Window Query-Optimal Clustering of Spatial Objects*

Bernd-Uwe Pagel
Department of Computer Science
University of Hagen
bernd-uwe.pagel@fernuni-hagen.de

Hans-Werner Six
Department of Computer Science
University of Hagen
hw.six@fernuni-hagen.de

Mario Winter
Department of Computer Science
University of Hagen
mario.winter@fernuni-hagen.de

Abstract

During the last decade various spatial data structures have been designed and compared against each other with respect to their performance. Still missing is a lower bound result, e.g. an optimal spatial data clustering, which would allow for the absolute comparison of the performance of the well-known data structures with the optimum. In this paper, we address the static situation where the data is known in beforehand. An optimal data clustering for this setting will also provide a lower bound for the dynamic situation where the input data is not known in advance and the data structure is built up by iterated insertions. Using as performance measure the expected number of data bucket accesses needed to perform a window query, the static clustering problem turns into a classical optimization problem. For the special case of bucket capacity $c_b = 2$ the optimization problem is solvable in polynomial time, whereas for $c_b \geq 3$ it is NP-hard. In experiments using simulated annealing heuristics for the optimization the best dynamic structures as well as the static packed R-tree perform about 20% worse than the optimum on average. However, we again want to emphazise that we understand our contribution as 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 with respect to their performance. Still missing is a lower bound result, e.g. an optimal spatial data clustering, which would allow for the absolute comparison of the performance of the well-known data structures with the optimum. In this paper, we address the static situation where the data is known in beforehand. An optimal data clustering for this setting will also provide a lower bound for the dynamic situation where the input data is not known in advance and the data structure is built up by iterated insertions. Besides this motivation, investigations of the static situation are highly desirable because this setting is rather typical for geographical applications.

Surprisingly little work, however, has been devoted to the static case so far. To our knowledge, only two approaches addressing the static situation exist in the literature.

The *packed R-tree* [KF93] arranges the objects according to a space filling curve, e.g. the Hilbert curve, in a preprocessing step 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 window queries.

In [BPS94] it is shown that the static clustering problem turns out to be a *classical optimization problem*, if the expected number of data bucket accesses needed to perform a window quer is used as performance measure [PSTW93, KF93]. 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. The bucket optimization problem can be solved for bucket capacity $c_b = 2$ by mapping the optimization problem onto a well-known graph matching problem. Simulations with "uniformly distributed" query windows show that a representative of the best dynamic structures, the LSD-tree [HSW89], is outperformed by 20% on average. Considering the general situation with bucket capacity $c_b \geq 3$ and *arbitrary* cost functions the optimization problem emerges as NP-hard.

In [BPS94] the complexity of the optimization problem for cost functions based on data bucket accesses and $c_b \geq 3$ was left as an open problem. Since the computational complexity prohibits an exact optimization anyway, a suitable heuristic is highly desirable. Such a heuristic would allow for the absolute comparison of the

*This work has been supported by the ESPRIT II Basic Research Actions Program of the European Community under contract No. 6881 (AMUSING).

performance of static and dynamic spatial data structures with the (approximated) optimum.

In this paper, we continue investigating the optimization problem formulated in [BPS94]. We prove the NP-hardness of the bucket optimization problem for objective functions fulfilling a certain monotony property which holds for the cost measure based on data bucket accesses. Next we use simulated annealing as optimization heuristic. Our simulations show that the results obtained by the heuristic lie within 5% of the optimum (which ist typical for carefully tailored simulated annealing). Experiments exhibit that for several data sets including the Sequoia 2000 benchmark [SFGM93] both the LSD-tree and the static packed R-tree perform between 10% and 30% worse than the optimum. Finally, we consider directory page accesses, too, and sketch first results on overall optimized R-trees.

The bounds established by the optimal algorithm, resp. the heuristic, allow for the first time the absolute comparison of arbitrary static and dynamic spatial data structures. This is in contrast to all previous investigations which compared data structures only (relatively) to each other. We hope that our results contribute some general arguments to the discussion about the merits of further spatial data structure tuning.

