2011 International Conference on Document Analysis and Recognition

A Graph Lattice Approach to Maintaining Dense Collections of Subgraphs as Image Features

Eric Saund
Palo Alto Research Center; 3333 Coyote Hill Rd.; Palo Alto, USA
Email: saund@parc.com

***Abstract*—Document classification and indexing methods depend on having informative image features. This paper shows how large families of complex features can be built out of simpler ones through construction of a graph lattice—a hierarchy of related subgraphs linked in a lattice. A graph lattice enables efficiency gains that make it possible to effectively employ bag-of-words methods for document classification using high-dimensional feature vectors. Each feature is itself a subgraph, and a feature vector is a count of occurrences of subgraphs in the image. The graph lattice enables methods for adaptively growing a feature space of subgraphs tailored to observed document genres. We demonstrate the approach through classification of forms containing rectilinear line art.**

***Keywords*-Graph Lattice; document classification**

I. Introduction

Recent years have seen a surge in bag-of-words approaches to visual categorization and image indexing in computer vision [4], [13]. Objects and scenes are modeled as large vectors of relatively simple feature measurements which are employed by hashing and vector comparison methods.

An important issue is the information captured by a feature. For document images containing dense text, the localized arrangement of keypoints centered at word blobs has proven effective at generating geometric "fingerprint" features [8] based on triangle area ratios.

A different way of encoding spatial configuration is through graphs. Objects and scenes are modeled as parts (nodes) and relations (links). An observed image generates a graph of parts and their relations to other parts, and recognition is performed by subgraph matching. Subgraph matching poses two difficulties. First, it is known to be exponentially expensive. This problem is to some extent alleviated by use of attributed graphs, that is, graphs whose nodes contain properties that constrain possible matches. Nonetheless, subgraph matching has been limited to relatively small subgraphs, due to the second difficulty. Namely, noise and variability cause observed graphs to deviate from ideal models. This demands use of inexact graph matching techniques, which drastically increases matching cost and largely removes the advantages of attributed graph matching because possible matches of differently-labeled nodes must now be explored. Conte et al. provide a thorough review of these issues [2].

This paper proposes to combine bag-of-words and graph matching approaches in a way that leverages the advantages of each. Features correspond to subgraphs, each of which encodes a limited amount of information about spatial configuration in a local neighborhood. By supporting large numbers of these feature subgraphs, we stand a good chance of capturing image structure through exact graph matching. This is analogous to the method of shingling in text retrieval [7], but challenges lie in extending to two dimensions. Other work in bringing text retrieval methods to image retrieval observes that a large vocabulary of structural features does provide for robustness through the use of statistical methods for matching [6].

This idea is applicable to classification and indexing of documents containing consistent subgraphs. Our focal problem is forms documents which have sparse and inconsistent word blobs, but which usually contain a network of rectilinear rule lines serving as region separators, data field locators, and organizational devices to group together similar fields. Rule lines intersect each other in well-defined ways that form junction and free-end terminator graph nodes. These are natural candidates for the nodes of an attributed graph representation (a *data graph*) while the rule line segments linking junctions become the links.

Junction-link relations define the topology and directional geometry of an observed line-art image. Dimensional geometry—the lengths of rule lines between junctions—is important too. But, these properties are continuous-valued in our target domain and therefore complicate mapping and matching in discrete graphs. In order to focus on the development of the graph lattice approach, this study is confined to rectilinear directional attributes.

Doctype classification is a principal step in virtually all automatic document processing systems. Other features used for forms classification include text content, connected component attributes, Haar filters, zonal densities, texture filters, and line-art locations and junctions. Methods for forms classification include string matching, template matching, nearest-neighbor, decision trees, generative feature density models, and neural networks [9], [10], [11], [12]. Chen and Blostein provide an extensive survey [1]. By and large, most prior work emphasizes sophisticated learning algorithms applied to simple, easy-to-compute features.

