Chapter 5

Image Analysis: A case study

Image retrieval systems form an interesting, but small subset of the potential application of image databases. Our conjuncture is that image analysis researchers can also benefit from such systems in their day-to-day activities.

Currently, in image processing research, each analysis step is programmed in a third generation programming language. These languages are not known for their ease of programming, code maintenance, and reuse-ability. An object oriented programming style, such as seen in Java and C++, partly solves the maintenance and reuse problems. Java still suffers from performance problems. These problems are countered with proprietary data structures and associated operations to obtain better performance. Likewise, C++ still suffers major compiler compatibility problems when distribution to different platforms is the objective.

Besides the cumbersome programming language, programmers in image analysis often focus on hard recognition problems in isolation. It is not uncommon to be confronted by throw away code, because there is limited tendency to develop code for reuse. As a result, they also lack writing proper documentation and the image analysis community is stuck with software hard to maintain and reuse. Although, exceptions on these rules exist, such as the Horus image library[112], the current software approach hinders progress in this area.

The approach taken in this thesis is to use database technology as a step forward. A database system challenged to support image analysis has to overcome the following problems:

Erroneous Data The initial image data (and all derived data) contains errors due to inaccuracy of the measuring devices. Errors largely come in two flavors: the discretisation error from scanning (devices) and use of inadequate data structures. To illustrate the former, scanning devices have physical constraints on the LED’s. To illustrate the latter, one could store a line simply by the end points, which again makes it dependent on the discretisation technique (i.e. grid precision)[51].

91
Another option is to use a triplet of center point, orientation and length, which is much less dependent on the discretisation technique.

**Proximity and Probabilistic reasoning** The fuzzy data involved requires a mechanism to support proximity-based queries and a probabilistic computational model. This is currently not supported by database management systems. A step into this direction is researched by [32].

**Multiple representations** Many possible derived data features exist and each one can be represented in several ways, while transformations between them is generally not loss-less. The system should be able to handle all in a uniform way and decide on what to materialize.

**Existing Algorithms** Many image analysis algorithms exist and one may not expect image researchers to rewrite them into database queries. Instead, these often time consuming algorithms should be callable from the query language directly. To be useful in a database context they should be side effect free, otherwise any optimization is impossible. A criterion hardly ever met.

In this chapter, we propose to add core image processing functionality to the database management system, making it a better tool for image analysis research as well. The approach taken is to identify missing parts using a single representative case: "line clustering". The "line clustering" problem has been chosen, because it is a long standing problem in image analysis research and, therefore, an acceptable solution is known. The outcome of this experiment are operator and database requirements.

The human visionary system seem to recognize straight lines in images easily. Even when the lines are broken into segments, partially visible, and with a small angular distortion. Actually, a human clusters the visible line segments, such that with a certain confidence he or she claims they belong to the same line. This confidence is based on syntactic information, i.e. "most lines are straight" and a priori semantic information "we deal with power lines". The later aspect is a focus for model driven image analysis.

In the rest of this chapter we show, by means of the case study, that an IDBMS simplifies experimentation with the different image analysis algorithms and it takes less time to implement them. However, it is not our intention to solve the line clustering problem at large. Consequently, we don't explore all performance aspects, but focus on the translation of an image analysis problem into a database problem.

### 5.1 The line clustering problem

Many computer vision and image processing applications involve the seemingly simple problem of line detection. A clustering of segments into lines
that faithfully represent the original image is a pre-requisite for many image understanding algorithms.

Conversion of an image into a set of lines is a two-phase process. In the first phase, the image is converted into an edge map using a segment detection algorithm, such as [15]. The second phase deals with extracting the straight lines from the edge map.

A major problem encountered in practice is the lack of accurateness in the segment extraction algorithms [14, 39]. Segments may be broken, rotated or translated from their actual position in the source image. These shortcomings of the detection/extraction algorithms show up more when the original image contains different line styles, such as dashed and dotted lines, or the image is cluttered with lines. Some segment extraction algorithms may be better in handling rotations, others in handling width displacements. Since edge detection and segment extraction are long standing topics in computer vision it is unlikely that error free algorithms will be found shortly. The result is that image analysis starts with a large collection of segments that barely resemble the lines in the original image.

