Towards Content-Based Retrieval of Technical Drawings through High-Dimensional Indexing

Manuel J. Fonseca Joaquim A. Jorge
Department of Information Systems and Computer Science
INESC-ID/IST/Technical University of Lisbon
R. Alves Redol, 9, 1000-029 Lisboa, Portugal
mjf@inesc-id.pt, jorgej@acm.org

Abstract

This paper presents a new approach to classify, index and retrieve technical drawings by content. Our work uses spatial relationships, visual elements and high-dimensional indexing mechanisms to retrieve complex drawings from CAD databases. This contrasts with conventional approaches which use mostly textual metadata for the same purpose.

Creative designers and draftspeople often re–use data from previous projects, publications and libraries of ready to use components. Usually, retrieving these drawings is a slow, complex and error–prone endeavor, requiring either exhaustive visual examination, a solid memory, or both. Unfortunately, the widespread use of CAD systems, while making it easier to create and edit drawings, exacerbates this problem, insofar as the number of projects and drawings grows enormously, without providing adequate retrieval mechanisms to support retrieving these documents.

In this paper we describe an approach that supports automatic indexation of technical drawing databases through drawing simplification techniques based on geometric features and efficient algorithms to index large amounts of data. We describe in detail the indexing structure (NB–Tree) we have developed within the context of a more general approach. Experimental evaluation reveals that our approach outperforms some of the best indexing structures published, enabling us to search very large drawing databases.

Keywords

Content–Based Retrieval, Graph Matching, High-Dimensional Indexing

1. INTRODUCTION

Recent studies [13] refer that the use of libraries with old cases is important to help designers identifying relevant features to include or problems to avoid. Additionally, in some design firms, designers often work by making or copying diagrams from their design team colleagues for further development [12]. Furthermore, in some informal conversations with designers, we found out that they re-use old drawings during the creation phase of a new project, to get some ideas or solutions already achieved. Eventhough, the re-use of drawings save time, the searching process is usually slow and problematic.

Unfortunately, the widespread use of CAD systems, while making the creation and edition of new drawings easier, exacerbates this problem, because the number of projects and drawings grows enormously, without providing adequate retrieval mechanisms to support retrieving documents. Present-day CAD systems rely on conventional database queries and direct-manipulation to achieve this.

Some of the solutions [1, 10] to this problem use textual databases to organize the information. Drawings are classified by keywords and additional information, such as, designer name, style, date and a textual description. However, solutions based on textual queries are not satisfactory, because they force the designers to knowing in detail the meta-information used to characterize drawings and they require humans to produce it. Opposed to the textual organization, we propose a visual classification based on shape and spatial relationships, which we consider more suited to this problem, because it uses the visual memory owned by designers and explore their ability on sketching as a query mechanism.

Additionally, recent studies [16, 13] show that designers use a small set of common graphical elements to describe the same drawings, validating our approach of using sketches to specify queries to a database of technical drawings.

The rest of the paper is organized as follows: In section 2 we give an overview of the related work in sketch–based retrieval. In section 3 we describe the system architecture