1520-5363/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDAR.2011.216

IEEE computer society

In the following, Section II describes the use of subgraphs as image features. We demonstrate the benefits of adding larger subgraph features for forms clustering and classification. Section III then shows how processing may be made efficient by use of a graph lattice architecture. Efficiency is important to maintaining large vocabularies of graph-lattice features. Organized as a lattice with appropriate bookkeeping, each subgraph requires only a small incremental amount of work to compute its mappings onto a data graph.

II. Data Graphs and Subgraph Features for Rectilinear Line Art Analysis

Effective handling of image noise and variability is a key issue in document recognition. The data corpus used in this paper is the NIST tax forms database [5], which consists of 11185 images, of size 2560 x 3300 pixels, representative of scanned hand-filled and typed United States Tax Forms. Figure 1a shows that the line art in this data is often noisy and distorted. Using standard as well as specialized image processing approaches, we can extract rule lines and their intersections at junctions with approximately 95% reliability. In a typical form of 250 terminations and junctions, we will therefore see about 10-15 errors. Typical errors are substitution of T-junctions and X-junctions (actually + junctions in upright orientation), and vice versa; introduction of spurious terminations where line-art is broken by image noise; and failure to identify L-junctions.

Rectilinear line-art gives rise to 13 junction/termination types as shown in Figure 2. For an observed line-art image, after extraction of rule lines it is straightforward to build a *data graph* consisting of attributed nodes and links among them, as shown in Figure 1b. A node's attribute is the index of its junction/termination type.

The NIST tax form data comprises 20 categories and adequately represents many commercial-scale forms classification problems. As a baseline step, we may ask whether the counts of primitive junction types is sufficiently informative to distinguish NIST tax forms. Figure 3 indicates not. This is a histogram of pairwise similarities between 1000 NIST images. We observe two primary modes. The high-similarity mode derives from forms of the same type, while the low-similarity mode derives from forms of different type. Note that the modes are not clearly separated, which means that there is an area of confusion about whether two images should be assigned to the same category. This confusion region is due to the noise and variability in these images, given that some different forms happen to be designed to contain approximately the same number of terminations, L-junctions, T-Junctions, and crossings (X-junctions). There are many possible ways of assessing the similarities/differences between feature vectors—in this case between 13-dimensional vectors of junction type counts. The similarity measure used here, called CMD (Common-Minus-Difference), is described below. Euclidean distance and cosine distance both produce less distinct histogram modes.

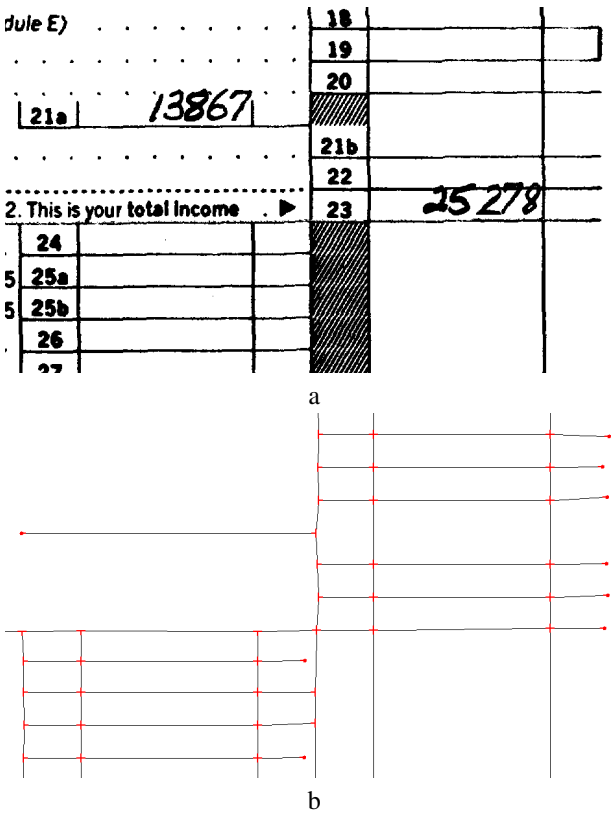


Figure 1. a. Section of a NIST tax form image. b. The data graph (junctions and links) extracted from it. Note errors of two missing T-junctions.

