Optimizing Spatial Data Structures For Static Data*

Lukas Bachmann, Bernd-Uwe Pagel, Hans-Werner Six

University of Hagen

Abstract. During the last decade various spatial data structures have been designed and compared against each other, all of them reflecting a dynamic situation with ongoing object insertion and deletion processes. Assuming the frequently occurring situation where the data is known in beforehand and using as performance measure the expected number of data bucket accesses needed to perform a window query, the original dynamic clustering problem turns into a classical optimization problem. For the special case of bucket capacity $c_b = 2$ we present a mapping onto the well-known graph matching problem. For the general case of $c_b \geq 3$ and arbitrary cost functions, the optimization problem is NP-hard. In first experiments with simulated annealing heuristics the best dynamic structures are outperformed by more than 25%. However, we understand our contribution as a lower bound result rather than another speed-up variant of classical spatial data structures.

1 Introduction

During the last decade various spatial data structures have been designed and compared against each other, all of them reflecting a highly dynamic situation with ongoing object insertion and deletion processes. Much less effort has been devoted to the static situation where the data is known in beforehand and insertions, resp. deletions, happen rather rarely. This situation frequently occurs in geographical applications. To our knowledge, so far only one approach addressing the static situation exists in the literature, the so-called *Packed R-tree* [3]. In a preprocessing step the objects are arranged to a space filling curve, e.g. the Hilbert curve, and then an R-tree is built bottom-up, similar to the bottom-up construction of an optimal B-tree. As a result, space utilization of the R-tree is high, its height is small and – also because of the clustering effect of the space filling curve – the Packed R-tree outperforms conventional R-trees mainly for large range queries.
Using the expected number of data bucket accesses needed to perform a window query as performance measure, the static clustering problem turns out to be a classical optimization problem. The solution of the optimization is a set of data buckets with given bucket capacity inducing an optimal data clustering w.r.t. the performance measure.

* This work has been supported partly by the European Community, ESPRIT II Project No. 6881 (AMUSING).

In this paper, we solve the bucket optimization problem for bucket capacity $c_b = 2$ and provide some simulations where dynamic LSD-trees are outperformed by about 15%. We sketch the general situation with bucket capacity $c_b \geq 3$. For arbitrary cost functions, the optimization problem turns out to be NP-hard. If the cost function fulfills a certain monotony property which holds for our cost measure, we conjecture but have no proof yet that the problem ist still NP-hard. Using simulated annealing as optimization heuristic, first simulation results exhibit an improvement over the best dynamic data structures by more than 25%. We would like to emphasize that we understand our contribution rather as a lower bound result than another speed-up of the classical dynamic spatial data problem.

2 The Optimization Problem

In order to state the static optimization problem precisely, we introduce a general framework. Let d be the dimension of the data space, $S_i = [0, 1)$, $1 \leq i \leq d$, and $S = S_1 \times S_2 \times \ldots \times S_d$ be the d-dimensional data space in which all geometric objects are defined. For sake of simplicity, we assume that each geometric object g is a d-dimensional interval $g = [g.l_1, g.r_1] \times \ldots \times [g.l_d, g.r_d]$, $g.l_i, g.r_i \in S_i$, $g.l_i \leq g.r_i$. For a point object $g.l_i = g.r_i$, $1 \leq i \leq d$, holds and the representation is abbreviated to $g = (g.l_1, g.l_2, \ldots, g.l_d)$. Let us assume that for storing a set $G = \{g_1, \ldots, g_n\}$ of n objects the data structure $DS(G)$ currently consumes m data buckets $B_1, B_2, \ldots, B_m$ with a capacity of c_b objects each. With each object $g \in G$, a bucket is uniquely associated. The bucket region $R(B_i) \subseteq S$ of a bucket B_i is the minimal d-dimensional interval enclosing all objects in B_i. We call $B = \{B_1, \ldots, B_m\}$ a bucket set and $R(B) = \{R(B_1), \ldots, R(B_m)\}$ the corresponding organization of the data space.

