Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK) 2001

Determining the Convex Hull in Large Multidimensional Databases

Christian Böhm, Hans-Peter Kriegel

University of Munich, Oettingenstr. 67, D-80538 Munich, Germany
{boehm,kriegel}@informatik.uni-muenchen.de

Abstract. Determining the convex hull of a point set is a basic operation for many applications of pattern recognition, image processing, statistics, and data mining. Although the corresponding point sets are often large, the convex hull operation has not been considered much in a database context, and state-of-the-art algorithms do not scale well to non main-memory resident data sets. In this paper, we propose two convex hull algorithms which are based on multidimensional index structures such as R-trees. One of them traverses the index depth-first. The other algorithm assigns a priority to each active node (nodes which are not yet accessed but known to the system), which corresponds to the maximum distance of the node region to the tentative convex hull. We show both theoretically as well as experimentally that our algorithms outperform competitive techniques that do not exploit indexes.

1. Introduction

Multidimensional data sets are prevalent in many modern database applications such as multimedia [12], CAD [20], medical imaging [24], molecular biology [23], and the analysis of time sequence data [1]. In these applications complex objects are usually translated into vectors of a multidimensional space by a *feature transformation* [31]. The distance between two different *feature vectors* is a measure for the similarity of the corresponding objects. Therefore, feature vectors are often used in similarity search systems [3] where the central task is the search for objects which are most similar to a given query object. Similarity queries are transformed into neighborhood queries in the feature space, which can be efficiently supported by multidimensional index structures. Therefore, multidimensional indexes are often maintained to support modern database applications.

If the user wants to get a deeper insight into the intrinsic structure of the data stored in the database, the similarity search is not sufficient. The user will rather want to run statistical analyses or apply data mining methods such as classification [25], clustering [29], outlier detection [22], trend detection [11], or the generation of association rules [2]. Several of the corresponding algorithms use similarity queries as *database primitives*, others mainly rely on other basic operations.

The determination of the convex hull is an example of a primitive operation which is useful for many analysis methods and has successfully been applied in application domains such as pattern recognition [4], image processing [28] or stock cutting and allocation [13]. The convex hull of a point set in the plane is defined as the smallest convex polygon containing all points. Intuitively, the convex hull is obtained by spanning a rubber band around the point set. Both, the points touched by the rubber band as well as the shape of the resulting polygon are called the *convex hull* of the data set (cf. fig. 1). In 3 and higher dimensional data spaces, the convex hull is analogously defined as the minimum convex polyhedron (polytope) of the point set.

We briefly sketch a few applications of the convex hull: The convex hull is an exact and comprehensive description of the shape of a cluster of data points and can therefore be used as a postprocessing step to cluster analysis algorithms [29]. Several clustering algorithms even use the convex hull in the definition of the notion of a cluster: E.g. [7] defines a cluster to be a set of points with minimum diameter of the corresponding convex hull. The determination of the convex hull becomes thus a primitive operation for the clustering algorithm. Another application of the convex

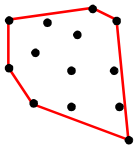


Fig. 1. Convex hull.

hull is the robust estimation [15]. A robust estimator (or Gastwirth-estimator) is based on the observation that points in the "inner" of a point set are generally more trustable than the extreme points. Therefore, the points of the convex hull are removed from the data set as a preprocessing step [19]. A further application of convex hull is the *isotonic regression*. Regression methods approximate point sets by functions from a given class, e.g. linear functions, such that the approximation error (least square) is minimized. In the isotonic regression, the function class are monotonic staircase functions. The isotonic regression of a point set can be found using the convex hull of the points after a transformation. A recent application of the convex hull is the Onion Technique [9] for linear optimization queries. This method is based on a theorem that a point which maximizes an arbitrary multidimensional weightening function can be found on the convex hull of the data set. The authors improve the search for such maximum points by separately indexing the points of the convex hull of the data set. Börzsönyi, Kossmann and Stocker propose the Skyline technique [5] which is a concept similar to the convex hull. The skyline of a dataset can be used to determine various poitn of a data sets which could optimize an unknown objective in the user's intentions. E.g. users of a booking system may search for hotels which are cheap and close to the beach. The skyline of such a query contains all possible results regardless how the user weights his criteria *beach* and *cost*. The skyline can be determined in a very similar way as the convex hull. The algorithms proposed in this paper can also be adapted for the skyline.

Due to the high practical relevance of the convex hull, a large number of algorithms for determining the convex hull in 2 and higher dimensional space has been proposed [4, 7, 8, 10, 16, 17, 21, 26, 27, 30]. Most of these algorithms, however, require the data set to be resident in main memory. In contrast, we will propose two algorithms to determine the convex hull of a point set stored in a multidimensional index. In this paper, we focus on the important case of point sets in the plane.