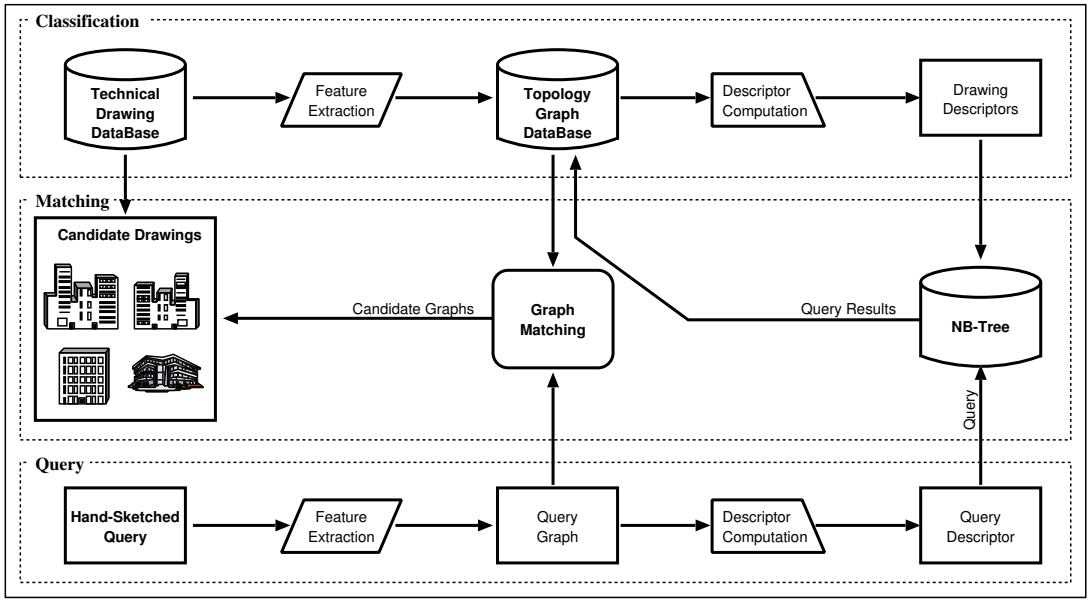

Figure 1. System architecture.

and all its components. Section 4 explains the basic idea of the indexing structure, the NB-Tree, and presents a performance comparison with other indexing structures. We conclude the paper by discussing the results and present further work directions for our system and related areas.

2. RELATED WORK

Recently there has been considerable interest in querying Multimedia databases by content. However, most of this work has focused on image databases as surveyed by Shi-Kuo Chang [9]. Moreover, in [23], the author analyses several image retrieval systems that use color and texture as main features to describe image content. On the other hand, drawings in electronic format (CAD) store data in structured form (vector graphics) requiring different approaches from image-based (color, texture) methods. Some initial work [1, 10] attempted to index technical drawings through textual databases. However, this fails to use the rich visual association mechanisms and designer's use of sketches to recover information.

Although, the three systems described below address the problem of content-based retrieval of drawings, they follow different principles and different algorithms to achieve their goals.

The first, developed by Mark Gross and Ellen Do, in the context of the Electronic Cocktail Napkin [16, 12, 15] addressed a visual retrieval scheme based on diagrams, to indexing databases of architectural drawings. Users draw sketches of buildings, which are compared with annotations (diagrams), stored in a database and manually produce by users. Eventhough, this system works well for small sets of drawings, the lack of automatic indexation and classification makes it impossible to use for large collections of drawings.

The S3 system [7] supports the management and retrieval of industrial CAD parts, described using polygons and thematic attributes. It retrieves parts using bi-dimensional contours drawn using a graphical editor or sample parts stored in a database. Although, this system presents good results in retrieving industrial CAD parts, it relies exclusively on matching contours, ignoring spatial relationships and shape information, making it unsuitable for retrieving complex multi-shape drawings.

In [22], Park describes an approach to retrieve mechanical parts based on the dominant shape. Objects are described by recursively decomposing its shape into a dominant shape, auxiliary components and their spatial relationships. The small set of geometric primitives and the not so efficient matching algorithm makes it hard to use with large databases of drawings.

In general, our approach improves on Berchtold [7] and Park [22] systems, since we aim to retrieve technical CAD drawings and privilege the use of spatial relationships and dominant shapes. Indeed, our method is more ambitious in the sense that we plan to do automatic simplification, classification and indexation of existing drawings, to make retrieval process more effective and accurate.

3. SYSTEM ARCHITECTURE

Our approach solves these problems by developing a mechanism for retrieving technical drawings, in electronic format, through hand-sketched queries, taking advantage of designer's natural ability at sketching and drawing.

Figure 1 illustrates the main components of our approach, which we describe below.

3.1. Classification

Most technical drawings contain detailed descriptions of objects, which are not necessary for a visual search and in fact, increase the cost of searching. We include in our

approach a process to remove visual details (i.e. small-scale features) while retaining the perceptually dominant elements and shapes in a drawing. Our method divides the technical drawing in several dominant blocks, that later will also be divided in other blocks, and extract the spatial relationships between them. We only use two spatial relationships, **Inclusion** and **Adjacency**. These relationships are weakly discriminating, however they are invariant with rotation and translation.

We then combine shape information with the spatial relationships into a topological graph and store it into a database for later use in matching candidate graphs. Figures 2, 3 and 4 illustrate the different steps of the classification process: technical drawing, block and spatial relationship extraction and topological graph creation, respectively.