Without loss of generality and only for simplicity reasons, we choose $d = 2$ for further considerations. This reduces bounding boxes, bucket regions, and query windows to two-dimensional rectangles.

Reasoning about optimal data structures starts with a cost function that allows for comparing one structure to another. Especially, a fair measure is desirable, which is independent of data structure and implementation details. We claim that the expected number of bucket accesses needed to perform a query is such a fair cost measure, because in practical applications data bucket accesses dominate query costs (e.g. in particular exceed by far external accesses to the paged parts of the corresponding directory concerning frequency and execution time).

Obviously, the cost measure depends on the actual bucket set B and the query behavior of the user. Since we are interested in expected values, we have to define the underlying probability model. To this end, we introduce a probabilistic query model $\mathcal{QM}$ reflecting the expected query behavior of the users.

For a bucket set B and a query model $\mathcal{QM}$, let $P(q \text{ meets } B_i)$ be the probability that performing query q forces an access of bucket B_i. Then the expected number of bucket accesses needed to perform query q – we

call it the *performance measure* $\mathcal{PM}$ *for* $\mathcal{QM}$ – is given by

$$\mathcal{PM}(\mathcal{QM}, B) = \sum_{i=1}^{m} P(q \text{ meets } B_i).$$

Example 1. In order to illustrate the performance measure $\mathcal{PM}(\mathcal{QM}, B)$ let us take its "instantiation" derived from a simple query model $\mathcal{QM}_1$, which is based on square windows of constant size on the average. Furthermore, every part of the data space is equally likely to be requested. The assumptions of this query model reflect a behavior which can sometimes be observed with novice and occasional users.
More precisely, $\mathcal{QM}_1$ is defined by a constant aspect ratio of 1:1, a constant window area c_A and a uniform window center distribution U. If we abstain from boundary considerations in favour of readability, then $P(q \text{ meets} B_i)$ is just the bucket region $R(B_i)$ of bucket B_i inflated by a frame of width $\sqrt{c_A}/2$. Let $R(B_i).L$ describe the width and $R(B_i).H$ the height of $R(B_i)$. Then we get (see Fig. 1)

$$\begin{aligned}\mathcal{PM}(\mathcal{QM}_1, B) &= \sum_{i=1}^{m} (R(B_i).L + \sqrt{c_A}) \cdot (R(B_i).H + \sqrt{c_A}) \\ &= \sum_{i=1}^{m} R(B_i).L \cdot R(B_i).H \\ &\quad + \sqrt{c_A} \cdot \sum_{i=1}^{m} (R(B_i).L + R(B_i).H) + c_A \cdot m.\end{aligned}$$

In $\mathcal{QM}_1$, one possible user behavior is modelled. For other kinds of user behavior and the corresponding query models the interested reader is referred to [8].
We are now able to define the *Bucket Set Problem* (*BSP*):
Given a set of geometric objects, a bucket capacity $c_b \geq 2$, and a query model $\mathcal{QM}$, determine the bucket set B for which the performance measure $\mathcal{PM}$ is minimal.

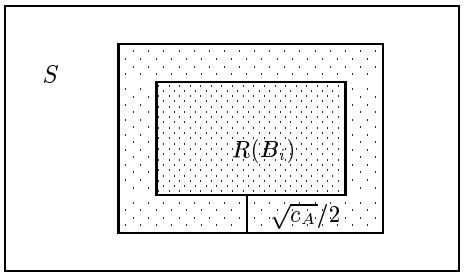

Fig. 1. $P(q \text{ meets} B_i)$

3 The Simple Bucket Set Problem

We start our investigations with the *simple Bucket Set Problem (SBSP)* where we assume bucket capacity 2 and query model $\mathcal{QM}_1$; B_{opt} denotes the optimal bucket set. In this case, the *BSP* shows strong similarities to the well-known *Minimum Weighted Matching Problem (MWMP)*, which is defined as follows:
Given a graph G with weigthed edges, determine a subset of edges such that

i) for each vertex v in G there exists exactly one edge e in the subset such that e is incident to v, and
ii) the sum of the edge weights is minimal.