The remainder of this paper is organized as follows: In section 2, we introduce two algorithms determining the convex hull of a point set in a multidimensional index and state some important properties of our solutions. Section 3 evaluates our technique experimentally and section 4 concludes the paper.

2. Convex Hulls in Large Databases

In this section, we will introduce our two algorithms for the determination of the convex hull of a large point set which is stored in a multidimensional index. For our methods, we need hierarchical index structures such as the R-tree [18] or its variants. Our algorithms, however, do not exploit particular properties of R-trees such as the fact that the page regions are minimum bounding rectangles. Therefore, our algorithms are easily adaptable to index structures which use non-bounding rectangles or other shapes such as spheres or polygons. Basic operations which must be efficiently supported are the intersection with polylines, and the minimum distance to polylines and points. We will first concentrate on the distance-priority algorithm and later introduce the depth-first algorithm. In both cases, we will prove important properties of the algorithms. We will show that the distance priority algorithm yields a minimum number of page accesses. For the depth-first algorithm, we will provide worst-case bounds for the worst case time complexity

For all algorithms, we apply the following simplification: Before starting the actual convex hull algorithm, we search the points which are extreme (maximum and minimum) in *x*- and *y*-direction (cf. figure 2). These 4 points, called *min-x-point*, *min-y-point*, *max-x-point*, and *max-y-point*, are guaranteed to be included in the convex hull. These points define 4 quadrants in which the convex hull is searched separately. In our description, we will restrict ourselves to the part of the convex hull between the min-x-point and the min-y-point. The advantage is that every line segment of the hull is oriented from upper left to lower right. Similar properties are valid for the other quadrants.

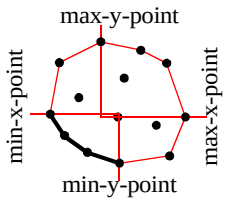


Fig. 2. Quadrant Restriction

2.1 The Distance Priority Algorithm

For developing algorithms for query processing upon multidimensional index structures it is recommendable to determine the conditions under which a page must be accessed. Figure 3 shows

the situations before (upper) and after (lower) the run of a convex hull algorithm. Initially only the min-x-point and the min-y-point is known. This knowledge, however, is enough to exclude p_5 definitely from processing, because p_5 cannot contain a point on the right of the current convex hull. On the lower side of figure 3 we recognize that all points of the convex hull are located on the pages p_1 and p_2. These pages are obviously necessary for the determination of the convex hull. But it is also necessary to know the content of page p_3 to decide whether the result is correct, because p_3 could contain a point in the shaded area near by the lower left corner which belongs to the convex hull. The pages p_4 and p_5, in contrast, are topologically completely contained in the convex hull polygon and are, therefore, not able to hold any points which could be part of the convex hull. This observation leads us to the first lemma:

Lemma 1. A correct convex hull algorithm must access at least the pages which are topologically not completely contained in the convex hull polygon.

Proof. If a page which is not completely contained in the convex hull polygon, is not accessed, this page could store a point which is also not contained in the determined convex polygon. Then, the determined polygon is not the convex hull. ❑

With our restriction to the quadrant between the min-x-point and the min-y-point it is easy to determine the pages which are not completely contained in the convex hull: The lower left corner of such regions is always left from the convex hull. If the convex hull is organized by e.g. an AVL-tree, the corresponding line segment can be found in O(log n) time.

Lemma 1 provides a stopping criterion for CH algorithms on a multidimensional index: We are done whenever all pages have been processed which are not contained in the convex hull. Our next lemma will give us the sort order in which the pages must be accessed. This lemma identifies the distance between the lower left corner of the page region and the *tentative convex hull* to be the key information. The tentative convex hull (*TCH*) is always the convex hull of all points which have already been processed. At the beginning, it is initialized with the line segment connecting the min-x-point and the min-y-point.

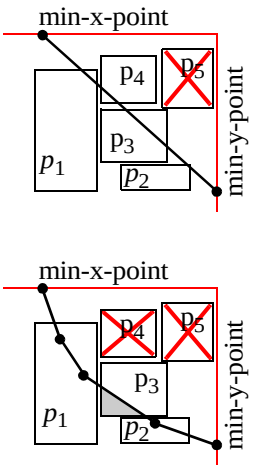
Lemma 2. If there is more than one page which is not completely contained in the TCH it is not possible that the page with the highest distance from the TCH is excluded by the convex hull.