The paper is organized as follows: Section 2 provides the background information about the performance measure and the optimization problem. Section 3 is devoted to complexity considerations. Section 4 gives a brief overview of the simulated annealing heuristic. Section 5 presents experimental results. Section 6 reports on first results of the overall optimization where directory accesses are also taken into account. Section 7 summarizes the results.

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. An appropriate and fair measure is desirable, where fair means that it is independent of data structure and implementation details. We choose the expected number of bucket accesses needed to perform a query as such a fair cost measure, which is appropriate too, because in practical applications data bucket accesses usually dominate query costs (e.g. in particular exceed by far external accesses to the paged parts of the corresponding directory).

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)

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

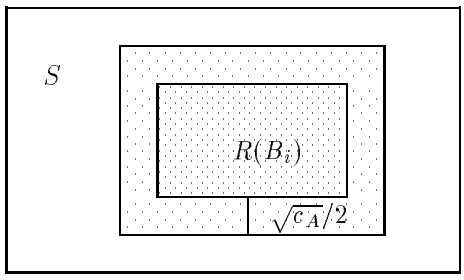

Figure 1: $P(q \text{ meets} B_i)$

$$+ \sqrt{c_A} \cdot \sum_{i=1}^{m} (R(B_i).L + R(B_i).H) + c_A \cdot m.$$

□

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 [PSTW93]. Note that the theoretical results presented below hold for all these models without any restriction.

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_{opt} for which the performance measure $\mathcal{PM}$ is minimal.

3 Computational Complexity

For a complete discussion of the complexity of the optimization problem, we start our investigations with a brief review of the *Simple Bucket Set Problem (SBSP)* where bucket capacity 2 is assumed. The *SBSP* 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.

Theorem 1. The *SBSP* can be solved in polynomial time.

Proof. See [BPS94] for details of the mapping of the *SBSP* onto the *MWMP* for nonbipartite graphs. The *MWMP* itself can be solved in $O(n^3)$ time (see e.g. [Law76]).

□

We now turn to the *Universal Bucket Set Problem (UBSP)* with bucket capacity $c_b \geq 3$ while assuming for the access probability $P(q \text{ meets } B_i)$ the following three properties which hold for any query model:

C1 $P(q \text{ meets } B_i) \geq 0$ holds for each bucket B_i.

C2 Let B_i be an arbitrary bucket which can accommodate at least one further object, then $P(q \text{ meets } B_i) \leq P(q \text{ meets } B_i \cup \{g\})$ holds for each object g (*monotony*).

C3 P can be computed in polynomial time.

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

Proof. We restrict the *UBSP* to $c_b = 3$ and show that the *3-PARTITION Problem (3-PP)* can be reduced to the *decision-oriented UBSP* which asks for the existence of a data space organization for a given cost.

The *3-PP* is defined as follows. Given a set A of $3m$ elements, a bound $S \in \mathbb{N}^+$, and for each $a \in A$ a weight $w(a) \in \mathbb{N}^+$ such that $S/4 < w(a) < S/2$ and $\sum_{a \in A} w(a) = mS$. The question is, whether A can be partitioned into m sets $A_1, A_2, \ldots, A_m$ such that $\sum_{a_i \in A_i} w(a_i) = S$ for $1 \leq i \leq m$. Note that by definition each A_i contains exactly three elements.

The rough idea of the transformation is simple. Set A of the *3-PP* is interpreted as a set G of geometric objects in the *decision-oriented UBSP* and each set A_i is associated with a bucket B_i. More precisely, we introduce two bijective functions g and b such that the geometric object $g(a) \in G$ corresponds to the element $a \in A$, i.e. $G = \{g(a_1), \ldots, g(a_{3m})\}$, and the bucket $b(A_i)$ to the set A_i.

Certainly, the key of the transformation lies in the construction of a suitable cost function C which must first fulfill the properties C1 to C3 and second meet the characteristics of the *3-PP*. Let $card(A_i)$ denote the cardinality of A_i. Then we claim that

$$C(b(A_i)) = |S - \sum_{a_i \in A_i} w(a_i)| + (card(A_i) + 1) \cdot S$$

is such a suitable cost function.