Since graph matching is a NP problem, we try to overcome this using spectral information on the graph. For each graph we compute a descriptor based on their spectrum [11, 25]. To support sub-graph matching, we also compute descriptors for sub-graphs of the main graph. The computation of the graph spectrum is based on the calculation of the eigenvalues of the adjacency matrix of the graph. The resulting descriptor is a multidimensional point, whose dimension depends on graph complexity. Additionally, it

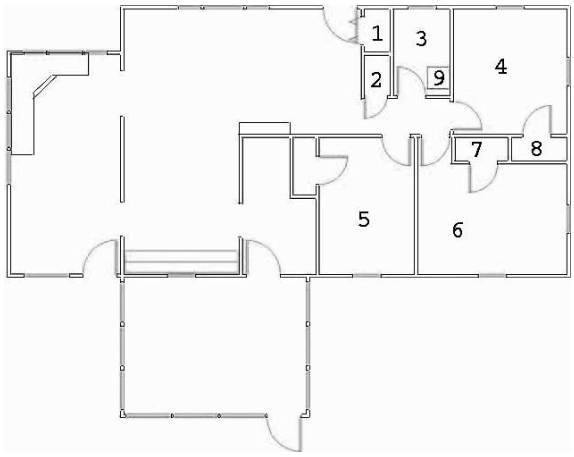

Figure 2. Technical Drawing.

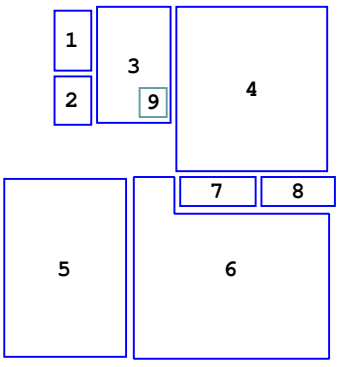

Figure 3. Block division for part of the plant.

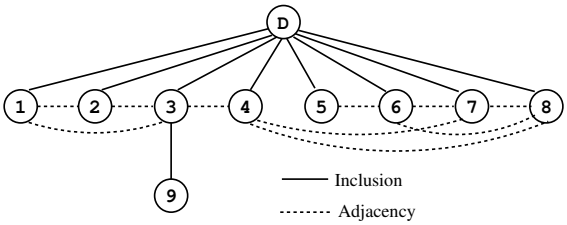

Figure 4. Topology graph.

captures local topology, is invariant to sub-graph re-order and is stable, since small changes in the graph produce little changes in the descriptor. However, the resulting descriptor is not unique. More than one graph can have the same descriptor, which gives rise to collisions. In [25] the authors argue that this collision frequency is small.

Since we need to index most sub-graphs of a given graph to allow for sub-graph matching, we end up with a large database comprising tens of thousands or potentially hundreds of thousands of descriptors, even to index hundreds to thousands of technical drawings. Thus, at the core of our approach, we need to have an efficient indexing structure for storing descriptors. This will be detailed in section 4.

3.2. Query

Our system includes a Calligraphic Interface to support the definition of hand-sketched queries, to supplement and overcome the limitations of textual queries. The query component performs the same steps of the classification process, with an additional recognition step to identify sketched shapes [14]. After identification of all shapes, the system extracts the spatial relationships, construct the topological graph and compute the corresponding descriptor. This multidimensional descriptor will be used as query to the indexing structure.

3.3. Matching

The results returned by the indexing structure are a set of descriptors similar (near in the space) to the query descriptor. Each returned descriptor correspond to a specific graph stored in the topology database, which will be used in the matching process to perform a deeper comparison.

Our classification and query process perform a first filtering based mainly on topology. This step reduces drastically the number of graphs to compare, selecting only graphs with a high probability of being isomorphic to the query graph.

Since the number of graphs to compare is reduced from thousands to dozens, a simple graph matching algorithm can be used, without any loss of efficiency. The isomorphic graphs will then correspond to candidate drawings stored in the database.

To support approximate matches, our indexing structure needs to provide the means for a fast and reliable K nearest-neighbors scheme, since most interesting candidates will probably yield approximate matches to the query. However, nearest neighbor search in high-

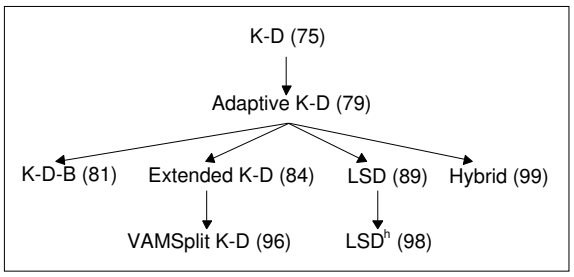

Figure 5. K-D-Tree evolution and organization.