Proof. To exclude a page p_1 from processing requires an unprocessed point v which has a higher distance from the TCH than the lower left corner of p_1. Let p_2 be the page on which the v is stored. Then, the distance between the TCH and the lower left corner of p_2 is at least as high as the distance between v and the TCH. Such a page cannot exist, because the lower left corner of p_1 has maximum distance from the TCH among all pages. ❑

Lemma 2 can also be visualized with figure 3. Page p_1 is the farthest page from the TCH. It is not possible that any point on p_2, p_3, p_4, or p_5 extends the TCH so much that p_1 will be completely contained. We use the result of lemma 2 for our first algorithm (cf. figure 3) which manages an active page list (APL) for all pages which are not excluded by the TCH. The pages are ordered by decreasing distance between the lower left corner and the TCH. Data pages which are not excluded are basically processed as in Preparata's online algorithm [26] by determining all points which are outside the TCH. Such points are inserted into the TCH in which case it could be necessary to discard neighboring points from the TCH to remove inconvexities. The correctness of our algorithm can be guaranteed by showing that at least all points of the convex hull are passed to the Preparata's algorithm, which has been shown to be correct elsewhere [27].

Lemma 3. The algorithm *distance_priority_ch* is correct.

Proof. As the TCH is always the convex hull of a growing subset of the actual set of points, no point of the TCH can be left from the actual convex hull. Every page is processed unless it has been pruned or one of its hierarchical predecessors has been pruned. If the page has been pruned, then it must have been right from the TCH at some stage of processing, and, therefore, it must be right from the actual convex hull. If one of the predecessors has been discarded, the predecessor must be right from the actual convex hull. If a predecessor is right from the convex hull, the page must also be right from it, because the region of the page is completely contained in the region of the predecessor. Thus, each page which is not com-




```
ALGORITHM distance_priority_ch (root: page)
    apl := {root} ;
    minxpoint := search database for minimum x ;
    minypoint := search database for minimum y ;
    tch := {minxpoint,minypoint};
    WHILE NOT is_empty (apl)
        b := element of apl with max. distance to tch ;
        DEQUEUE b FROM apl ;
        LOAD b FROM DISK ;
        IF is_data_page (b)
            FOR EACH v ELEMENT OF b
                IF is_left (v, tch)
                    tch := tch UNION v ;
                    WHILE NOT convex (tch)
                        REMOVE next(v) FROM tch ;
        ELSE
            s := SET OF CHILD PAGES OF b ;
            apl := apl UNION s ;
        FOR ALL PAGES p ELEMENT OF apl
            IF is_right (p,tch)
                apl := apl \ p ;
```

Fig. 3. The distance priority algorithm for the CH

pletely contained in the convex hull, is processed. Every point of the convex hull is contained in a page that is not completely contained in the convex hull. Therefore, every point of the convex hull plus some additional points from the data set are fed into Preparata's algorithm. Therefore, the correctness is guaranteed. ❑

Now we will discuss the performance of our algorithm from a theoretical point of view. Our first statement is that our algorithm yields an optimum number of page accesses when assuming that the index is given.

Lemma 4. The distance priority algorithm yields a minimum number of page accesses.

Proof. According to lemma 1, all pages must be processed which are not completely contained in the convex hull. In each step of the algorithm, the page $p_{\min}$ with the highest distance between the convex hull and the lower left corner of $p_{\min}$ is processed. According to lemma 2, the convex hull cannot be extended such that $p_{\min}$ is contained in it. The algorithm accesses exactly the pages which are not completely contained in the convex hull. ❑

Finally, we will show that our distance priority algorithm yields a worst-case complexity of $O(n \log n)$ for the CPU-time. This complexity is also the lower bound for algorithms which are not based on an index.

Lemma 5. The CPU time complexity of *distance_priority_ch* is in $O(n \log n)$.

Proof. In each step of the algorithm, one page of the index is processed. The number of index pages is linear in the number of stored points. We assume the APL to be organized in a priority queue, which requires $O(\log n)$ time for each inserted or removed element. The tentative convex hull is organized in an AVL-tree or 2-3-tree which allows fast access in $O(\log n)$ time to the y-coordinates of the stored points. Therefore, insertions and deletions can be performed in $O(\log n)$ time. The decision whether a new point is left or right from the tentative convex hull can be made in $O(\log n)$ time, because it requires the search of the points in the TCH with the next higher and the next lower y-candidates. If the point is inserted to the TCH, some neighboring points may be deleted from the TCH, which requires again $O(\log n)$ time for each deletion. During the run of the complete algorithm, at most n points can be inserted and deleted from the TCH. Taking together, both the management of the APL and the management of the TCH require $O(n \log n)$ time during the complete runtime of the algorithm. ❑

2.2 The Depth-First Algorithm

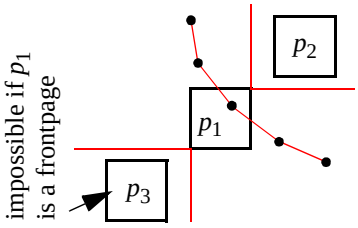


Fig. 4. Front pages and excl. pages.