In the minimal subset no two edges are incident to the same vertex. We call the two endpoints of an edge in the subset a *match*. In other words, each vertex belongs to exactly one match and the sum of the match weights is minimal.
The similarity of *SBSP* and *MWMP* becomes intuitively clear if we regard two geometric objects sharing the same bucket as a match while the weight of the match is determined by the value of the cost function $\mathcal{PM}$ for that bucket.
We now explain the mapping of the *SBSP* to a *MWMP* for nonbipartite graphs in more detail. For a given set of geometric objects $G = \{g_1, \ldots, g_n\}$ we obtain the *corresponding (weigthed) graph* $WG = (V, E, w)$ as follows:

$$V = G \cup A,$$

where $G = \{g_1, \ldots, g_n\}$ is the set of geo-vertices and $A = \{a_1, \ldots, a_n\}$ is the set of artificial vertices,

$$E = \{(g_i, g_j) \in G^2 | i \neq j\} \cup \{(g_i, a_i) \in G \times A\} \cup \{(a_i, a_j) \in A^2 | i \neq j\},$$

and

$$w : E \rightarrow \mathcal{R}_0^+$$

is the weighting function depending on $\mathcal{QM}_1$, where conditions 1. - 3. hold:

1. Let $e = (g_i, g_j) \in G^2$ and B be the bucket storing g_i and g_j.
   Then $w(e) = P(q \text{ meets } B)$.
2. Let $e = (g_i, a_i) \in G \times A$ and B be the bucket storing only g_i. (Note that the bucket region $R(B)$ degenerates to object g_i.)
   Then $w(e) = P(q \text{ meets } B)$.
3. $e = (a_i, a_j) \in A^2$.
   Then $w(e) = 0$.

The basic idea behind the graph construction is as follows: The introduction of artificial vertices and edges is necessary to cope with the situation that buckets may contain single objects. Edges of the first subset of E mirror the situation that the two connected objects belong to the same bucket, while edges of the second subset model single object buckets. The last subset contains auxiliary edges which have no semantics within the *SBSP*, but provide the basis for a meaningful matching in the *MWMP*.

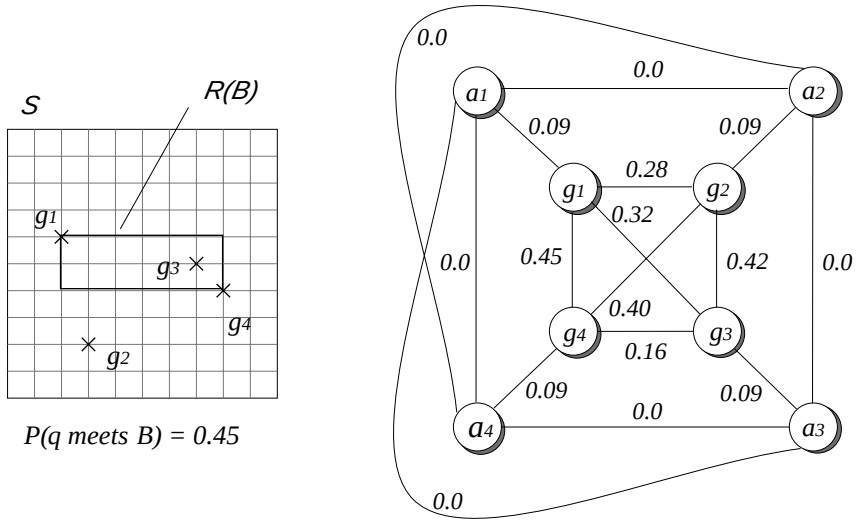

Fig. 2. A *BSP* and its corresponding graph.