The way out of this dilemma is to focus on segment clustering algorithms to derive approximately correct lines. By developing clustering algorithms with a few controlling parameters it becomes possible to automate line detection up to a point that human intervention is reduced to a minimum. The proper parameter settings can be obtained by an expert user in an interactive application, which shows the clustering results for various parameter settings.

![Figure 5.1: Example Image with Line Segments](image)

For practical purposes [40, 76, 64] assume an image with a sparse number of lines, e.g. contours of a single sharp object. The example image, Figure 5.1, is taken from [64] which deals with powerline maps.
5.1.1 Clustering Hierarchy

The line clustering problem can be redefined as clustering the segments obtained from the extraction algorithm to form lines closely resembling the original line in the image. The predominant way to solve this is by constructing a clustering hierarchy, which groups of segments are more likely to belong to the same original line.

![Figure 5.2: Example Clustering Hierarchy](image)

See Figure 5.1.1 for a sample segment set taken from a utility map and the corresponding cluster hierarchy. In such a hierarchy each node represents a hypothetical line \( s_i \), which best fits the underlying segment set. The leaves of the hierarchy contain the initial segments. Each node combines two segment sets into \( S_i \). Note that \( s_i \) may, but need not collide with a detected edge.

5.1.2 Clustering Factors

The error classes caused by the detection phase are: orthogonal distance \( d(s_i, s_j) \), rotational displacement \( \theta_{s_i} - \theta_{s_j} \), and difference in line width between a segment and the hypothetical line. For dashed lines we include the factor coverage, i.e. \( \text{coverage}(s_i, \{s_i\}) \), measured as the sum of the segment lengths projected on the hypothetical line \( s_i \), as the ratio to its total length.

![Figure 5.3: Example Rotational Error](image)

The clustering factor orthogonal distance \( d(s_i, s_j) \) specifies that segments far apart are less likely to belong to the same original line. The same holds for segments with a large angular displacement. This factor depends on
the length of the line segments, since small segments are subject to larger
discretisation errors than long segments. Figure 5.1.2 clearly shows the error
carried by a single pixel shift. A detector cannot separate a point from a
small line.

The width difference factor models that segments of a single original line
cannot differ too much in width. The coverage ratio needed to detect dashed
lines states that the more of the hypothetical line is covered by the segments
the higher the support for that hypothetical line is.

5.1.3 Clustering Function

Segment sets \{s_i\} are constructed using a cluster function \(M(\{s_i\})\), which
produces a likelihood value in the range \([0..1]\) for a set of segments supporting
a single hypothetical line \(s_l\). The hypothetical line \(s_l\) for a set of segments
\{\(s_i\)\} is obtained through a least square fitting algorithm. From [64] we
obtained the following cluster function:

\[
M(\{s_0, s_1, \ldots, s_n\}) = \sqrt[n]{\lambda(s_l, s_0) * \lambda(s_l, s_1) * \ldots * \lambda(s_l, s_n) * \frac{L(s_l, \{s_i\})}{l_{sc}}} \\
(5.1)
\]

Where the support function \(\lambda(s_l, s_i)\) is defined as:

\[
\lambda(s_l, s_i) = G_{\sigma_{\theta_s i}}(\theta_{s_l} - \theta_{s_i}) * G_{\sigma_\sigma_{d(s_l, s_i)}}(d(s_l, s_i)) \\
(5.2)
\]

And where \(G_\sigma(x)\) is defined as the Gaussian distribution:

\[
G_\sigma(x) = \frac{1}{\sqrt{\pi \sigma^2}} e^{-\left(x/\sigma\right)^2} \\
(5.3)
\]

The standard deviation of the rotational displacement \(\theta\) of \(s_l\) and \(s_i\) is
controlled by \(\sigma_{\theta_{s_i}}\). The permissible deviation is larger for shorter seg-
ments. In Equation 5.2 \(d(s_l, s_i)\) is the orthogonal distance between \(s_l\) and
\(s_i\), measured as the shortest distance between the center of \(s_i\) and the line
\(s_l\). The standard deviation in orthogonal distance is controlled by \(\sigma_\sigma\).