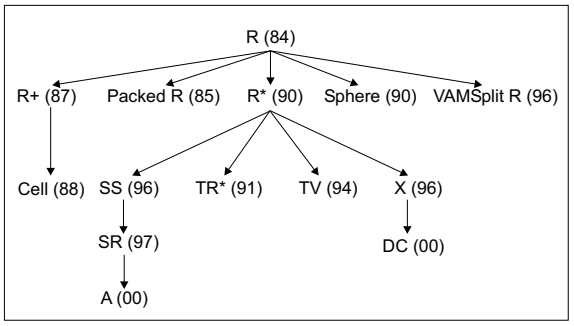

Figure 6. R-Tree evolution and organization.

dimensional data spaces is a difficult problem. In what follows, we will present an algorithm to solve this.

4. INDEXING STRUCTURE

To support processing on large amounts of high–dimensional data, a variety of indexing structures have been proposed in the past few years. Some of them are structures for low–dimensional data that were adapted to high–dimensional data spaces. However, such structures, which provide good results on low–dimensional data, do not perform sufficiently well on high–dimensional vector spaces. Recent studies [26] show that the majority of indexing techniques are less efficient than sequential search, if we consider dimensions greater than 10. Other indexing structures are incremental evolutions from existing ones, where, sometimes, the increase in complexity is not matched by corresponding enhancements in performance. Finally, there are other indexing techniques that result from the combination of several approaches, making their algorithms very complex and hard to code.

The indexing techniques developed so far can be classified into three categories. One that aggregates all structures derived from the K-D-Tree [21], such as the VAM-Split K-D-Tree [27], the LSD-Tree [19], the LSD^h-Tree [18] and more recently the Hybrid-Tree [8]. Figure 5 illustrates the evolution of the indexing structures derived from the K-D-Tree.

A second class of structures is composed by trees derived from the R-Tree [17], such as the R*-Tree [2], the SS-Tree [28], the SR-Tree [20], the VAMSplit R-Tree [27], the X-Tree [6] and more recently the A-Tree [24]. Figure 6 presents the several structures based on the R-Tree.

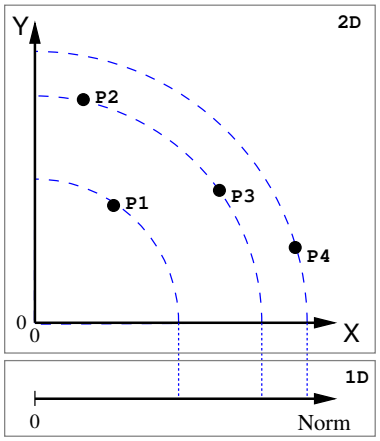

Figure 7. Dimension Reduction for 2D points.

The third category of structures combines several methodologies to improve the performance of the final structure. In this class we can find structures like the VA-File [26], the Pyramid Technique [3], a Voronoi based structure [5] and more recently the IQ-Tree [4].

The main difference between the first two categories is the route they use to divide the data space. Structures in the first category are classified as space–partitioning methods that divide the data space along predefined lines (hyperplanes) regardless of data distribution. The resulting regions are mutually disjoint, with their union being the complete space. Structures from the second class are classified as data–partitioning structures, which divide the data space according to the data distribution. In this category regions can overlap.

The increasingly complex data structures and specialized approaches to high-dimensional indexing make it difficult to ascertain if there might be a reasonably fast and general approach to address many of these problems. We believe there might be some merit in taking a step back and looking at simpler approaches to indexing these data. Motivated by these ideas, we developed the NB-Tree, an indexing technique based on a simple, yet efficient algorithm to search points in high–dimensional spaces, using dimension reduction. Multidimensional points are mapped to a 1D line by computing their Euclidean Norm. In a second step we sort these using a B$^+$-Tree on which we perform all subsequent operations.

4.1. The NB-Tree Algorithm

The NB-Tree provides a simple and compact means to indexing high–dimensional data points. We use an efficient 1D data structure the B$^+$-Tree to index the points sorted by their Euclidean norm (**N**orm + **B**$^+$-Tree = **NB**-Tree).

To achieve flexibility we use dimension reduction: multidimensional points are mapped to a straight 1D line by computing their Euclidean Norm. The second step is to sort them by Euclidean norm, using a B$^+$-Tree. This way,

```
insertPointInNB-Tree(point)
{
  norm = computeEuclideanNorm(point);
  insertPointInB+-Tree(point, norm);
}
```

Figure 8. Insertion algorithm pseudo-code.

all operations are performed on the B^+-Tree. Since this is the most efficient 1–dimensional structure, the NB-Tree inherit its good performance, specifically for point queries.