First we prove the properties C1 to C3. Properties C1 and C3 hold obviously. In order to show property

C2, let $b(A_i)$ contain none, one or two object(s) and let $a \in A - A_i$. From $S/4 < w(a) < S/2$ we get $0 \leq |S - \sum_{a_i \in A_i} w(a_i)| \leq S$ for the corresponding set A_i. Alltogether we yield

$$\begin{aligned} C(b(A_i)) &\leq S + (card(A_i) + 1) \cdot S \\ &= (card(A_i) + 2) \cdot S \\ &\leq C(b(A_i) \cup \{g(a)\}) \end{aligned}$$

and hence the monotony of C.

We now show that C meets the characteristics of the *3-PP*. It suffices to show that it is always less costly to put three objects together in one bucket than keeping them separately or storing just subsets together. To prove this, let a_1, a_2 and a_3 be arbitrary elements of A. Then we get:

$$\begin{aligned} C(b(\{a_1, a_2, a_3\}) &= \underbrace{|S - \sum_{i=1}^{3} w(a_i)|}_{\leq \frac{S}{2}} + 4S \\ &\leq \underbrace{|S - \sum_{i=1}^{2} w(a_i)|}_{\geq 0} + \underbrace{|S - w(a_3)|}_{\geq \frac{S}{2}} + 4S \\ &< C(b(\{a_1, a_2\}) + C(b(\{a_3\}). \end{aligned}$$

Similarly we yield $C(b(\{a_1, a_2\}) < C(b(\{a_1\}) + C(b(\{a_2\})$ and that it is always preferable to put six arbitrary objects into two bucket than to spread them over three or more buckets. Hence, the optimal "bucket set" in a *decision-oriented UBSP*-instance consists of buckets of three objects each.

Summing up the second summands of the single bucket costs gives us $4S$ as a lower bound for the optimal cost. This bound is reached iff the sum of the first summands equals 0 which occurs iff a solution for the corresponding *3-PP*-instance exists. Hence, deciding the existence of a data space organization with cost $4S$ in a *decision-oriented UBSP*-instance decides the *3-PP*-instance.

Since the *3-PP* is NP-complete [GJ75] and the reduction can be performed in polynomial time, the *decision-oriented UBSP* is NP-complete. Hence, the *UBSP* is NP-hard.

□

We conclude this section with some remarks concerning the significance and applicability of the complexity results. Obviously, the NP-hardness of the *UBSP* is determined by the properties of the cost function. A closer characterization of the performance measure according to some other query model, i.e. restricting the class of (monotone) cost functions, sheds some new light on the complexity of the modified *UBSP*.

We remind, for example, of a well-known result for (normalized) monotone submodular cost functions which besides the monotony fulfill two additional properties:

- $C(\emptyset) = 0$ (*normalization*).
- $C(B_i \cup B_j) + C(B_i \cap B_j) \leq C(B_i) + C(B_j)$ for any legal buckets B_i, B_j (*submodularity*).

Tightened up this way, the *UBSP* can be reformulated, s.t. it can by solved by the ellipsoid method. The running time of this optimization algorithm is polynomial, but of an unattractively high degree [GLS81].

However, common spatial cost functions are not submodular and even if we assume the existence of polynomial algorithms for subproblems, resp. concrete instances, the tremendous number of objects occurring in real applications dictates the use of appropriate heuristics anyway.

4 Simulated Annealing Heuristics

Simulated annealing [vLA87] can be viewed as a widely applicable approximation technique for solving combinatorial optimization problems. Excellent approximation results have been reported from many different application domains. Applications in the database area include query optimization problems [IW87, Swa89, IK90] and data clustering [MBM90, HLL94]. We decided to use simulated annealing to solve the *UBSP* approximately for the following reasons:

- Approximation quality and runtime can be traded off seamlessly.
- Lutton and Bonomi [LB86] successfully applied simulated annealing to the *MWMP* which is closely related to the *BSP*. Recently, Hua et.al. [HLL94] presented an efficient decomposition-based approach for a class of related problems, i.e. clustering problems which are too large to fit in main memory.
- Simulated annealing is a general algorithm, suitable for all query models and easy to implement as well.