Example 2. To illustrate the mapping let us consider a set G of four point objects $g_1 = (0.2, 0.7)$, $g_2 = (0.3, 0.2)$, $g_3 = (0.7, 0.5)$, and $g_4 = (0.8, 0.4)$. Assuming $\mathcal{QM}_1$ with window area $c_A = 0.09$, i.e. each window covers 9% of the data space, we get the corresponding graph depicted in Fig. 2. Then, for example, the weight of a match of g_1 and g_4 equals the probability that bucket B storing g_1 and g_4 is accessed by a random window query.

Theorem 1. *The SBSP is equivalent to the MWMP for the corresponding graph* $WG = (V, E, w)$.

Proof. We just sketch the main steps of the proof and refer the interested reader to [6] for more details.
We get a solution of the *SBSP* from a solution of the corresponding *MWMP* by interpreting every match $g_i, g_j)$ of two geo-vertices as a bucket storing g_i and g_j, and every match (g_i, a_i) of a geo-vertex and an artificial vertex as a bucket containing only object g_i. Every match (a_i, a_j) of artificial vertices is neglected. From the construction of the corresponding weighted graph it is obvious that a *SBSP* solution and a corresponding *MWMP* solution are equally valuated.

□

Example 3. Let us return to the situation presented in example 2. Fig. 3 depicts an optimal matching and the corresponding bucket set $B_{opt} = \{B_1, B_2, B_3\}$ where B_1 stores g_1, B_2 stores g_2 and B_3 stores g_3 and g_4. We have $P(q \text{ meets } B_1) = 0.09$, $P(q \text{ meets } B_2) = 0.09$, $P(q \text{ meets } B_3) = 0.16$, and hence $\mathcal{PM}(\mathcal{QM}_1, B_{opt}) = 0.34$.

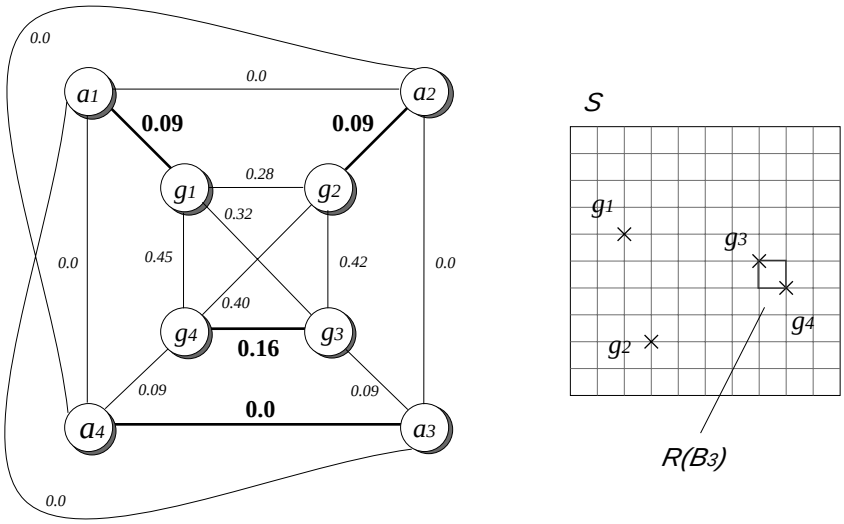

Fig. 3. An optimal matching and its corresponding bucket set B_{opt}.

The *MWMP* for nonbipartite graphs can be solved in $O(n^3)$ time [4]. The transformation itself can be done in $O(n^2)$ time.

4 Experimental results

For practical reasons, we have just implemented an $O(n^4)$ variant of the nonbipartite *MWMP* algorithm and compared various optimal bucket sets B_{opt} and bucket sets B_{LSD} generated by an LSD-tree [2]. The LSD-tree can be regarded as a representative of dynamic spatial data structures and is fully competitive to other structures like the R-tree and its variants [1, 7]. [2]

In order to provide a visual impression of the data space organizations created by the optimal algorithm, resp. the LSD-tree, we start with a small set of 75 "uniformly distributed" rectangles. The overlapping factor is 1.0, i.e. the sum of all rectangles equals the data space area. The window area covers 1% of the data space. Fig. 4 depicts the original data set and the resulting data space organizations. Notice how smoothly the optimal bucket regions adapt to the input rectangles. The visual impression is reflected by the technical evaluation: the result of the optimal algorithm superceeds that of the LSD-tree by nearly 13%.