4.1.1. Creating an NB-Tree

To create an NB-Tree we start by computing the Euclidean norm of each N–dimensional point from the dataset, using the Formula:

$$\|P\| = \sqrt{p_0^2 + p_1^2 + \cdots + p_{N-1}^2} \qquad (1)$$

where $P = (p_0, p_1, \cdots, p_{N-1})$.

The resulting norm and the N–dimensional point are then inserted in a B^+-Tree, using the norm as key. After insertion of all points we get a set of N–dimensional points order by their norm value.

Figure 7 shows an example of the dimension reduction for 2D points, while Figure 8 presents the pseudo-code to implement the insertion algorithm.

4.1.2. Searching

The searching process in the NB-Tree started by computing the norm of the query point. Then we perform a search in the 1–dimensional B^+-Tree. The next search steps will depend of the query type. Current indexing structures usually support three types of queries. The first is **Point Query**, which checks if a specific point belongs or not to the database. The second type of query, **Range Query**, returns the points inside a specific range of values. In our case that range will be specified by an hyper-ball. Finally, the **KNN Query** (K Nearest Neighbors) returns the K nearest neighbors of the query point. This is the most often-used query in content–based retrieval and is the one we describe with more detail in this paper, because it is used intensively by our matching component.

We start the KNN search by doing a ball query (small ball in Figure 9). After this ball query we check if we have enough points inside the ball to satisfy the query. If not, we start an iterative process, where the size of the ball increases gradually until we get all the points specified by the query. Figure 9 illustrates the KNN search (in 2D), while Figure 10 presents the pseudo-code for the KNN algorithm.

To improve the performance of our algorithm we assume that the nearest neighbors will be no far than a given distance from the query point (largest ball in Figure 9). This way we store less intermediate results, speeding up the searching process. The distance used to prune the intermediate results depends on the number of points in the structure and grows logarithmically with data points dimension.

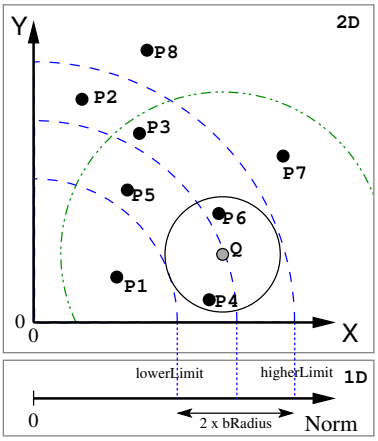

Figure 9. 2D KNN Query example.

```
list* knnQuery(query, knn)
{
 qNorm = computeEuclideanNorm(query);
 bRadius = compInitialRadius();
 pointList = ballQuery(query, bRadius);
 if (enoughPoints(pointList, knn))
  return pointList;
 else {
  //grow the ball by going up and
  // down on the B+-Tree
  do {
   point = btree->search(higherLimit);
   higherLimit += delta;
   while (pointNorm <= higherLimit) {
    dist = dist2Query(point, query);
    if (dist <= pruneDist)
     pointList->addPoint(point);
    point = btree->nextPoint();
   }
   point = btree->search(lowerLimit);
   lowerLimit -= delta;
   while (pointNorm >= lowerLimit) {
    dist = dist2Query(point, query);
    if (dist <= pruneDist)
     pointList->addPoint(point);
    point = btree->prevPoint();
   }
  }while(!enoughPoints(pointList,knn));
 }
 return pointList;
}
```

Figure 10. KNN Query pseudo-code.

4.2. Performance Evaluation

While our approach seems to yield commensurable times to other structures tested, we had difficulties comparing it to other approaches namely the IQ-Tree. Indeed the im-

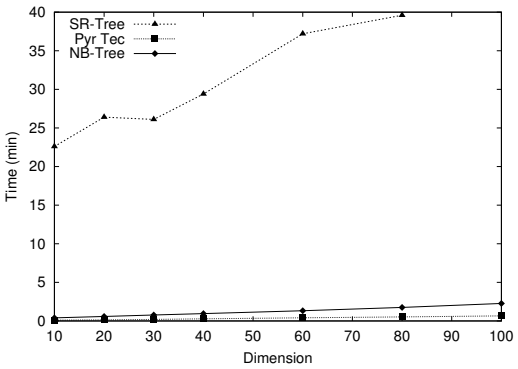

Figure 11. Creation time. Dataset size constant (100,000 points) and variable dimension.