The last part of Equation 5.1 controls the gap size between segments. \(L(s_l, \{s_i\})\) is the sum of the lengths of all segments projected onto the line \(s_l\)
and \(l_{sc}\) is the total length of covered segment of the line \(s_l\). Larger coverage
indicates a higher chance that these segments belong to the same original.
These functions are used to calculate clustering values to build a hierarchy
of segment sets.

The hierarchy is built bottom up in an iterative process with the initial
tree containing the individual segments as leaves. Using a hill-climber algo-

rithm the tree is constructed by repeatedly combining two nodes. Once a
clustering set is formed it cannot be split again.
Two Segment sets are clustered if there is no other combination with a higher support value. In particular, at each step, node pairs with a maximum support value are located and merged into a new node. The support function value is always based on the original segments, because even the best fitting algorithm for finding a hypothetical line has inaccuracies. Merging inaccurate hypothetical lines could result in lines that do not represent any of the lines in the original image. For example, a circle broken into segments could mistakenly be clustered into a single straight line.

The clustering process continues until all nodes have been merged into sets and the clustering tree is completed by a single root. The hierarchy is then a complete clustering of all the segments into a single hypothetical line. From the clustering hierarchy, it becomes possible to select a clustering according to a user-controlled threshold value. This threshold could be found using a process of visual feedback, i.e., an edgemap overlaid with the clustering for a given threshold \( \delta \). Clustering of segment sets which are too far apart or have very different orientations are ignored.

The algorithms proposed in [76, 64] calculates the likelihood value for each segment subset. This leads to a combinatorial explosion, because of the large number of segments involved. The algorithm does not scale. Their algorithmic complexity is cubic in the number of segments, \( (n) \), considered for clustering \( O(n^3) \). Furthermore, the clustering function is expensive, because it recalculates the hypothetical line from the segment set considered.

### 5.2 Database Optimization

The solution described in the previous section involves user controllable parameters \( (\sigma_{a(l(s))}, \sigma_o, \delta) \), and are based on complex calculations. Moreover, the clustering algorithm uses a single hypothetical line at a time processing. Although it gets the job done, it is far from optimal.

Conversion of this approach to a database set-oriented approach seems beneficial. The following optimizations strategies would be applied by a database programmer confronted with the task:

- **Domain independent methods**
  - Use spatial indices, to reduce the \( O(n^2) \) problem with appropriate filters.
  - Factoring out expensive calculation, to reduce the CPU cost.

- **Domain specific filtering**
  - Use angular distortion to start the search for lines under the assumption that their angular distance is always limited.
  - Use line length to start with long lines, these are considered less erroneous.
5.2. DATABASE OPTIMIZATION

- Domain specific algorithms
  - Use a set oriented approach gives a better handle to reduce repeated calculations.
  - Divide & conquer using repetitive splitting into disjoint sets until the optimum is found.

Using the same algorithm we could reduce the calculation of the expensive clustering function by careful selection of candidates for clustering. We can use domain independent indices to speed up this search, when a maximum segment distance is known. For example searching for candidate clustering pairs could be done using a spatial index structure, such as the R-tree [52]. We could also factor out the constant factors in the clustering function, which cannot be done transparently by a C-compiler.

5.2.1 Mathematical Optimization

Solutions as described by [40, 76, 64] use functions to calculate a clustering value for a set of segments, i.e. they all use similarity measures. The clustering function combines the clustering factors into a single value that quantifies the support for the hypothetical straight line. A better approach is in the first step to use these factors to filter out approximately 90% of non-interesting cases, and then solve the remainder using these expensive measures.

The clustering algorithm discussed uses a clustering hierarchy, but specific applications will use a threshold value to select only a subset of this hierarchy, i.e. all clusterings \( \{s_i\} \) where \( M(\{s_i\}) \) exceeds the threshold \( \delta \). Having this threshold we could reverse engineer the clustering function and find information to reduce our search space.

Two optimization methods could be applied: search optimization and filtering. The first, improves searching for candidate clusterings. For example a search for candidates based on spatial locality. In the naive algorithm the candidates considered are all segment set combinations. A filtering optimization would significantly reduce the set of candidate clusterings using a cheap operation as follows. Knowing two segments can never be clustered when their orientations differ too much, we could cheaply filter these out using a selection on their orientation difference. To see how we apply these optimizations, take a closer look at the clustering function 5.1.