[2] All data structures and algorithms have been implemented in Eiffel [5] on a SUN SPARCstation 10.

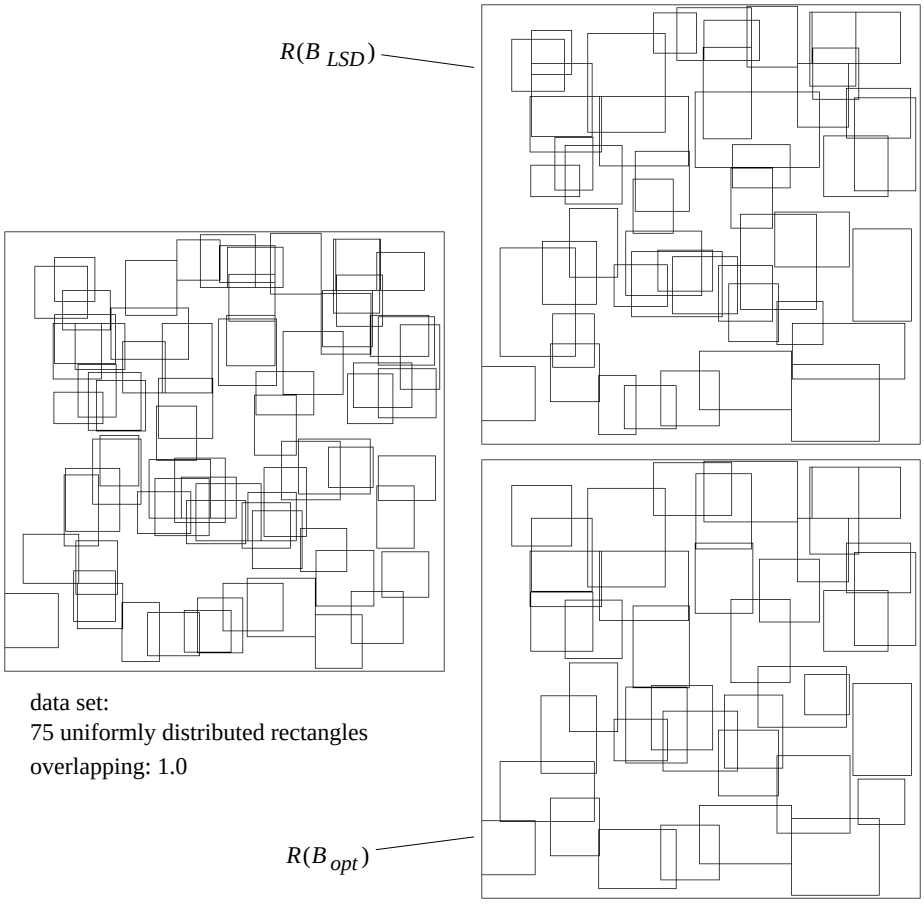

Fig. 4. Visual comparison of B_{opt} and B_{LSD} for 75 rectangles and $c_b = 2$.

In the following experiments we use four different data sets, namely 200 points, resp. rectangles, which are distributed uniformly or follow a 2-heap distribution. The sum of the rectangle areas equals the data space area. The results are presented in Table 1. They exhibit a first tendency, namely the more complex the data set the higher the improvement gained by the optimal algorithm.

We now address the obvious question what happens if other range queries are performed than assumed, for example, if the expected value of the real window area differs from the window area which has been assumed for the optimization. Fig. 5 shows that even in this case B_{opt} substantially superceeds B_{LSD} for a wide range of (expected) window areas, if a medium-sized window, the "1% window", has been chosen for the optimization.