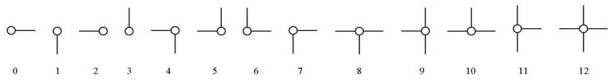


Figure 2. The 13 junction types in rectilinear line art.

If features and similarity measures could be found that produce two well-separated modes in the histogram, this would enable a very straightforward greedy algorithm for clustering and classifying images. If the same-category and different-category modes were well separated, threshold values could be determined from the histogram automatically.

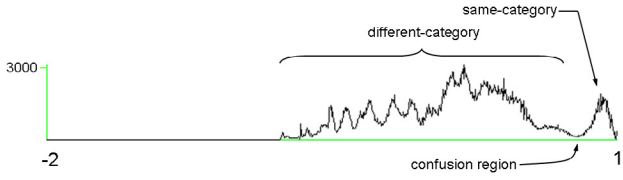


Figure 3. Histogram of pairwise CMD similarity scores (see text) for 1000 NIST documents based on primitive junction type counts.

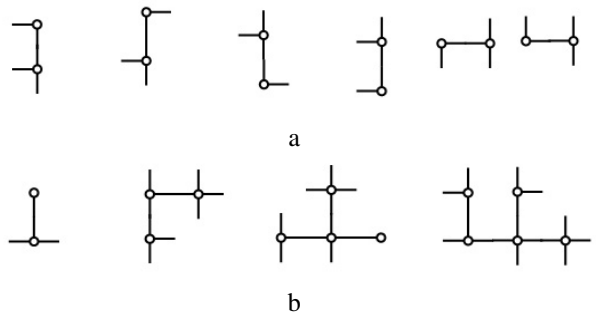


Figure 4. a. Six of the 98 size 2 groupings of primitive junctions. b. A few larger subgraphs found on tax forms.

More complex features are obtained by considering not simply counts of primitive junctions, but counts of pairs, triples, and larger collections of junctions. Given the constraints of how junctions may link to one another, there are 98 ways that the thirteen primitive junction types may be grouped pairwise, as suggested in Figure 4a. The number of possible junction-triple subgraphs is much larger, and clearly the number grows rapidly with subgraph size. Higher order subgraphs provide increasing constraint on how the primitive junctions in the observed data graph are linked, up to the highest possible subgraph which is the data graph itself. Feature vectors incorporating higher order subgraphs can therefore be more informative than primitive junction counts.

The space of possible higher-order subgraphs is only sparsely populated by subgraphs actually observed in data. In adding subgraph features, it is useful to utilize only these, and omit non-observed subgraphs from the feature vector. Figure 4b presents a few of the many subgraph features of different sizes observed in NIST tax form data. The graph lattice supports strategies for selecting relatively large—therefore potentially highly discriminative—subgraphs based on observed data graphs.

We describe here the results of extending the feature vector to count subgraph matches for subgraphs of sizes two, three and four. The first step in the process is to build the feature representation. We choose a subset of observed data graphs which we expect to include representatives of most subgraphs that the full data set will contain. Roughly, this subset should contain one or a few examples of each image category. For these, we extract and catalog every subgraph of the target size. By observing a plateau in growth of the subgraph catalog with sample documents, we determined that 50 randomly selected NIST image samples is sufficient to build a useful vocabulary of subgraphs for clustering and classification.

Given a feature measurement vector defined by subgraphs of degree 1 to D, where D is maximum subgraph size, we measure counts of matches to an observed data graph. Each match produces a mapping of subgraph nodes to data graph nodes. One such mapping is illustrated in Figure 7. Efficient means for performing the requisite subgraph matching is described in Section III.

We have found that straight subgraph match counts do not constitute a good feature vector for comparing data graphs. The reason has to do with overweighting of larger subgraphs. We therefore perform a re-weighting of subgraph match counts according to overlaps in the junctions they map to, and we perform vector comparisons on the resulting *Junction Normalized Mapping Count* vector.