Consider Equation 5.1 and assume \( n - 1 \) segments perfectly fit the hypothetical line \( s_l \). This together with threshold \( \delta \) leads to Equation 5.4.

\[
\sqrt{\lambda(s_l, s_n) \ast \frac{L(s_l, \{s_i\})}{l_{sc}}} \geq \delta \tag{5.4}
\]
When only considering the angular displacement, i.e. assume coverage is perfect and orthogonal displacement is 0, leaves Equation 5.5. Only considering the orthogonal displacement gives Equation 5.6.

\[
\frac{1}{\sqrt{\pi \sigma^{2}_{a_i(s_n)}}} e^{-\frac{(\theta_i - \theta_n)^2}{2}} \geq \delta \Rightarrow \theta_i - \theta_n \leq \sqrt{-\log(\delta n) \cdot \frac{\pi \sigma^{2}_{a_i(s_n)}}{\sigma_{a_i(s_n)}}}
\]

(5.5)

\[
\frac{1}{\sqrt{\pi \sigma^{2}_{o}}} e^{-\frac{d(s_i, s_n)^2}{2 \sigma_{o}^2}} \geq \delta \Rightarrow d(s_i, s_n) \leq \sqrt{-\log(\delta n) \cdot \frac{\pi \sigma^{2}_{o}}{\sigma_{o}}}
\]

(5.6)

Assuming no rotational and orthogonal displacement leads to Equation 5.7. The maximum gap between two segment sets, \(S_1\) and \(S_2\), can be obtained when both the sets are perfectly aligned, then \(L(s_i, \{s_i\})\) is the sum of all segment lengths and \(l_{sc}\) is the sum of all segment lengths plus the gap size.

This gives an upper bound on the distance between to perfectly aligned segments, i.e. a maximum gap size.

\[
\frac{L(s_i, \{s_i\})}{l_{sc}} \geq \delta \Rightarrow \text{gap} \leq \frac{1 - \delta}{\delta} \cdot (L(S_1) + L(S_2))
\]

(5.7)

To allow for candidate search based on segment distance we need to combine this with the maximum rotational displacement and maximum orthogonal distance because these factors could influence the distance between segments. There are three extremes to consider when searching the maximum segment distance, maximum gap size, maximum orthogonal distance and maximum rotational displacement. The maximum segment distance is the maximum of these three extremes. See Figure 5.2.1 for these extremes. The gap size and orthogonal distance can be calculated directly.

Figure 5.4: (a) max. gap size (b) max. orthogonal distance (c) max. rotational displacement

The angle \(\theta\) between the hypothetical line \(s_i\) and the segment \(s_n\) depends on the segment distance, \(d(s_i, s_n)\). This leads to Equation 5.8.

\[
d(s_i, s_n) \leq \sqrt{\text{max gap}^2 + \sin(\text{max angle}) \cdot 1/2l(s_n)^2}
\]

(5.8)
5.2.2 Split based algorithm

The problem of the clustering algorithms discussed so far is their complexity. For each level of the clustering hierarchy \( n^2 \) combinations are checked.

One way to reduce the possible combinations is to look at the data characteristics at hand. For example Figure 5.2.2 shows the segments length and orientation of an A4 sized utility map. This figure clearly shows that the large segments can be easily split into two groups around the peaks, 60 and 175 degrees. This could give two (disjoint) input sets for the optimized clustering algorithm.

When we apply the same method for the segment width and polar r coordinate we can identify groups of segments, which are strong candidates for clustering. The splitting in groups should allow for overlapping groups, since not all clustering factors have clear split positions.

![Figure 5.5: Angle Length](image)

5.3 A hybrid solution

In this section we illustrate integration of an existing algorithm for line clustering with our extensible database management system Monet. The approach taken is to store the segments and clustering hierarchy in a database table. The clustering algorithm extracts portions of this table for processing and stores the results back into the database. This hybrid algorithm is directly based on [64], and illustrates use of a DBMS as a single object server. It does not involve delegation of work.

5.3.1 Database representation

Integration of the line clustering algorithm and database technology requires definition of a data model. The ideal situation, from an image processing
point of view, is when the C++ structures in the application program are understood and managed by the DBMS. This ideal is pursued by object-oriented systems, such as persistent C++ and ODBMS.