Summing up, we emphazise that even for our simple and small data sets and for bucket capacity 2, B_{opt} outperforms B_{LSD} by up to nearly 20%. However, because of the $O(n^3)$, resp. $O(n^4)$, time complexity, the opti-

Window area 1%	Point data		Rectangular data	
	uniform	2-heap	uniform	2-heap
$\mathcal{PM}(\mathcal{QM}_1, B_{LSD})$	1.6365	1.5893	10.5546	7.3371
$\mathcal{PM}(\mathcal{QM}_1, B_{opt})$	1.4383	1.3561	9.1035	6.0875
improvement in %	12.11	14.67	13.75	17.03

Table 1. Cost comparisons of B_{LSD} and B_{opt} for 200 points, resp. rectangles, and $c_b = 2$.

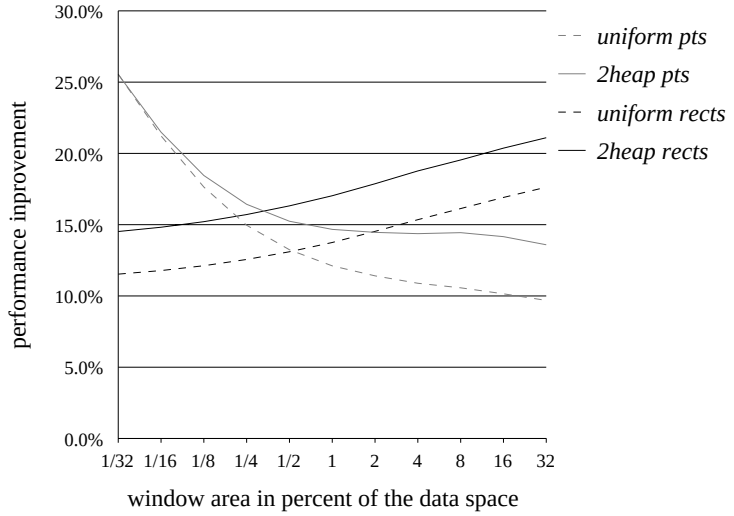

Fig. 5. Relative improvement of B_{opt} over B_{LSD} for variant window areas. The 1% window is used as optimization parameter.

mal algorithm cannot be used in real applications with thousands of objects. Furthermore, the next section exhibits that the optimization problem gets even harder if we choose a more realistic bucket capacity $c_b \geq 3$.

5 The Universal Bucket Set Problem

So far we have considered the *BSP* for bucket capacity 2 and $\mathcal{QM}_1$. We now turn to the general situation with bucket capacity $c_b \geq 3$ and arbitrary cost function C fulfilling the only assumption that it can be computed in polynomial time. While the specialized *BSP* can be solved in $O(n^3)$, the *universal BSP (UBSP)* is *NP-hard*.

Theorem 2. *The UBSP is NP-hard.*

Proof. The proof is rather technical and goes beyond the scope of this paper. We only sketch the general outline. (see [6] for more details). We restrict the *UBSP* to $c_b = 3$ and show that the *3-D ASSIGNMENT PROBLEM* can be reduced to the *UBSP*.
We explain this problem for the 2-dimensional case. Given a $n \times n$-matrix of real numbers, determine a subset of elements such that

i) each row and each column is covered by exactly one element, and
ii) the sum of the elements is minimal.

Obviously, this problem is equivalent to the *MWMP* for complete graphs and hence can be solved in $O(n^3)$. We now indicate the reduction of the 3-dimensional problem to the *UBSP*. Let $I = \{i_1, \ldots, i_n\}$, $J = \{j_1, \ldots, j_n\}$, and $K = \{k_1, \ldots, k_n\}$ denote the three index sets of the $n \times n \times n$-matrix. The main idea of the transformation is to interpret each index $i \in I \cup J \cup K$ as a geo-object, giving an object set G with $|G| = 3n$. Furthermore, each matrix element represents the cost of a bucket storing the three objects given by the three indices of the matrix element. With suitable completions concerning buckets with less than three objects, the *3-D ASSIGNMENT PROBLEM* can be reduced to the *UBSP*. Since the *3-D ASSIGNMENT PROBLEM* is NP-hard [4] and the reduction can be performed in polynomial time, the *UBSP* is NP-hard.