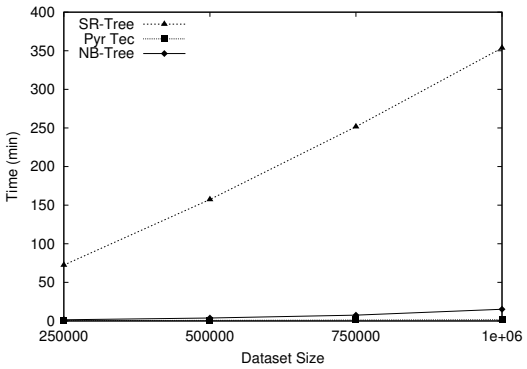

Figure 12. Creation time. Dimension constant (20) and variable database size.

plementations available did not provide the correct results and thus it is difficult to compare run-times which seem to be commensurable. Implementations of other popular approaches were available but some of them crashed on datasets of significant size, preventing comparison. We chose the SR-Tree and the Pyramid Technique as a benchmark because there are reliable and stable implementations, which provide correct results and scale up to our intended test data sizes.

In this section we describe the experimental evaluation performed to compare our NB-Tree with the SR-Tree and the Pyramid Technique. We conducted a set of experiments to analyze creation and query times as a function of dataset dimension and size. All experiments were performed on a PC Pentium II @ 233 MHz running Linux 2.4.8, with 384 MB of RAM and 15GB of disk.

We evaluated the three structures using datasets of randomly generated uniform distributed data points of fixed size (100,000) and variable dimension (10, 20, 30, 40, 60, 80 and 100). We also created datasets with fixed dimension (20) and variable size (250,000, 500,000, 750,000 and 1,000,000). Additionally, we randomly generated a set of 100 queries for each dimension, which we later used to evaluate the searching performance of each approach. We selected the number of nearest neighbors to search for to be always ten.

However, these data sets as they are generated from uniformly distributed coordinates, do not seem to be accurately representative of ”real problem” data. We plan to perform another experimental evaluation as soon as we have access to sets of real data. However, we can consider that the uniform distribution of points is a worst case than real data, because ”real data” tend to concentrate in clusters, thus requiring smaller balls to capture the nearest neighbors.

Below we present the results of our experimental evaluation for uniform data, organized by creation and KNN search times.

4.2.1. Creation

Most published work tends to ignore insertion times. This is because conventional scenarios focus on large static databases which are far more often queried upon than updated. However, there are many applications requiring frequent updating of datasets. For these, low insertion times are an important usability factor.

We have compared the creation times to these of the SR-Tree and of the Pyramid Technique. Figure 11 shows the time spent to create each structure when the dimension of the data changes. As we can see, the NB-Tree largely outperforms the SR-Tree and the Pyramid Technique outperforms both. While the Pyramid Technique takes 6 seconds to insert 100,000 points of dimension 10, the NB-Tree takes 24 seconds and the SR-Tree takes 23 minutes. If we now consider higher dimensions, such as 80, the difference increases even more with the Pyramid Technique taking 31 seconds, the NB-Tree taking 2 minutes and the SR-Tree taking 40 minutes.

In Figure 12 we can see the time for the same action, but now with dataset size changing. Although, all structures present a linear growing with the dataset size, the SR-Tree creation times grow faster than those of the other structures. While the Pyramid Technique requires no more than 2 minutes to create a tree with one million of data points, the NB-Tree requires 15 minutes and the SR-Tree takes around six hours. From this observation it is clear that the NB-Tree and the Pyramid Technique are more suited to large datasets than the SR-Tree.

In summary, and looking at Figures 11 and 12, we can say that the Pyramid Technique and the NB-Tree have very similar creation times while they largely outperform the SR-TRee.

We were not able to create the SR-Tree for the dataset of dimension 100, in our system, due to memory requirements. Thus, in the following performance charts we do not display the values for dimension 100 corresponding to the SR-Tree.