Although the distance priority algorithm can be proven to be optimal with respect of the number of accessed pages, it does not affect the worst-case boundaries for the CPU runtime which are O(n log n) for our algorithm as well as for known algorithms which are not database-algorithms. There is a theoretical result that O(n log n) is also the lower limit for the worst-case complexity of the convex hull operation.This limit, however, is only valid for points which are not stored in an index. We may wonder whether the existence of a multidimensional index enables us to improve the worst-case complexity? This would be no contradiction to the theoretical result, because constructing an index requires O(n log n) time. Additionally, it is possible to extract the points in the order of ascending x- or y-coordinates from an index in linear time. The question is, whether all of the worst-case complexity is subsumed in the operation of index construction, or if there is a substantial remaining worst-case complexity. It seems intuitively clear that we cannot do better than O(n), because n is the size of the result set in the worst case. In this section, we will develop our depth-first algorithm, and we will show that this algorithm runs in O(n) time in the worst case. The general idea is to process the pages in a sequence order that corresponds to the sequence order of the points in the convex hull, starting at the min-x-point and ending at the min-y-point.

Therefore, the points are inserted into and deleted from the tentative convex hull only at the end, and no points up to the end point is ever accessed. This saves the O(log n) factor in the management of the tentative convex hull (TCH). The other O(log n) factor for the management of the active page list is saved by the depth-first traversal itself. In this context, the most thrilling question is whether or not a sequence order of pages corresponding to the sequence order of the convex hull exists at all and whether or not this sequence order is compatible with a depth-first traversal through the tree. Both questions are not self-evident, because the page regions are objects yielding a spatial position *and* extension which makes it difficult to apply any ordering to them. E.g. points could be uniquely ordered by x- or y-coordinate or by their angle and distance with respect to some appropriate reference point. Such kinds of ordering have been successfully applied for traditional convex hull algorithms. For extended spatial objects, however, the x- and y-projections of different objects overlap, as well as the corresponding angle does. We will develop in the first two lemmata a new appropriate ordering for page regions and show that it is compatible with the sequence order of the convex hull. For these two lemmas, we assume overlap-free indexes. Note that some index structures allow overlap among sibling page regions to improve the adaptation to the data distribution in the presence of heavy update to the database. For unlimited overlap (i.e. arbitrarily bad indexes), obviously our method cannot guarantee a general O(n) time complexity in the worst case. Our lemma gives us a statement to exclude some pages from processing even if the TCH in this area is not yet known. We still restrict our convex hull to the quadrant between the min-x-point and the min-y-point. For other quadrants, analogous properties hold.

Lemma 6. If a page p_2 is completely in the sector above and right from another page p_1, this page cannot contain any point of the convex hull.

Proof. The convex hull cannot be left or below from any point which is stored in the page p_1. Therefore, p_1 is either completely contained in the convex hull or it is intersected by it. In our quadrant, all line segments of the convex hull yield a negative skew. Therefore, the convex hull cannot intersect p_2. ❑

For an illustration of lemma 6, cf. the pages p_1 and p_2 in figure 4. In any set of non-overlapping pages, some pages exclude other pages from processing according to lemma 6. We call those pages which are not excluded the *frontpages* of the set. We know for each frontpage p_1 that no other frontpage is completely in the sector above and right (such as p_2) or completely in the sector below and left (such as p_3) of the page. In the first case, p_2 is excluded by p_1. In the second case, p_3 would exclude p_1 such that p_1 cannot be a frontpage.

Definition 1 (Frontpage ordering):
A frontpage p_2 is greater than a frontpage p_1 (or equal) with respect to frontpage ordering ($p_2 \geq_{fpo} p_1$) if at least one of the following conditions hold:

- p_2 is completely right from p_1
- p_2 is completely below from p_1
- p_1 and p_2 are identical

Lemma 7. The relation "$p_2 \geq_{fpo} p_1$" for non-overlapping frontpages p_1 and p_2 is a total ordering.

We briefly sketch the idea of the proof which is technically complex but not very interesting. The reflexivity is obvious. The antisymmetry is not given for pages which yield overlap or which don not fulfill the frontpage property: If, for instance, p_1 is both, above and right from p_1, we have $p_2 \geq_{fpo} p_1$ and $p_1 \geq_{fpo} p_2$ which would violate the antisymmetry. For overlap-free frontpages, in contrast, it follows that p_1 is either above, or right from p_2 but not both (in this case, p_2 would exclude p_1). Similarly, p_2 is either below or left from p_1 but not both. In all cases, we cannot have $p_2 \geq_{fpo} p_1$ and $p_1 \geq_{fpo} p_2$ for two different pages p_1 and p_2. For the transitivity, a similarly complex argumentation with many case distinctions is necessary, but viable.