A second consideration is the similarity measure used to compare junction-normalized feature vector (subgraph match) counts. Obvious choices are Euclidean distance and cosine distance. We have found that neither of these works as well as another similarity measure, CMD (Common-Minus-Difference):

$$s(v_1, v_2) = \frac{\sum_i (\min(v_{1,i}, v_{2,i}) - |v_{1,i} - v_{2,i}|)}{\max(|G_1|, |G_2|) * N_d} \quad (1)$$

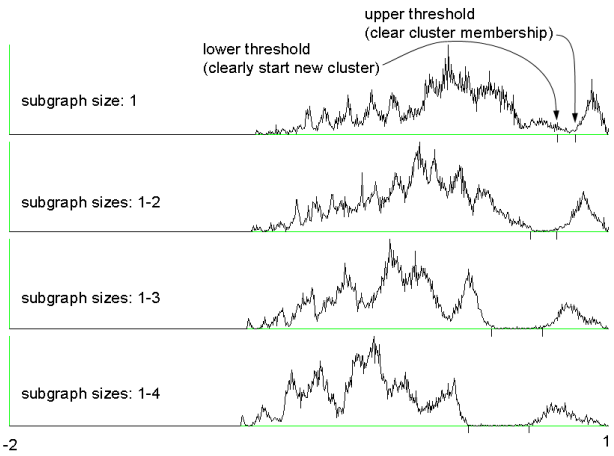
where G_k is the size (number of junctions) of data graph k and N_d is the number of subgraph sizes considered in the junction-normalized feature vector.

Due to the normalization term based on the sizes of the data graphs being compared, the range of the CMD similarity measure is -2 (minimum, least similarity) to 1 (maximum, best similarity).

Under the junction-normalized feature count and the CMD similarity score, we find that higher order subgraph features do indeed lead to improved discrimination. Figure 5a presents pairwise similarity histograms for feature vectors up to subgraph size 4. For the NIST data, beyond subgraph feature size 2, different image categories are clearly separated. Under this representation using feature vectors comprising subgraphs of sizes 1-3 or 1-4, a simple greedy clustering algorithm correctly sorts all 11,185 NIST images into their respective 20 categories, with one category split into two.

Clustering results are presented in Figure 5b. Clustering uses two automatically-computed thresholds. If an observed feature similarity to a cluster's centroid exceeds the upper threshold, the document is added to that cluster. If feature similarity falls below a lower threshold for all existing clusters, a new cluster is started. Otherwise, the image is put aside into an "unknown" category until all images have been considered. This process is iterated with the unknown images until no more can be assigned to a cluster based on the upper threshold. Finally, each unknown image is assigned to the best-matching cluster.

Quality of clustering is scored as the edit-distance to the groundtruth correct assignment of images to categories. One edit operation is tallied for each incorrectly classified document, and one edit operation is tallied for merging any two clusters representing the same groundtruth category.



a

subgraph size(s)	feature vector size	errors	spurious clusters	cluster quality (edit distance to G.T.)
1	13	6	18	24
1-2	87	3	5	8
1-3	534	0	1	1
1-4	2953	0	1	1

b

Figure 5. a. Histograms of pairwise CMD distances for 200 NIST documents based on feature vectors of subgraph match counts, ranging from size 1 subgraphs only through subgraphs of sizes 1 through 4. Small vertical lines show automatically-computed thresholds (clearly within-cluster and start-new-cluster) for greedy clustering. b. Clustering results for 11,185 combined NIST SpecialDatabase6 and SpecialDatabase2 images.

Forms clustering and classification is almost 100% correct for subgraphs of size 3 and larger; the only error is an extra cluster duplicating one of the groundtruth categories. To our knowledge is the best result reported to date on the benchmark NIST forms data set, improving even on supervised classifiers which by definition are supplied with a known number of categories [10], [11], [12].

III. Graph Lattice Data Structure

A. Nodes and Struts

A program architecture for computing feature vectors representing large subgraphs should support two main purposes: (1) efficient computation of subgraph (feature element) matches to observed data graphs, and (2) effective construction of more complex features (larger subgraphs) from smaller ones. The graph lattice fulfills these needs.