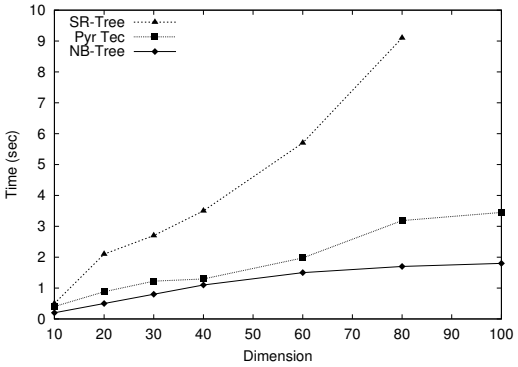

Figure 13. Searching times for KNN. Dataset size constant (100,000 points) and variable dimension.

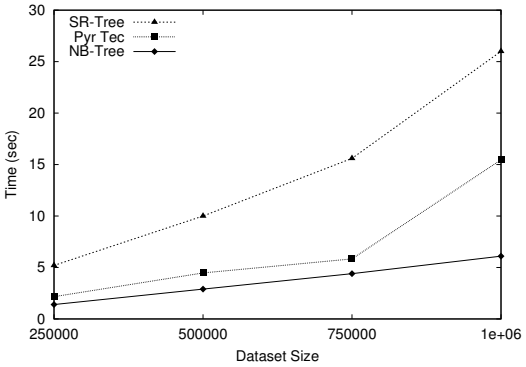

Figure 14. Searching times for KNN. Dimension constant (20) and variable database size.

4.2.2. KNN Search

Nearest-neighbor queries are useful when we want to look at the point in the dataset which most closely matches the query.

Figure 13 depicts the performance of nearest neighbor searches when the dimension increases. We can see that the NB-Tree outperforms the Pyramid Technique and the SR-Tree for any dimension of the dataset. Our approach computes the ten nearest neighbors in less than one second for dimensions up to 40 and less than two seconds for dimensions up to 100. Moreover, we can notice that the NB-Tree seems to present an asymptotic behavior with the dimension while the SR-tree seems to exhibit at least a quadratic growth.

As we can see in Figure 14 our approach also outperforms both the Pyramid Technique and the SR-Tree for a varying dataset size. The NB-Tree exhibits a linear growth with the dataset size, while the other structures grow faster when the dataset size increases.

In short, the NB–Tree presents a good tradeoff between creation and searching times, clearly outperforming some of the best indexing structures available, for uniformly distributed data. It can be argued that these represent a worse case than, say, (hyper)normally distributed data, in that we need to use larger balls to statistically guarantee that we capture enough points to satisfy the initial KNN query.

5. CONCLUSIONS and FUTURE WORK

We have presented a multidimensional indexing approach suitable for content–based retrieval of structured graphics and drawings. The approach is centered on recasting the general graphical matching problem as an instance of graph matching. To this end we index drawings using a *topology graph* which describes adjacency and containment relations for drawing blocks. We then transform these graphs to descriptor vectors in a way similar to hashing to obviate the need to perform costly graph-isomorphism computations over large databases, using a stable method. Finally a k–NN search over large databases provides the means to efficiently retrieve sub-drawings that match a given query in terms of its topology. To optimize this stage we have developed a fast yet simple method to index large databases which is flexible and scales better than other well–known methods. Through a suitable filtering procedure we feel confident that we will be able to sieve out a few meaningful samples from large structured graphical document databases using graphical information alone. Although work remains to be done at the initial stages, of the query and classification pipeline, we are confident on the ability to extend this approach to other classes of graphical data, such as surfaces, other than structured vector drawings.

ACKNOWLEDGMENTS

This work was funded in part by the Portuguese Foundation for Science and Technology and the European Commission.

References

[1] D. V. Bakergem. Image Collections in The Design Studio. In *The Electronic Design Studio: Architectural Knowledge and Media in the Computer Age*, pages 261–272. MIT Press, 1990.

[2] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In Hector Garcia-Molina and H. V. Jagadish, editors, *Proc. of the Int. Conference on Management of Data (SIGMOD'90)*, pages 322–331. ACM Press, 1990.

[3] Stefan Berchtold, Christian Böhm, and Hans-Peter Kriegel. The Pyramid-Technique: Towards Breaking the Curse of Dimensionality. In *Proceedings of the International Conference on Management of Data (SIGMOD'98)*. ACM Press, 1998.

[4] Stefan Berchtold, Christian Bohm, H. V. Jagadish, Hans-Peter Kriegel, and Jorg Sander. Independent quantization: An index compression technique for high-dimensional data spaces. In *Proceedings of the*

16th International Conference on Data Engineering (ICDE'00), pages 577–588, San Diego, USA, 2000.