We continue this section with a brief introduction of simulated annealing and then tailor the generic simulated annealing algorithm to the special *BSP* requirements.

4.1 Introducing Simulated Annealing

Simulated annealing can be regarded as the probablistic counterpart of the (deterministic) neighbourhood search. We first sketch the deterministic approach in order to achieve a better understanding of the randomized algorithm.

Besides an objective function, the application of a *neighbourhood search* in a configuration space assumes a *generation mechanism*, i.e. a simple prescription

PROCEDURE Simulated Annealing;
Determine start configuration i;
Initialize temperature $T > 0$;
REPEAT
 REPEAT
  Perturb config. i into config. j;
  IF $\Delta C_{ij} \leq 0$
   OR ELSE $\exp(-\frac{\Delta C_{ij}}{T}) < random[0,1)$ **THEN**
   Update config. i by j
  END
 UNTIL for a long time the cost remains stable **OR**
   a predefined number of iterations per cost level
   is exceeded
   ('equilibrium is approached sufficiently closely');
 Decrease temperature;
UNTIL stop criterium is satisfied (system is 'frozen').

Figure 2: The general simulated annealing algorithm in pseudocode.

how to perform a transition from one configuration to another by a small perturbation. Neighbourhood searching determines a start configuration and then runs an iterative process that generates a sequence of transitions as follows: A candidate is selected from the neighbourhood of the current configuration (all configurations that can be reached in one transition) and evaluated w.r.t. the cost function C. If the candidate configuration has a lower cost, the current configuration is replaced, otherwise another neighbour is selected. The process terminates when no further improvement is possible. However, this algorithm does not prevent from ending up in a local optimum far away from the global one, s.t. neighbourhood search is usually executed for a large number of start configurations in order to increase the chance for a successful completion.

Simulated annealing eliminates most disadvantages of the neighbourhood search: solutions depend no longer on the starting point and closely approach the optimum. To this end, an acceptance probability p is introduced which determines whether a new candidate is accepted or not. If the candidate configuration provides a better cost, then $p = 1$, and $p > 0$, otherwise. In the latter case, the value of p depends on the cost difference and an additional control parameter T, called "temperature". In general, the higher the temperature T, the greater the probability of accepting a new candidate. During the execution of the algorithm, the temperature is gradually lowered, "annealed", until virtually no changes are accepted anymore. Let i and j be two neighbouring configurations and let $\Delta C_{ij} = C(j) - C(i)$. Then Fig. 2 illustrates the general simulated annealing algorithm in pseudocode.

The main advantage of the simulated annealing algorithm compared to the neighbourhood search is, pictorially spoken, that a small hill can be climbed down in order to reach a big mountain nearby. Thus a temporary deterioration of the current configuration is tolerated in order to avoid a local optimum. By the way, simulated annealing is a well-founded method based on the theory of Markov chains.

4.2 Tailoring Simulated Annealing to the BSP

In this subsection, we instantiate the generic simulated annealing algorithm to cope with the *BSP*. The configuration space is set up by the set of all possible bucket sets except empty buckets or buckets violating the capacity bound. Any bucket set may serve as initial configuration. Nevertheless, since an initial configuration with good performance decreases the overall runtime, we start with bucket sets generated by the packed R-tree, the LSD-tree or the R-tree, respectively.

The neigbourhood of a configuration is implicitly defined by the transition function. Roughly spoken, a configuration is perturbed by choosing a source bucket B_i, a target bucket B_j which can be empty, and an object $g \in B_i$ which is moved from B_i to B_j. If B_i becomes empty, it is removed from the bucket set.