Although this leads to a seamless interconnection, i.e. no impedance mismatch, the state of the art DBMS optimization techniques have limited effect. The underlying reason is the iterative processing model still adhered to in the application code, rather than a more declarative set-based approach.

The solution is to use an extensible relation DBMS, which offers a declarative query language which could be extended with new abstract data types, commands and search accelerators. Examples are Postgres and Informix, where the user can introduce abstract data types, whose representation is a byte sequence. Conversion of application data structures into their database equivalents then merely amounts to conversions into/from a byte sequence.

Another example is our extensible database system Monet[8], which supports user-defined abstract data types, called atoms, search accelerators, and new commands. These are introduced using the Monet extension and C/C++ programming language.

5.3.2 Data Model

Recall that in Monet the data is fully vertically decomposed into Binary Association Tables (BATs), see Section 2.3. The data model for this application domain consists of object identifiers (oids), segments, and sets of oids. Oids are part of the standard data types of Monet, and thus all operations needed are at hand. For the segments a new atom (abstract data type) was needed, see Figure 5.6 for the module specification.

The input for the line clustering problem is a large set of line segments obtained by scanning a grey-value image, using an edge detector followed by a straight line extraction algorithm. This initial information is placed in a BAT, which contains for each segment an object identifier and a segment representation, see Figure 5.7 for a slice taken from this BAT.

The nodes of the hierarchy are also stored in a BAT. For each node again an object identifier is used together with a set of oids. These sets of oids represent the sets of segments. See Figure 5.3.2 for the required schema.

5.3.3 Clustering Algorithm

The subsequent step is to recode the algorithm [64] in MIL. Although this is a straight forward mapping, it pays off to judiciously use indices to trim the space explored. The cost incurred by combinatorial explosion encountered in the naive line clustering algorithm can be reduced using a spatial filtering technique. The rationale is that two segments are merge candidates if they are spatially local.
5.3. A HYBRID SOLUTION

...
other words, only the clusterings segments which are spatially local, and in
the same angle and width ranges, are the candidates considered.

In stage 3 all clustering values are calculated using the original cluster
function. This cluster function has been shown to be accurate, but it is also
rather expensive. It needs to find the hypothetical line with the maximum
support value. Finding the best hypothetical line through the set of segments
is done using a least square fitting algorithm. Because the support function
depends on the hypothetical line, which changes when new segments are
added, no reuse of the calculations can be done.

The support values are known, the clusterings with a maximum support
value for each of the containing clusters should be selected. Thereafter the
clustered segments and corresponding boxes are deleted from their BAT’s,
and the new sets are inserted in those BATs. For each new cluster a new
bounding box is calculated and inserted into the R-tree BAT. The bounding
box of the set of segments is that box containing all segments. This stepping
process will continue until nothing more can be clustered.
Algorithm hybrid

1. **Spatial join**, join all possible clusterings based on spatial locality

2. **Heuristic selection**, make a selection based on the angle heuristics. Filter out pairs with very dissimilar orientation.

3. **Support values**, calculate the precise support values for the hypothetical line.

4. **Select maximums**, select clusterings which have a maximum support value

5. **Update**, update the BATs containing the nodes and boxes

Since segment clusters are represented by their segment identifiers, interface functions should find the corresponding segments and convert those into the data structures needed for the clustering functions. This means there is a lot of conversion overhead. For the different parts of the calculation of the heuristics and support values a number of segment and node characteristics are needed, like center point, angle, and mean angle. Also these characteristics have to be recalculated each time the different functions are called.

Using different selection criteria has proven to significantly reduce the number of times the support value is calculated. As a consequence the support function is no longer the dominant performance overhead. The heuristics are now responsible for most of the execution time.

The use of the R-tree reduces the search space depending on the threshold. Since the tree is built only once the cost is low compared to the costs of the other functions.

### 5.4 Database Solution

In this section we demonstrate how a modern DBMS can be used to tackle the line clustering problem. In Section 5.4.2 we show that algorithms based on these primitives outperform the solutions given in the previous section.

#### 5.4.1 Line cluster model

The input to the line clustering problem is a large set of line segments obtained by scanning a grey-value image. The segments are represented as atomic values in the database using the extensions introduced in Section 5.3.