The basic element of a graph lattice is the graph lattice node, as distinguished from a node of a data graph. A graph lattice node represents a subgraph, plus its mappings onto data graphs and its relations to larger and smaller subgraphs in the lattice. As a matter of terminology, we call smaller subgraphs *parent* nodes, and larger subgraphs generated from them by adding junctions, *child* nodes. In general two subgraphs of arbitrary size could be conjoined to create a larger subgraph, perhaps of size the sum of the sizes of parent nodes, or perhaps smaller if overlap is allowed. Thus far we have found such a general mechanism unnecessary. Instead, we only support single-level links in the graph lattice representing the accretion of a single junction (data graph node) at a time. In other words, child graph lattice nodes are always one degree larger than their parents.

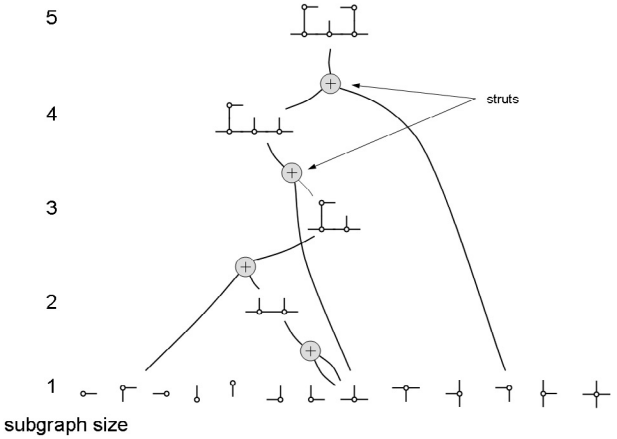


Figure 6. A graph lattice consists of layers of subgraphs related by addition or deletion of primitives (junctions, in the case of rectilinear line art). These relations are maintained through *struts* maintaining the mappings and linkages formed by incrementally added primitives.

The relations among graph lattice nodes are maintained by *struts*. See Figure 6. The purpose of a strut is twofold. First, it maintains the junction index mappings between a parent and child node. In general any graph lattice node will index its component junctions in arbitrary order, and a strut keeps these organized between parent and child graph lattice nodes. Second, a strut indicates the primitive type, placement on the parent, and links for the junction that constructs the child from the parent.

B. Sweep-Upward Matching

Matching of the complete set of nodes in a graph lattice to an observed data graph is efficient because each graph lattice node (subgraph) can be incrementally mapped based on mappings of its parents. See Figure 7. Processing proceeds bottom up, layer-by-layer, in a *sweep-upward* algorithm. At layer subgraph-size $= d$, for each node ${}^{d}n_i$ we compute subgraph mappings as follows: choose arbitrarily one parent node at layer $d-1$. The strut to this parent defines the possible mappings of ${}^{d}n_i$ onto the data graph based on mappings of the parent. The strut also defines the single extra data graph junction that node ${}^{d}n_i$ contributes. We essentially follow the approach used in the VF2 subgraph matching algorithm of Cordella et al. [3]; it is easy to test the data graph to see if it contains such a junction, properly linked, to support a complete mapping of ${}^{d}n_i$. In our Java implementation, using the sweep-up algorithm, the compu-

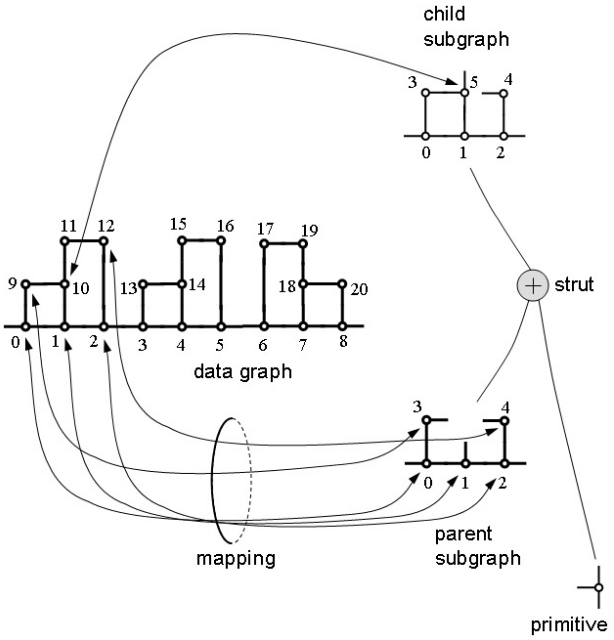


Figure 7. In the sweep-upward algorithm, subgraph mappings are based on the previously-computed subgraph mappings at the parent level. Struts maintain bookkeeping data structures enabling only a quick check of whether the incremental junction is present in the data graph (upper mapping arrow).