The temperature T follows the cooling schedule $T_{m+1} = \alpha \cdot T_m$ for $\alpha, T_0 \in [0.8, 0.95)$.

Unfortunately, the cardinality N_{n,c_b} of the configuration space grows hyperexponentially, fulfilling the recurrence equation

$$N_{n+1,c_b} = \sum_{k=0}^{\min\{n,c_b-1\}} \binom{n}{k} \cdot N_{n-k,c_b},$$

which approaches the bell numbers very closely even for small c_b and n. In order to speed up the algorithm we have carefully restricted the configuration space by rejecting configurations whose non-optimal performance can be predicted and we have also refined the transition function. However, going into more details is beyond the scope of this paper.

5 Experimental Results

We have implemented an $O(n^4)$ variant of the nonbipartite *MWMP* algorithm as well as several simulated annealing heuristics and compared various optimal bucket sets B_{opt} with bucket sets B_{pR} and B_{LSD} generated by an packed R-tree [KF93] and an LSD-tree [HSW89], respectively.[1]

The LSD-tree can be regarded as a representative of the best dynamic spatial data structures. In [PST93] it is shown that the LSD-tree is fully competitive to other structures, for instance, the R-tree [Gut84] and the R*-tree [BKSS90].

[1] All data structures and algorithms have been implemented in Eiffel [Mey91] on a SUN SPARCstation 10.