Next, we will prove the most important lemma for our depth-first algorithm which claims that the frontpage ordering is compatible with the ordering of points in the convex hull. The consequence is that if we process the pages by ascending frontpage ordering, we get the points of the convex hull in the right order.

Lemma 8. For two points $v_1=(x_1,y_1)$ and $v_2=(x_2,y_2)$ of the convex hull which are stored on different pages p_1 and p_2, the following statements are equivalent:

(1) $p_2 \geq_{fpo} p_1$
(2) $x_2 \geq x_1$
(3) $y_2 \leq y_1$

Proof.

(2) $\Leftrightarrow$ (3): The equivalence of statement (2) and (3) is an immediate consequence of the properties of the convex hull and our restriction to the quadrant between the min-*x*-point and the min-*y*-point.

(1) $\Rightarrow$ (2): Case (a): p_2 is completely right from p_1. In this case, all points stored on p_2 have higher *x*-coordinates than any point stored in p_1, and therefore, $x_2 \geq x_1$.

Case (b): p_2 is completely below p_1. In this case, all points stored on p_2 have lower *y*-coordinates than any point stored in p_1, and therefore, $y_2 \leq y_1$. Due to the equivalence of statement (2) and (3), we know that $x_2 \geq x_1$.

(2) $\Rightarrow$ (1): We know that p_2 stores a point with a higher x-coordinate than a point that is stored in p_1. Therefore, p_2 cannot be completely left from p_1. Due to "(2)$\Leftrightarrow$(3)" we further know that p_2 stores a point with a lower *y*-coordinate than a point that is stored in p_1. So p_2 cannot be completely above p_1. In order to be overlap-free, p_2 must either be completely right from p_1 or completely below. In both cases it follows that $p_2 \geq_{fpo} p_1$. ❑

Last before we are ready to present our depth-first algorithm, we state that the frontpage ordering is compatible with the depth-first traversal of the tree:

Lemma 9. If $p_2 \geq_{fpo} p_1$ (but not $p_2 = p_1$) then also $c_{2,i} \geq_{fpo} c_{1,i}$ for all child pages $c_{1,i}$ of p_1 and $c_{2,i}$ of p_2 which are frontpages.

Proof. Follows from definition 1 and as child pages are contained in the region of their parent. ❑

From our set of lemmata it follows that we can traverse the tree in a depth-first fashion, in each node calling those child nodes which are front pages, ordered by the frontpage order. If we do so, we find the points of the convex hull in the right order. This property is exploited in our algorithm depth_first_ch (cf. figure 6). Here, the part of the algorithm which corresponds to the conventional hull determination (IF is_data_page...) corresponds to Graham's scan. The part of the algorithm

```
ALGORITHM depth_first_ch (b: page)
    LOAD b FROM DISK ;
    IF is_data_page (b)
        V := set of points stored on b ;
        ORDER V BY x-coordinate ASCENDING ;
        FOR EACH v ELEMENT OF V
            l := linesegment connecting the last
                 point of the tch with minypoint ;
            IF is_left (v, l)
                m := linesegment connecting v with
                     the point before last point of tch
                WHILE is_right (last point of tch, m)
                    DELETE last point FROM tch ;
                APPEND v TO tch ;
    ELSE (* directory page *)
        P := set of child pages of b ;
        (* restrict P to the frontpages: *)
        FOR EACH p ELEMENT OF P
            FOR EACH qELEMENT OF P
                IF excludes (p,q)
                    REMOVE q FROM P ;
        ORDER P BY "≥fpo" ASCENDING ;
        FOR EACH p ELEMENT OF P
            l := linesegment connecting the last
                 point of the tch with minypoint ;
            IF NOT is_right (p,l)
                depth_first_ch (p) ;
```

Fig. 6. The depth first algorithm for the CH

for the directory pages first determines those child pages which are frontpages. Then these pages are ordered according to the frontpage ordering relation and accessed unless they can be pruned.

This section is concluded by two lemmata stating the correctness and the worst-case runtime of the algorithm which is proven to be in O(*n*).

Lemma 10. The algorithm depth_first_ch is correct.

Proof. Analogously to lemma 3, we can show that every point of the convex hull is passed to the conventional convex hull algorithm. The variant used in our algorithm requires the points of the convex hull to be passed in the appropriate order, i.e. with ascending *x*-coordinates and descending *y*-coordinates. This is follows from lemmata 8 and 9. ❑

Lemma 11. The algorithm depth_first_ch runs in O(*n*) time in the worst case, if the data set is stored in an overlap-free multidimensional index.