[5] Stefan Berchtold, Daniel Keim, Hans-Peter Kriegel, and Thomas Seidl. Indexing the solution space: A new technique for nearest neighbor search in high-dimensional space. *IEEE Transactions on Knowledge and Data Engineering*, 12(1):45–57, 2000.

[6] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96)*, Mumbai(Bombay), India, 1996.

[7] Stefan Berchtold and Hans-Peter Kriegel. S3: Similarity in CAD Database Systems. In *Proc. of the Int. Conference on Management of Data (SIGMOD'97)*, 1997.

[8] Kaushik Chakrabarti and Sharad Mehrotra. The Hybrid Tree: An index structure for high dimensional feature spaces. In *Proceedings of the 15th International Conference on Data Engineering (ICDE'99)*, pages 440–447, 1999.

[9] S. K. Chang, B. Perry, and A. Rosenfeld. *Content-Based Multimedia Information Access*. Kluwer Press, 1999.

[10] M. Clayton and H. Wiesenthal. Enhancing the Sketchbook. In *Proc. of the Association for Computer Aided Design in Architecture (ACADIA'91)*, Los Angeles, CA, 1991.

[11] Dragos Cvetkovic, Peter Rowlinson, and Slobodan Simic. *Eigenspaces of Graphs*. Cambridge University Press, United Kingdom, 1997.

[12] Ellen Y. Do. What's in a Diagram that a Computer Should Understand? In *Proc. of The Sixth Int. Conf. on Computer Aided Architectural Design Futures (CAADF'95)*, pages 103–114. The Global Design Studio, 1995.

[13] Ellen Y. Do. *The right tool at the right time*. PhD thesis, Georgia Institute of Technology, September 1998.

[14] Manuel J. Fonseca and Joaquim A. Jorge. Experimental Evaluation of an on-line Scribble Recognizer. *Pattern Recognition Letters*, 22(12):1311–1319, 2001.

[15] Mark Gross and Ellen Do. Demonstrating the Electronic Cocktail Napkin: a paper-like interface for early design. In *Proc. of the Conf. on Human Factors in Computing Systems (CHI'96)*, pages 5–6, 1996.

[16] Mark D. Gross. Indexing Visual Databases of Designs with Diagrams. In *Visual Databases in Architecture*, 1995.

[17] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In Beatrice Yormark, editor, *Proc. of the Int. Conference on Management of Data (SIGMOD'84)*, pages 47–57. ACM Press, 1984.

[18] A. Henrich. The LSDh-tree: An access structure for feature vectors. In *Proceedings of the 14th International Conference on Data Engineering (ICDE'98)*, pages 362–369, 1998.

[19] A. Henrich, H.-W. Six, and P. Widmayer. The LSD tree: Spatial access to multidimensional point and non-point objects. In *Proceedings of the 15th International Conference on Very Large Data Bases (VLDB'89)*, pages 45–53, 1989.

[20] N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proceedings of the International Conference on Management of Data (SIGMOD'97)*, pages 369–380. ACM Press, 1997.

[21] B. C. Ooi, R. Sacks-Davis, and K. J. McDonell. Spatial K-D-tree: An indexing mechanism for spatial databases. In *Proceedings of IEEE COMPSAC Conf.*, 1987.

[22] Jong Park and Bong Um. A New Approach to Similarity Retrieval of 2D Graphic Objects Based on Dominant Shapes. *Pattern Recognition Letters*, 20:591–616, 1999.

[23] Yong Rui, Thomas S. Huang, and Shih-Fu Chang. Image Retrieval: Past, Present, and Future. *Journal of Visual Communication and Image Representation*, 1998.

[24] Yasushi Sakurai, Masatoshi Yoshikawa, Shunsuke Uemura, and Haruhiko Kojima. The A-tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation. In *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB'00)*, pages 516–526, Cairo, Egypt, 2000.

[25] A. Shokoufandeh, S. Dickson, K. Siddiqi, and S. Zucker. Indexing Using a Spectral Encoding of Topological Structure. *IEEE CVPR*, 2, 1999.

[26] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB'98)*, pages 194–205, New York, USA, 1998.

[27] David A. White and Ramesh Jain. Similarity Indexing: Algorithms and Performance. In *Proceedings SPIE Storage and Retrieval for Image and Video Databases*, 1996.

[28] David A. White and Ramesh Jain. Similarity indexing with the SS-tree. In *Proceedings of the 12th IEEE International Conference on Data Engineering*, pages 516–523, 1996.