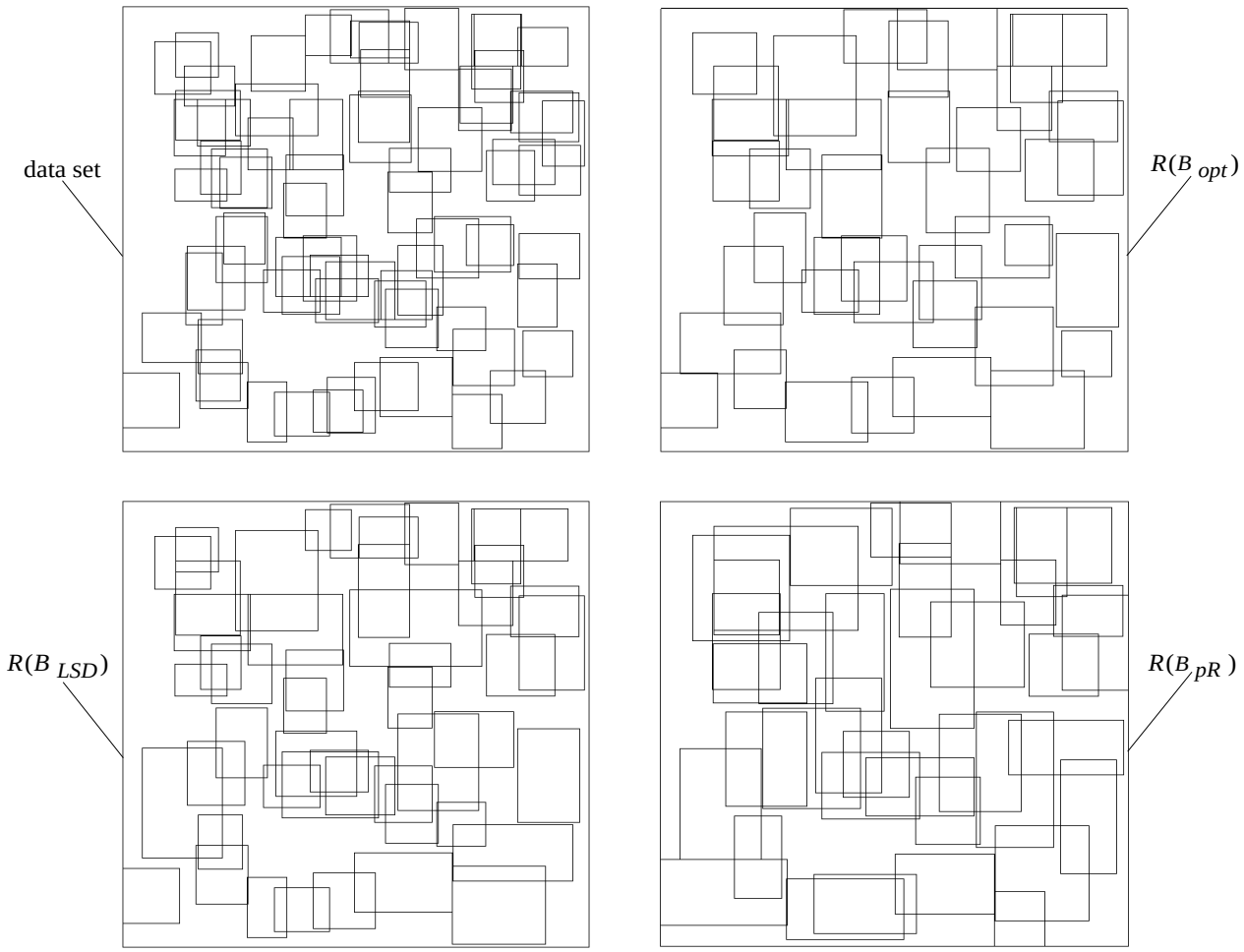

Figure 3: Visual comparison of B_{opt}, B_{pR} and B_{LSD} for 75 uniformly distributed rectangles with overlapping 1.0 and $c_b = 2$.

In all experiments, we use "uniformly distributed" square windows with an area covering 1% of the data space area. We start with bucket capacity $c_b = 2$.

In order to provide a visual impression of the data space organizations created by the optimal algorithm, the packed R-tree and 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. Fig.3 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 quantitative evaluation: the result of the optimal algorithm beats that of the packed R-tree by nearly 15% and that of the LSD-tree by nearly 13%.

In the following experiments we use six different data sets, namely 200 points, resp. rectangles, which are distributed uniformly, follow a 2-heap distribution or are extracted from the Sequoia 2000 regional benchmark [SFGM93]. The synthetic rectangles are nearly quadratic and cover the data space area 7.5 times. The results are presented in Table 1. They exhibit a clear tendency, namely the more complex the data input w.r.t. object shape and object distribution the higher the improvement gained by the optimal algorithm. Comparing the results for points and rectangles, the LSD-tree and the packed R-tree behave completely different. While the LSD-tree shows a certain deterioration, the packed R-tree seems to cope much better with rectangles than with points. Further investigations exhibit that the packed R-tree behaves the better, the regular the distribution, the more the objects overlap and the larger the area of the query windows.

Next we address the obvious question what happens if other window queries are performed than assumed during optimization, e.g. if the average area of the real windows differs from the expected one. Fig. 4, for instance, shows for the real data set that even in this case B_{opt} (using the 1% window as optimization parameter) substantially beats B_{pR} and B_{LSD} for a

Window area 1%	Point data			Rectangular data		
	uniform	2-heap	real	uniform	2-heap	real
$\mathcal{PM}(\mathcal{QM}_1, B_{opt})$	1.4383	1.3561	1.3232	9.1035	6.0875	2.0534
$\mathcal{PM}(\mathcal{QM}_1, B_{pR})$	1.8668	1.7802	1.8564	9.8266	6.8667	2.3276
deterioration in %	29.79	31.27	40.29	7.94	12.80	13.35
$\mathcal{PM}(\mathcal{QM}_1, B_{LSD})$	1.6365	1.5893	1.5273	10.5546	7.3371	2.4728
deterioration in %	13.78	17.19	15.42	15.94	20.53	20.42
$\mathcal{PM}(\mathcal{QM}_1, B_{SA})$	1.4577	1.3815	1.3868	9.2628	6.2744	2.1745
deterioration in %	1.35	1.87	4.80	1.75	3.07	5.90

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

wide range of real window areas. Note the extreme shape of the packed R-tree curve for points.

