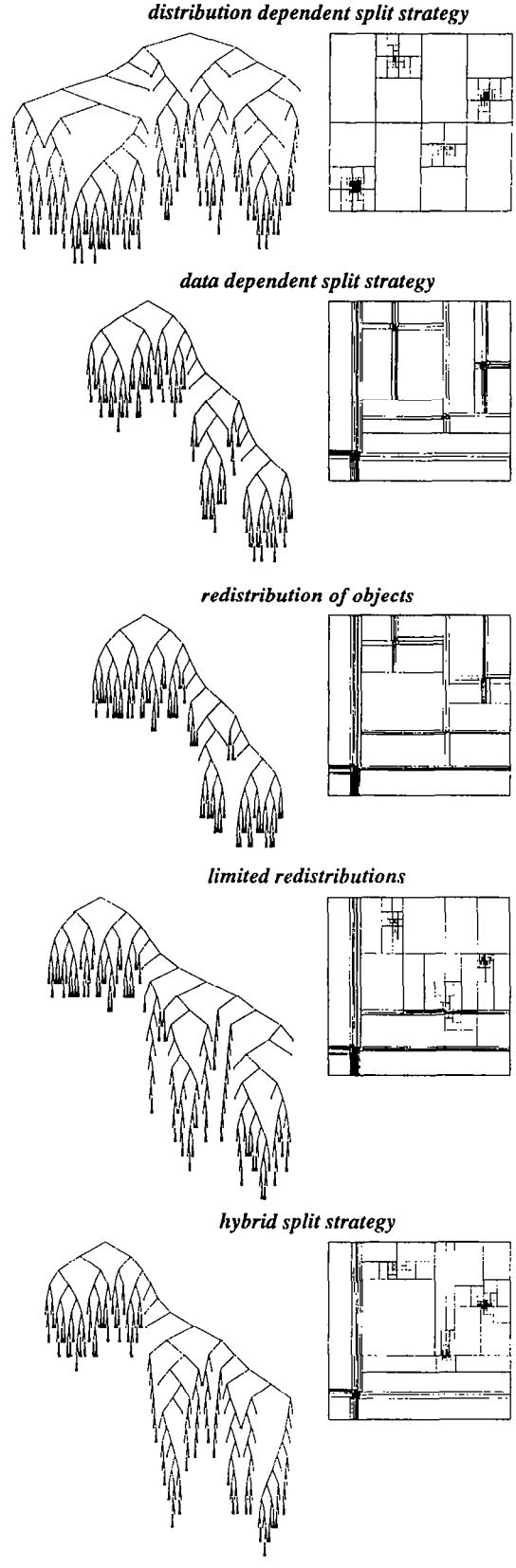

Figure 9: Resulting data space partitions and LSD-trees for the multi-heap distribution

distrib.	*influence of the data dependent split position* 1.0	0.8	0.6	0.4	0.2	0.0
uniform	53420	11246	3180	405	20	1
sorted	610	400	472	577	833	69123
multi-h.	23674	7301	2727	585	292	35897
corner	40483	14117	8812	2953	1169	1252
zipcodes	938	399	529	689	805	7324

Table 2: Influence of the basic split strategies on p_{hybrid}

decisions with these data sets. They become confused by the fact that the points are inserted heap after heap and state after state. Unfortunately, this seems to be a practically relevant way to insert points into a data set. However, even with these inauspicious data sets the hybrid split strategy and the limited redistribution schema do not perform significantly worse than the data or the distribution dependent split strategy.

5 Conclusion

We have presented a hybrid split strategy and a limited redistribution schema which try to combine the advantages of data and distribution dependent split strategies. The presented experimental results show that this aim has been achieved to a satisfactory degree. In contrast to data dependent split strategies, the access structure does not degenerate for insertions in presorted order, and for skew object distributions the adaptation of the access structure is better than for a distribution dependent split strategy assuming a uniform distribution of the objects. However, the experimental results show also that the achieved results are not completely satisfactory for *bulk-sorted* insertions. More sophisticated techniques to detect the risk of a degeneration might be an interesting field of research in this direction.

Nevertheless, the use of the hybrid split strategy or the limited redistribution schema yields an access structure which is extremely robust with respect to skewly distributed objects and insertions in presorted order and can be beneficial for all *k-d*-tree based access structures.

References

[Ben75] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, 1990.

[Fre87] M. Freeston. The BANG file: a new kind of grid file. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 260–269, San Francisco, 1987.

[GG95] Volker Gaede and Oliver Günther. Survey on multidimensional access methods. Technical Report ISS-16, Humboldt Universität zu Berlin, Wirtschaftswissenschaftliche Fakultät, Institut für Wirtschaftsinformatik, August 1995.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, Boston, 1984.

[Hen96] A. Henrich. Improving the performance of multi-dimensional access structures based on k-d-trees. In *Proc. IEEE 12th Int. Conf. on Data Engineering*, pages 68–75, New Orleans, Louisiana, USA, February 1996.

[HM95] A. Henrich and J. Möller. Extending a spatial access structure to support additional standard attributes. In M. J. Egenhofer and J. R. Herring, editors, *Proc. 4th Int. Symposium on Advances in Spatial Databases (SSD'95)*, volume 951 of *LNiCS*, pages 132–151. Springer, 1995.

[HS91] A. Henrich and H.-W. Six. How to split buckets in spatial data structures. In *Proc. Int. Conf. on Geographic Database Management Systems*, Esprit Basic Research Series DG XIII, pages 212–244, Capri, 1991. Springer.

[HSW89] A. Henrich, H.-W. Six, and P. Widmayer. The LSD-tree: spatial access to multidimensional point and non point objects. In *Proc. 15th Int. Conf. on VLDB*, pages 45–53, 1989.

[KO91] M.J. van Kreveld and M.H. Overmars. Divided k-d Trees. *Algorithmica*, 6:840–858, 1991.

[LS89] D.B. Lomet and B. Salzberg. A Robust Multi-Attribute Search Structure. In *Proc. IEEE 5th Int. Conf. on Data Engineering*, pages 296–304, 1989.

[NHS84] J. Nievergelt, H. Hinterberger, and K.C. Sevcik. The Grid File: an adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, 1984.

[OMSD87] B.C. Ooi, K.J. McDonell, and R. Sacks-Davis. Spatial kd-Tree: An Indexing Mechanism for Spatial Databases. In *IEEE COMPSAC*, pages 433–438, 1987.

[Rob81] J.T. Robinson. The K-D-B-Tree: A Search Structure for Large Multdimensional Dynamic Indexes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 10–18, 1981.

[SK90] B. Seeger and H.-P. Kriegel. The buddy-tree: an efficient and robust access method for spatial data base systems. In *Proc. 16th Int. Conf. on VLDB*, pages 590–601, Brisbane, 1990.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-tree: a dynamic index for multi-dimensional objects. In *Proc. 13th Int. Conf. on VLDB*, pages 507–518, 1987.