The line clustering problem can be rephrased as finding (disjoint) subsets that provide maximum support for a hypothetical line derived from the
segments in the subset. Alternatively, each segment is assigned to a single subset and moving a segment from one cluster to another reduces the clustering value for both.

The key operation of the line clustering algorithms is to analyze a segment set. That is, the basic object of manipulation is a set of segments. In the previous algorithm in fact, the object was constructed as soon as transfer occurred to the clustering algorithm. This is not necessary. Within the database environment a line can be represented by a multi-valued function from a group identifier to a set of segments. How these segments are glued together to form a line is a separate issue.

A group can be reduced to a single value using the Reduce command. Using the Map command a function can be applied to each group member and a group parameter, see Table 5.4.1.

The algorithms are all linear in the size of their operands. They can also be parallelized easily.

The easiest way to represent the initial segment set within Monet is to introduce a mapping from \( g_i \rightarrow \{s_j\} \), where \( g_i \) denotes a group identifier and \( s_j \) a segment identifier. This can be represented with a single Binary Association Table called \textit{groups} and forms the first level clustering of segments. For each group we calculate a bounding box \( g_i \rightarrow \{b_i\} \). The bounding boxes are stored in the BAT named \textit{boxes}. The remainder of the clustering runs as follows:

**Algorithm DBMS**

**Filtering on spatial locality** Determine the candidate pairing of groups using an overlap operator over their bounding boxes, i.e. \( C := \text{overlap}(\text{boxes, boxes.reverse}) \)

**Create Groups** Create a candidate group for each joined pair and store it in a BAT, i.e. \( \text{CG} := \text{C.mark()}.\text{join(nsegs)} \)

**Filtering on max angle diff** Determine the maximum angle difference between the average and the segments angle for all groups, and select those segments where the angle difference is less than than \text{max_angle_diff} i.e.

\[ C\text{angle} := \{\text{sum}\}(NG) \]

\[ C\text{nr} := \text{sum}(C.\text{join(nnr)}) \]
5.4. DATABASE SOLUTION

\[ \text{Cmangle} := \lfloor/(\text{Cangle}, \text{Cnr}) \]
\[ \text{Cmadiff} := \max([\text{angle_diff}]/(\text{Cangle}, \text{Cmangle})) \]
\[ \text{CG} := \text{CG}.\text{semijoin}(\text{Cmadiff.select(0.0, max_angle_diff})) \]

**Calculate Hypothetical line**  Calculate interpolated straight line through the center points. First get center of all line segments in the candidate group, from this calculate the center point for the hypothetical line, a second point is calculated using interpolation, this point is used to find the lines orientation i.e. \( \text{CCenter} := \lfloor/\{\text{sum}\} (\text{CCenter}), \text{Cnr}) \)
\[ \text{dx} := \lfloor/[x](\text{CCenter}), [x](\text{NCenter}) \]
\[ \text{CX} := \lfloor*[\text{dx}, \text{dx}] \]
\[ \text{CY} := \lfloor*[\text{dx}, [y](\text{CCenter}) \]
\[ \text{CXsum} := \{+\} (\text{CX}) \]
\[ \text{CYsum} := \{+\} (\text{CY}) \]
\[ \text{Angle} := \lfloor[\text{atan2}]/([\text{CXsum}, \text{CCenter})], [\text{CYsum}, [y](\text{NCenter})) \]
\[ \text{Line} := \lfloor[new\_line]/(\text{NCenter}, \text{Angle}) \]

**Calculate support values** Calculate the product of the width, rotational, and orthogonal distance heuristics

**Select new clusters** Select groups with maximum support values for both containing clusters

**Update retained information** Delete clustered groups and insert the new groups

The result of the distance join is used to construct groups of segments. This again uses the search optimization based on spatial locality. These candidate groups are further reduced using filters on orientation and segment width. The operations needed for the calculation of the hypothetical lines are done in a setwise fashion, which gives us the possibility to reuse the intermediate results, such as the sum of the segment centers and the mean orientation. The calculations needed for the filters can be reused during the calculation of the support values. With these support values the best clusterings could be selected. At the last step we have to update the retained information, such as bounded boxes and mean orientation.

We can reuse intermediate results because we changed from a single segment at a time approach to a more database approach of set at a time bulk operations. There are no extra implementation efforts involved for this set-at-a-time approach, because the primitives needed are already supported by the database kernel.