tation of the junction-normalized feature vector for a graph lattice of 2,953 graph lattice nodes (subgraphs) averages 43 milliseconds per data graph over a set of typical NIST tax form documents. Using the sweep-up algorithm, subgraph matching takes approximately 1 microsecond per subgraph for subgraphs ranging up to size 4; this is a speedup of better than 10x over brute-force attributed subgraph matching.

C. Growing a Graph Lattice

Our approach trades storage necessary to hold the graph lattice for faster matching time via the sweep upward algorithm. The number of possible subgraphs grows exponentially with subgraph size. This explosion is managed through the judicious selection of a relatively smaller number of subgraphs represented explicitly as graph lattice nodes, among the space of all possible subgraphs at any degree. Any data domain will reflect characteristic patterns as more or less commonly occurring subgraphs, while many other possible subgraphs will not be observed at all. Thus, the selection of graph lattice nodes constitutes knowledge about the data domain in the form of the subgraph features we choose to maintain. As a general framework, we propose an approach to constructing graph lattices based on incremental evaluation and acceptance of candidate graph lattice nodes which accrete to an existing simpler graph lattice of accepted nodes, based on observed examples. See Figure 8. The algorithm is to iterate the following steps.

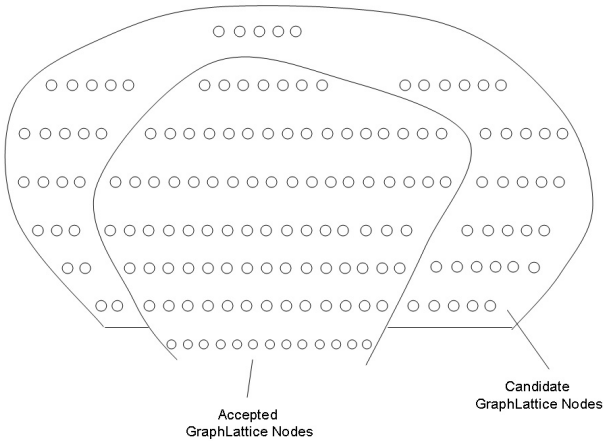


Figure 8. A graph lattice can be constructed by promoting Candidate nodes to Accepted status. Accepted nodes are eligible to spawn more Candidates.

Step 1: Generate Candidate graph lattice nodes from Accepted graph lattice nodes and observed data.

Every mapping of a Level d subgraph onto an observed data graph can serve as a seed for spawning new Level $d+1$ graph lattice nodes which are supergraphs of the graph represented by that graph lattice node. Each primitive linked to the perimeter of the subgraph can itself give rise to a subgraph one data graph node larger in size, and therefore one level higher in a graph lattice. Step 1 of the graph lattice construction algorithm is to examine mappings of Accepted graph lattice nodes onto observed data graphs and spawn new Candidate graph lattice nodes based on actual supergraphs encountered in the data. This step may involve mapping the existing graph lattice to previously seen data graphs or to new, previously unobserved data graphs. This process involves some bookkeeping to follow subgraph mappings, detect duplicate candidates, and build struts to all of a Candidates' parents at at Level N.

Step 2: Select Candidate graph lattice to promote to Accepted graph lattice nodes

The second step of the graph lattice construction algorithm is to select certain Candidate graph lattice nodes for promotion to Accepted status. In the forms image clustering work reported above, we employ a trivial Candidate selection strategy of accepting all candidate nodes up to a certain level.

Depending on the acceptance strategy used in Step 2., various stopping criteria may become meaningful, including: (1) The graph lattice contains a threshold number of Accepted graph lattice nodes at a given level; (2) The graph lattice contains a threshold number of Accepted graph lattice nodes in total; (3) The list of Candidate graph lattice nodes is exhausted (the forms clustering/classification application

uses this); (4) Quality measures for Candidate nodes falls below a threshold.