Proof. In the worst case, each page of the index is accessed once. The number of pages is linear in *n*. For each page, only a constant number of operations is raised. The only exception is the deletion of the last point of the TCH (in constant time) which is repeated until the convexity of the TCH is maintained. The total number of deletions during the run of the algorithm, however, is also restricted by *n*. ❑

3. Experimental Evaluation

To demonstrate the practical relevance of our technique and the superiority over competitive approaches we implemented our distance priority algorithm, two variants of depth-first algorithms, and two related approaches, the scalable variants of Graham's scan as well as Preparata's online algorithm. For efficiency all variants have been coded in C and tested on HP C160 workstations under HP-UX 10.12. The data set was stored on a disk with 5 ms seek time, 5 ms rotational delay, and a transfer rate of 4 MByte per second.

All implementations of our new index-based algorithms operate on bottom-up constructed X-trees [6]. We implemented two variants of depth-first algorithms. The first variant processes the child nodes of the current node ordered by the front-page ordering as described in section 2.2 (in

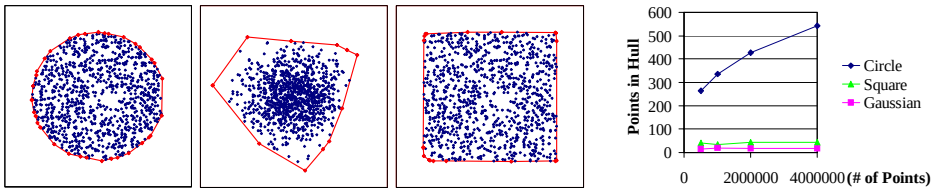


Fig. 5. Convex hulls and their characteristics: circle, gaussian and square

the following called *depth-first with FPO*). In the second variant the child nodes are ordered by their maximum distance to the TCH (*depth-first with priority*). In contrast to the distance priority algorithm, this algorithm traverses the index depth-first, i.e. every branch is completed before the next branch can be started. Therefore, it is not necessary to manage a priority-queue. To be scalable to non-memory databases, Graham's scan [16] was implemented using the mergesort algorithm

For our experiments we used 3 data sets with 4,000,000 data points with different distribution characteristics. The first dataset contains points which are uniformly distributed in a unit circle. The points of the second data set are normally distributed (*Gaussian data set*). The third set contains uniformly distributed points in the unit square. All data sets and their convex hulls (for 1,000 points) are depicted in figure 5. The characteristics of the convex hull are depicted on the right side of figure 5. The number of points in the convex hull of the circle data set is fast yet sublinearly increasing with increasing data set size. In contrast the number of points in the convex hull of the other data sets is not much influenced by the database size.

In all subsequent experiments, the number of data points varied between 500,000 and 4,000,000. The block size of the index and data files was consistently set to 2 KBytes. Graham's scan is the only algorithm which needs a page cache for processing (for the sorting step). We constantly allowed 800 database pages in cache which is between 5% and 40% of the database.

Figure 7 depicts the results of the experiments on the circle data set. In the left figure, the number of page accesses of the different algorithms is compared. On the circle data set, the distance priority algorithm and the two depth-first algorithms required exactly the same number of page accesses (up to 530). In contrast, Graham's scan needs 48,000 page accesses for sorting of the data set. The online algorithm by Preparata is with 16,000 block accesses also clearly outperformed. The diagram in the middle shows the CPU times. Here the depth-first variant with frontpage ordering yields the best performance with 1.1 seconds for 4,000,000 data points. As the circle data set has many points in the convex hull, the reduced effort for the management of the TCH (which is a consequence of frontpage ordering because points are only inserted or deleted at the end of the TCH) results in substantial performance gains. The depth-first variant needed 4.4 seconds of CPU time or four times as much as depth-first with FPO. Slightly worse (5.1 sec) due to the management of the priority queue was the distance priority algorithm. The variants without index, however, needed with 43 (Preparata) and 110 (Graham) seconds, respectively, a factor of 39 (100) times more CPU power than our best performing algorithm. The right diagram shows the total time subsuming CPU time and I/O time. As all algorithms are I/O bound the differences between the index-based algorithms are alleviated. For 4,000,000 data points, the FPO algorithm needed 6.2 seconds, the other depth-first algorithm needed 9.6 seconds and the distance priority algorithm needed 10.36 seconds. With 200 seconds, the online algorithm was outperformed with factor 32. The improvement factor over Graham's scan (580 seconds) was even 94.