5.4.2 Experiments and Results

Extensive experiments were performed to compare both effectiveness and efficiency of the algorithms. The input for those experiments were segment
sets obtained from real-life utility maps. Following [64] we used the standard deviation \( \sigma_o \) and \( \sigma_a \) 1.5 and 0.18 respectively.

<table>
<thead>
<tr>
<th># segments</th>
<th>time(s)</th>
<th>time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sequential</td>
<td>DBMS</td>
</tr>
<tr>
<td>300</td>
<td>23</td>
<td>2.3</td>
</tr>
<tr>
<td>600</td>
<td>76</td>
<td>2.9</td>
</tr>
<tr>
<td>1200</td>
<td>298</td>
<td>4.7</td>
</tr>
<tr>
<td>1600</td>
<td>454</td>
<td>5.4</td>
</tr>
<tr>
<td>2000</td>
<td>660</td>
<td>6.9</td>
</tr>
<tr>
<td>2736</td>
<td>1330</td>
<td>8.7</td>
</tr>
<tr>
<td>4000</td>
<td>2749</td>
<td>15</td>
</tr>
<tr>
<td>5472</td>
<td>5280</td>
<td>93</td>
</tr>
</tbody>
</table>

Table 5.2: Results for the sequential and DBMS algorithms

The first experiments show the efficiency of the database approach against an algorithm used by [64] in image analysis. It enables comparison of the traditional C++-based implementation with the DBMS algorithm. The set of segments \( S \) is extracted from a single utility map. A family, of sets \( F \) is constructed, such that the elements in \( F \) form a subset lattice, i.e. \( \forall f_i \in F \exists f_j \in F : f_i \subseteq f_j \) The experiment was done with a fixed threshold \( \delta \) of 0.5. The result are shown in Table 5.2.

As can be seen from the first experiment the DBMS set at a time algorithm is an order of magnitude faster than the sequential algorithm. This result can be attributed to effectiveness of the spatial index.

The second set of experiments with the hybrid and DBMS algorithms was focussed on the efficiency of these algorithms. Different input sets were taken. The sets were extracted from different utility maps each with its own segment density. Figures 5.4.2, 5.4.2 and 5.4.2 show the histograms of the segment orientation. The experiment was done with various thresholds to evaluate the performance degradation under larger clustering hierarchies. Table 5.3 shows the results.

The set at a time algorithm out performs the Hybrid algorithm on large data sets, because then it profits from the information retained. With small sets recalculation is less expensive.

All the experiments were done on a Sun SPARC-X running the Solaris operating system and using an early Monet V3 version.

### 5.5 Conclusion

It has been shown that an extensible DBMS can be used to tackle the line-clustering problem. The overhead of the conversion between database
structures / application structures is not a dominant factor. Moreover, there exists a small algebraic extension to the DB core functionality, which enables us to tackle the line clustering problem with database kernel support. The performance is promising compared to the original solution written in C++.

<table>
<thead>
<tr>
<th># segments</th>
<th>threshold</th>
<th>hybrid</th>
<th>DBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>67</td>
<td>0.25</td>
<td>1.1</td>
<td>1.4</td>
</tr>
<tr>
<td>67</td>
<td>0.4</td>
<td>0.9</td>
<td>1.2</td>
</tr>
<tr>
<td>67</td>
<td>0.75</td>
<td>0.5</td>
<td>0.8</td>
</tr>
<tr>
<td>2585</td>
<td>0.25</td>
<td>82</td>
<td>67</td>
</tr>
<tr>
<td>2585</td>
<td>0.4</td>
<td>56</td>
<td>36</td>
</tr>
<tr>
<td>2585</td>
<td>0.75</td>
<td>13</td>
<td>7</td>
</tr>
<tr>
<td>4362</td>
<td>0.25</td>
<td>77</td>
<td>45</td>
</tr>
<tr>
<td>4362</td>
<td>0.4</td>
<td>50</td>
<td>35</td>
</tr>
<tr>
<td>4362</td>
<td>0.75</td>
<td>9</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5.3: Result for the Hybrid and DBMS algorithms
Figure 5.10: Histogram of the line segment angles, A4 sized image

Figure 5.11: Histogram of the line segment angles, A0 sized image