We also run some experiments comparing the approximation with the optimal solution in order to evaluate the simulated annealing algorithm. It turns out that the results of the heuristic lie within 5% of the optimum (see the last two rows in Table 1).[2]

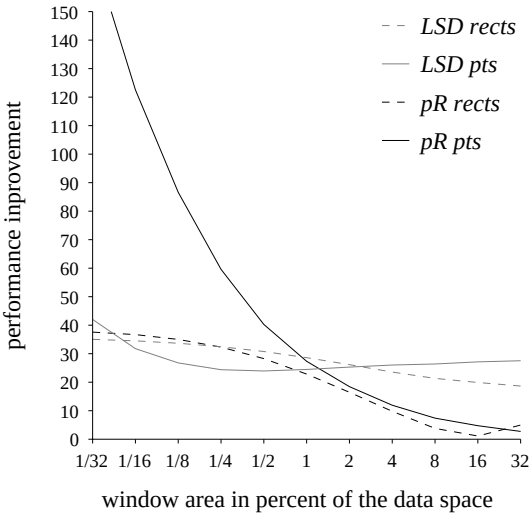

Figure 4: Relative improvement of B_{opt} over B_{pR} and B_{LSD}, respectively, for real data and variant window areas. The 1% window is used as optimization parameter.

We now turn to the more realistic situation with bucket capacity $c_b \geq 3$ and run the simulated annealing heuristics to produce "lower bounds". In our experiments we basically reuse the six data sets of Table 1, but we now generate 2000 objects with overlapping factor 1.0 and consider a larger part of the regional benchmark (also about 2000 objects). Table 2 shows the performance results for bucket capacity $c_b = 16$.

[2] All simulated annealing results are based on Markov chains of length n, i.e. at each temperature (stationary state) up to n object movements are performed.

6 Taking External Directory Accesses Into Account

Although the time penalty incured by external directory page 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 page accesses as well. Usually, with each directory page a directory region is associated which is the bounding box of all data bucket, resp. directory, regions pointed to from the directory page. Since directory regions again form a data space organization, such an overall approach seems to be feasible and has indeed been sketched in [PSTW93, KF93].

Clearly, the NP-hardness of the *UBSP* is carried over to the overall problem. So we have started to extend the simulated annealing heuristic to optimize spatial data structures as a whole. We have decided to concentrate on R-trees, because R-trees provide the highest freedom for clustering data buckets from B_{opt} into directory pages.

Basically, there exist two possible strategies for the overall approach. 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

Window area 1%	Point data			Rectangular data		
	uniform	2-heap	real	uniform	2-heap	real
$\mathcal{PM}(\mathcal{QM}_1, B_{SA})$	3.8196	3.4961	3.2724	4.8976	5.2187	5.0795
$\mathcal{PM}(\mathcal{QM}_1, B_{pR})$	4.3012	3.9264	3.6182	5.4335	6.5426	5.8838
deterioration in %	12.61	12.31	10.57	10.94	25.37	15.83
$\mathcal{PM}(\mathcal{QM}_1, B_{LSD})$	4.4168	3.8714	3.7339	5.8110	6.5307	6.4840
deterioration in %	15.64	10.74	14.10	18.65	25.14	27.65

Table 2: Cost comparisons of B_{pR}, B_{LSD} and B_{SA} for 2000 points, resp. rectangles, and $c_b = 16$.

best strategy and its efficiency is still an open problem.

Nevertheless, using the expected value of the sum of the directory page accesses and the data bucket accesses as performance measure, first experimental results exhibit an improvement of more than 15% over the LSD-tree and up to 20% over the packed R-tree for common window areas. For the LSD-tree there seems to be a tendency that the (quantitative) observations for the pure data bucket optimization also hold for the overall case. For the packed R-tree, however, the suboptimal organization of its directory becomes obvious. This outcome was not really unexpected because our data bucket experiments have already indicated that the packed R-tree (relatively) performs the better the higher the overlapping of the objects. Since the overlapping of directory regions is usually below the overlapping of the data bucket regions (the overlapping of directory regions usually decreases level by level from the lowest level to the root level), the directory represents the less efficient part of the packed R-tree.