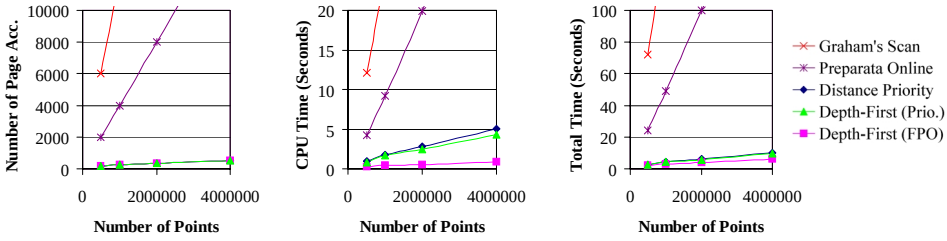


Fig. 7. Experimental results of the circle data set

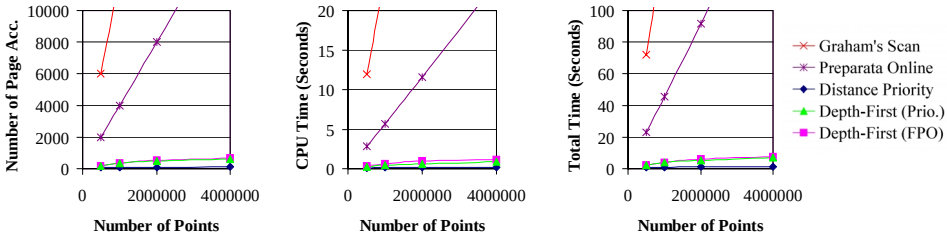


Fig. 8. Experimental results of the square data set

The characteristic of the square data set which is depicted in figure 8 is quite different from the characteristic of the circle data set. The number of points in the convex hull is with 43 very small. Therefore, the management of the convex hull does not play an important role and the advantage of frontpage ordering over our other index-based approaches disappears. For this set, the distance priority algorithm clearly yields the best number (120) of page accesses. With 650 page accesses, the depth-first algorithm is clearly worse than the distance priority algorithm, but still very good in comparison with the algorithms without index. Ordering child pages by priority rather than by FPO which is a kind of compromise between the other two approaches yields a negligible advantage (630 page accesses). As the TCH management does not play an important role and since processing of each page needs CPU power, the distance-priority algorithm yields also the best CPU performance. With respect to the overall time, the distance priority algorithm outperforms the depth-first variant with priority ordering by a factor 5.1, the depth-first algorithm with FPO by 5.5, the online algorithm by 130, and Graham's scan by the factor 420.

These results are also confirmed by the gaussian data set . The corresponding experiment (total time only) is shown in figure 9. With 1.3 sec. (distance priority) and 1.2 sec (both depth-first algorithms), respectively, all index based approaches required about the same time. In contrast, Preparata's online algorithm needed 180 seconds, i.e. more than 150 times slower than our approaches. With 590 seconds, Graham's scan is even outperformed by factor 510. Like in all our experiments, the improvement factors of our new techniques over competitive techniques was increasing with increasing database sizes. For instance, the improvement factor over the online algorithm (Gaussian data set) starts with 25 for 500,000 data points. For 1 million points, the factor increases to 52, then 89 (2 million points), and finally 150. The analogous sequence for the improvement over Graham's scan is (80, 170, 291, 510).

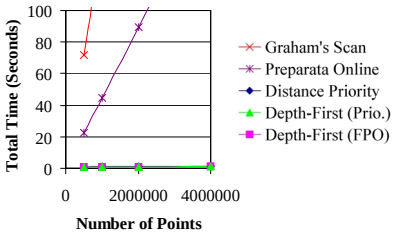


Fig. 9. Experiments on Gaussian data set

4. Conclusions

In this paper, we have proposed to use multidimensional index structures for the determination of the convex hull of a point database. Indexes can be either traversed depth-first or the access of the next node is controlled by the priority of the node which corresponds to the maximum distance between the node and the currently available part of the convex hull. The analytical evaluation of our technique shows that the distance priority algorithm is optimal with respect to the number of disk accesses. The depth-first algorithm, in contrast, has a better (i.e. linear) worst-case complexity with respect to CPU time. Our experimental evaluation demonstrates the superiority over approaches storing the point set in flat files. The database implementation of the most well-known convex hull algorithm, Graham's scan, is outperformed by factors up to 510.

Acknowledgments

We are grateful to Donald Kossmann for fruitful discussions.