For the NIST data set, the time required to build a graph lattice of degree 4 from the data graphs of 50 randomly sampled images is three seconds. To build a graph lattice of degree 4 from 1000 NIST images takes 50 seconds, resulting in 4,560 graph lattice nodes.

IV. Conclusion

This paper has demonstrated the mechanics of building and using graph lattices where each node of the lattice represents a subgraph in a data graph derived from an image. This has been applied to the classification of forms documents containing rectilinear line art, where we can now efficiently compute high-dimensional vectors of informative features corresponding to counts of exact matches of numerous subgraphs. The approach has been shown to improve upon state-of-the art results for classification and clustering on the benchmark NIST data set. We have verified the approach in a second data set of commercial importance, namely clustering and classification of AMA Dental forms which occur in over 100 subtypes.

There remains much to explore about the basic idea of constructing and maintaining lattices of related subgraphs.

Beyond the trivial method of building a graph lattice by accepting all Candidate nodes up to some point, many other strategies are possible. For example, one idea we have explored briefly is selection of highly indicative candidates using an entropy-based measure of node type diversity. Subgraphs dominated by one type of primitive tend to reflect repeated patterns like meshes, while subgraphs containing many different primitive types are more often unique to a particular forms image class.

In a graph lattice it is easy to manage chains of subgraph parentage up to large subgraphs which are unique or highly indicative of a layout pattern. These can be used not only as elements of high-dimensional feature vectors, but also as terms in inverted indices that identify known classes through voting.

Another important issue is extension to continuous-valued attributes such as dimensional attributes of the arrangement of primitive image features. This may entail some sort of quantization of subgraphs and the introduction of quantitative similarity links among them.

Acknowledgment

Many thanks are due to the members of the PARC Perceptual Document Analysis group for numerous ideas and suggestions. Thank you to Fang Liu for discussions and for building a preliminary implementation of the graph lattice machinery.

References

[1] N. Chen and D. Blostein. A survey of document image classification: problem statement, classifier architecture and performance evaluation. *IJDAR*, 10, 2007.

[2] D. Conte, P. Foggia, C. Sansone, and M. Vento. How and why pattern recogntion and computer vision applications use graphs. In A. Kandel, H. Bunke, and M. Last, editors, *Applied Graph Theory in Computer Vision and Pattern Recognition*. Springer, 2007.

[3] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE T.PAMI*, 26(10), October 2004.

[4] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. *ECCV Workshop on Statistical Learning in Computer Vision*, 2004.

[5] D. Dimmick, M. Garris, and C. Wilson. Structured forms database. Technical Report Technical Report Special Database 2, SFRS, National Institutte of Standards and Technology, December 2001.

[6] S. Feng and R. Manmatha. A discrete direct retrieval model for image and video retrieval. In *7th International Conference on Image and Video Retrieval*, July 2008.

[7] U. Manber. Finding similar files in a large file system. In *Proceedings of the USENIX Winter 1994 Technical Conference*, 1994.

[8] T. Nakai, K. Kise, , and M. Iwamura. Use of affine invariants in locally likely arrangement hashing for camera-based document image retrieval. In *7th International Workshop, DAS 2006*, 2006.

[9] D. Neogi, S. K. Ladd, and V. Govindaraju. Systems and methods for classifying electronic documents by extracting and recognizing text and image features indicative of document categories. United States Patent Application 20090116757, May 2009.

[10] K. U. Reddy and V. Govindaraju. Form classification. In Yanikoglu and Berkner, editors, *Document Recognition and Retrieval XV*, volume 6815, 2008.

[11] P. Sarkar. Image classifcation: Classifying distributions of visual features. In *International Conference on Pattern Recognition*, August 2006.

[12] C. Shin, D. Doermann, and A. Rosenfeld. Classification of document pages using structure-based features. *IJDAR*, 3, 2001.

[13] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proc. 9th International Conference on Computer Vision (ICCV 2003)*, 2003.