□

It is an open problem, whether the *UBSP* remains NP-hard, if the cost function C fulfills the following monotony property: Let B_i be an arbitrary bucket which can accommodate at least one further object, then $C(\{B_i\}) \leq C(\{B_i \cup \{g\}\})$ holds for each object g. Note that for any query model QM, our performance measure $\mathcal{PM}$ fulfills the monotony property.
Since the NP-hardness of the *UBSP* prohibits an exact optimization procedure, we have started to implement a heuristic based on simulated annealing. First results exhibit an improvement of more than 25% over the LSD-tree. The experiments indicate the tendency the more complex the situation (large bucket capacity and data sets with high overlapping factor and biased distribution) the higher the improvement (see [6] for more details).

6 Conclusion

We have shown that in case of static data sets the bucket optimization problem can be solved in time $O(n^3)$ for bucket capacity $c_b = 2$, and is NP-hard for $c_b \geq 3$. Using a simulated annealing heuristic first results exhibit a performance improvement of more than 25% over the best dynamic spatial data structures. For complex situations, an even higher improvement can be expected. However, we would like to emphasize that we understand our contribution rather a lower bound result than another speed-up of the classical dynamic problem.

Main problem to be solved is whether the *UBSP* is also NP-hard for monotonic cost functions. (Remember that our performance measure is monotonic.) We strongly conjecture that the monotonic *UBSP* remains NP-hard.
Although the time penalty incured by external directory accesses is small compared to data bucket accesses, it would be desirable (at least from a theoretical viewpoint) to extend the performance measure and the optimization efforts to cover external directory accesses as well. Usually, with each directory page a directory region is associated which is the bounding box of all data bucket regions pointed to from the directory page. Since the directory page regions again form a data space organization, such an integrated approach seems to be feasible.
Another problem concerns the algorithm which builds an optimal directory on top of the optimal bucket set B_{opt}. An R-tree will be an obvious candidate, because an R-tree provides the highest freedom for clustering data buckets from B_{opt} into directory pages. Basically, there exist two possible strategies. The first strategy builds the directory of the R-tree bottom-up from the optimal bucket set B_{opt} by level-oriented recursive calls of the optimization procedure. The second strategy creates the directory on the fly, i.e. during the bucket optimization process. The best strategy and its efficiency is still an open problem.

References

1. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 47–57, Boston, 1984.
2. Henrich, A., Six, H.-W., Widmayer, P.: The LSD-tree: spatial access to multidimensional point- and non-point objects. In: 15th Int. Conf. on VLDB, pp. 45–53, Amsterdam, 1989.
3. Kamel, I., Faloutsos, C.: On packing r-trees. In: Proc. 2nd Int. Conf. on Information and Knowledge Management, pp. 490–499, Washington D.C., 1993.
4. Lawler, E.L. Combinatorial Optimization: Networks and Matroids. Holt, Rhinehart and Winston, 1976.
5. Meyer, B. Eiffel: The Language. Prentice Hall, 1991.
6. Pagel, B.-U., Six, H.-W.: On statically optimal spatial data structures. Technical report, University of Hagen, in preparation.
7. Pagel, B.-U., Six, H.-W., Toben, H.: The transformation technique for spatial objects revisited. In: Abel, D., Ooi, B.-C. (eds.): Proc. 3rd Int. Symposium on Large Spatial Databases (SSD), pp. 73–88, Singapore, June 1993. Lecture Notes in Computer Science No. 692, Springer Verlag.
8. Pagel, B.-U., Six, H.-W., Toben, H., Widmayer, P.: Towards an analysis of range query performance in spatial data structures. In: Proc. ACM 12th Symposium on Principles of Database Systems (PODS), pp. 214–221, Washington, D.C., May 1993.