References

1. Agrawal R., Faloutsos C., Swami A.: *Efficient similarity search in sequence databases*, Int. Conf. on Found. of Data Organization and Algorithms, 1993.
2. Agrawal R., Imielinski T., Swami A.: *Mining Association Rules between Sets of Items in Large Databases*, ACM SIGMOD Int. Conf. on Management of Data, 1993.
3. Ankerst M., Kriegel H.-P., Seidl T.: *A Multi-Step Approach for Shape Similarity Search in Image Databases*, IEEE Transactions on Knowledge and Data Engineering (TKDE), Vol. 10, No. 6, 1998.
4. Akl S. G., Toussaint G. T.: *Efficient Convex Hull Algorithms for Pattern Recognition Applications*, Int. Joint Conf. on Pattern Recognition, 1978.
5. Börzsönyi S., Kossmann D., Stocker K.: *The Skyline Operator*, Int. Conf on Data Engineering, 2000.
6. Berchtold S., Böhm C., Kriegel H.-P.: *Improving the Query Performance of High-Dimensional Index Structures Using Bulk-Load Operations*, Int. Conf. on Extending Database Technology, 1998.
7. Brown K. Q.: *Geometric Transformations for Fast Geometric Algorithms*, Ph.D. thesis, Dept. of Computer Science, Carnegie Mellon Univ., Dec. 1979a.
8. Bykat A.: *Convex Hull of a Finite Set of Points in Two Dimensions*, Info. Proc. Lett., No. 7, 1978.
9. Chang Y.-C, Bergman L. D., Castelli V., Li C.-S., Lo M.-L., Smith J. R.: *The Onion Technique: Indexing for Linear Optimization Queries*, ACM SIGMOD Int. Conf. on Management of Data, 2000.
10. Eddy W.: *A New Convex Hull Algorithm for Planar Sets*, ACM Trans. Math. Software 3 (4), 1977.
11. Ester M., Frommelt A., Kriegel H.-P., Sander J.: *Algorithms for Characterization and Trend Detection in Spatial Databases*, Int. Conf. on Knowledge Discovery and Data Mining (KDD), 1998.
12. Faloutsos C., Barber R., Flickner M., Hafner J., Niblack W., Petkovic D., Equitz W.: *Efficient and Effective Querying by Image Content*, Journal of Intelligent Information Systems, Vol. 3, 1994.
13. Freeman H., Shapira R.: *Determining the Minimum-Area Encasing Rectangle for an Arbitrary Closed Curve*, Comm. ACM, Vol 18, No. 7, 1975.
14. Gaede V., Günther O.: *Multidimensional Access Methods*, ACM Computing Surveys, 30 (2), 1998.
15. Gastwirth J.: *On Robust Procedures*, Journal Amer. Stat. Ass., Vol 65, 1966.
16. Graham R. L.: *An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set*, Info. Proc. Lett., Vol. 1, 1972.
17. Green P. J., Silverman B. W.: *Constructing the Convex Hull of a Set of Points in the Plane*, Computer Journal, Vol. 22, 1979.
18. Guttman A.: *R-trees: A Dynamic Index Structure for Spatial Searching*, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984.
19. Huber P. J.: *Robust Statistics: A Review*, Ann. Math. Stat., Vol. 43, No. 3, 1972.
20. Jagadish H. V.: *A Retrieval Technique for Similar Shapes*, ACM SIGMOD Int. Conf. Manag. Data, 1991.
21. Jarvis R. A.: *On the Identification of the Convex Hull of a Finite Set of Points*, Info. Proc. Lett., Vol. 2, 1973.
22. Knorr E. M., Ng R. T.: *Algorithms for Mining Distance-Based Outliers in Large Datasets*, Int. Conf. on Very Large Data Bases (VLDB), 1998.
23. Kriegel H.-P., Seidl T.: *Approximation-Based Similarity Search for 3-D Surface Segments*, GeoInformatica Int. Journal, Vol. 2, No. 2, 1998.
24. Korn F., Sidiropoulos N., Faloutsos C., Siegel E., Protopapas Z.: *Fast Nearest Neighbor Search in Medical Image Databases*, Int. Conf. on Very Large Data Bases (VLDB), 1996.
25. Mitchell T. M.: *Machine Learning*, McCraw-Hill, 1997.
26. Preparata F. P.: *An Optimal Real Time Algorithm for Planar Convex Hulls*, Comm. ACM 22, 1979.
27. Preparata F. P., Shamos M. I.: *Computational Geometry*, Springer New York, 1985.
28. Rosenfeld A.: *Picture Processing by Computers*, Academic Press, New York, 1969.
29. Sander J., Ester M., Kriegel H.-P., Xu X.: *Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications*, Data Mining and Knowledge Discovery, Vol. 2, No. 2, 1998.
30. Shamos M. I.: *Computational Geometry*, Ph.D. Thesis, Dept. of CS, Yale University, 1978.
31. Seidl T., Kriegel H.-P.: *Efficient User-Adaptable Similarity Search in Large Multimedia Databases*, Int. Conf. on Very Large Data Bases (VLDB), 1997.