7 Summary

In this paper, we prove the NP-hardness of the bucket optimization problem for cost functions fulfilling a certain monotony property which holds for the cost measure based on data bucket accesses. In order to cope with the NP-hardness of the general case we use simulated annealing as optimization heuristic. Our simulations show that the results produced by the heuristic lie within 5% of the optimum (which ist typical for carefully tailored simulated annealing). Experiments exhibit that for several data sets including the Sequoia 2000 benchmark [SFGM93] both a representative of the best dynamic structures, the LSD-tree, and the static packed R-tree perform between 10% and 30% worse than the optimum. In general, the more complex the object distribution the higher the performance loss. Finally, we consider directory page accesses, too, and sketch first results on overall optimized R-trees.

The bounds established by the optimal algorithm, resp. the heuristic, allow for the first time the absolute comparison of arbitrary static and dynamic spatial data structures. This is in contrast to all previous investigations which compared data structures only (relatively) to each other. We hope that our results contribute some general arguments to the discussion about the merits of further spatial data structure tuning.

Acknowledgments. The authors wish to thank Elisa Kwon and James Frew for making the Sequoia 2000 benchmark available to them.

References

[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*, Atlantic City, 1990.

[BPS94] L. Bachmann, B.-U. Pagel, and H.-W. Six. Optimizing spatial data structures for static data. In *Proc. Int. Workshop on Advanced Research in Geographic Information Systems*, pages 247–258. Springer LNCS Vol. 884, March 1994.

[GJ75] M.R. Garey and D.S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM J. Comput.*, 4:387–411, 1975.

[GLS81] M. Grötschel, L. Lovasz, and A. Schrijver. The ellipsoid method and its consequences

in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[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.

[HLL94] K.A. Hua, S.D. Lang, and W.K. Lee. A decomposition-based simulated annealing technique for data clustering. In *Proc. ACM Symposium on Principles on Database Systems*, pages 117–128, Minneapolis, Minnesota, 1994.

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

[IK90] Y.E. Ioannidis and Y.C. Kang. Randomized algorithms for optimizing large join queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 312–321, Atlantic CIty, NJ, 1990.

[IW87] Y.E. Ioannidis and E. Wong. Query optimization by simulated annealing. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 9–22, San Francisco, CA, 1987.

[KF93] I. Kamel and C. Faloutsos. On packing R-trees. In *Proc. 2nd Int. Conf. on Information and Knowledge Management*, pages 490–499, Washington D.C., 1993.

[Law76] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rhinehart and Winston, 1976.

[LB86] J.-L. Lutton and E. Bonomi. Simulated annealing algorithm for the minimum weighted perfect euclidean matching problem. *R.A.I.R.O. Recherche opérationnelle*, 20:177–197, 1986.

[MBM90] F.J. McErlean, D.A. Bell, and S.I. McClean. The use of simulated annealing for clustering data in databases. *Information Systems*, 15(2):233–245, 1990.

[Mey91] B. Meyer. *Eiffel: The Language*. Prentice Hall, 1991.

[PST93] B.-U. Pagel, H.-W. Six, and H. Toben. The transformation technique for spatial objects revisited. In D. Abel and B.-C. Ooi, editors, *Proc. 3rd Int. Symposium on Large Spatial Databases (SSD)*, pages 73–88, Singapore, June 1993. LNCS No. 692, Springer.

[PSTW93] B.-U. Pagel, H.-W. Six, H. Toben, and P. Widmayer. Towards an analysis of range query performance in spatial data structures. In *Proc. ACM PODS Symposium on Principles of Database Systems)*, pages 214–221, Washington, D.C., May 1993.

[SFGM93] M. Stonebraker, J. Frew, K. Gardels, and J. Meredith. The SEQUOIA 2000 storage benchmark. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 2–9, Washington, D.C., May 1993.

[Swa89] A. Swami. Optimization of large join queries: Combining heuristics and combinatorial techniques. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 367–376, Portland, Oregon, 1989.

[vLA87] P.J.M. van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, Dordrecht, 1987.