IJDAR (2001) 4: 69–83

International Journal on Document Analysis and Recognition

Automatic document classification and indexing in high-volume applications

E. Appiani[1], F. Cesarini[2], A.M. Colla[1], M. Diligenti[3], M. Gori[3], S. Marinai[2], G. Soda[2]

[1] Elsag spa TRI Department, Via G. Puccini, 2, 16154 Genova, Italy; e-mail: {enrico.appiani,annamaria.colla}@elsag.it
[2] DSI, Università di Firenze, Via S. Marta, 3, 50139 Firenze, Italy; e-mail: {cesarini,simone,giovanni}@dsi.unifi.it
[3] DII, Università di Siena, Via Roma, 56, 53100 Siena, Italy; e-mail: {diligmic,marco}@ultra3.dii.unisi.it

Received March 30, 2000 / Revised June 26, 2001

Abstract. In this paper a system for analysis and automatic indexing of imaged documents for high-volume applications is described. This system, named STRETCH (STorage and RETrieval by Content of imaged documents), is based on an *Archiving and Retrieval Engine*, which overcomes the bottleneck of document profiling bypassing some limitations of existing pre-defined indexing schemes. The engine exploits a structured document representation and can activate appropriate methods to characterise and automatically index heterogeneous documents with variable layout. The originality of STRETCH lies principally in the possibility for unskilled users to define the indexes relevant to the document domains of their interest by simply presenting visual examples and applying reliable automatic information extraction methods (document classification, flexible reading strategies) to index the documents automatically, thus creating archives as desired. STRETCH offers ease of use and application programming and the ability to dynamically adapt to new types of documents. The system has been tested in two applications in particular, one concerning passive invoices and the other bank documents. In these applications, several classes of documents are involved. The indexing strategy first automatically classifies the document, thus avoiding pre-sorting, then locates and reads the information pertaining to the specific document class. Experimental results are encouraging overall; in particular, document classification results fulfill the requirements of high-volume applications . Integration into production lines is under execution.

Keywords: Document classification – Decision tree – MXY tree – Reading strategy

1 Introduction

In this paper we describe a method for the automatic classification and indexing by content of imaged documents, implemented in an archiving and retrieval system developed within the Esprit project STRETCH (STorage and RETrieval by Content of imaged documents)[1]. Several application fields (for instance, banking and insurance, corporate databases, cultural institutions, public administration) require the storage and indexing of huge volumes of documents in digital archives. Moreover, in several applications, specific pieces of information must be extracted from the documents for data entry required by workflow procedures. The latter operation, traditionally manual and labour-intensive, is currently approached by introducing OCR software in many high-volume applications.

It is usually rather easy to tune commercial OCR software to read forms, i.e., documents which adhere to one or a few specific models [22] where the information can be located precisely. However, in several practical applications, it is necessary to process documents with many different layouts, where information to be extracted is variable or located in different positions [5]. In order to read the relevant information for automatic indexing it is necessary to "understand" each document layout to classify the document and select the appropriate reading strategy. In this case document profiling is much more complex and tuning even the most effective commercial OCR/ICR is quite expensive. Manual presorting of documents into meaningful classes is the usual compromise adopted to obtain the required accuracy.

Some applications do not require all document types to be indexed; in this case document classification accuracy is even more important to avoid processing a conspicuous fraction of the documents. Therefore, the major challenge of STRETCH in several applications is to achieve accurate document classification, then to apply an appropriate and accurate reading strategy to the fields to be employed for indexing. Only rejected documents need to be manually checked, greatly improving the productivity of the document workflow.

The system designed and implemented in STRETCH is based on an Archiving and Retrieval Engine, which exploits a structured Object-Oriented document repre-

Correspondence to: S. Marinai

[1] For details, see the project homepage at http://www.aetnet.it/stretch/

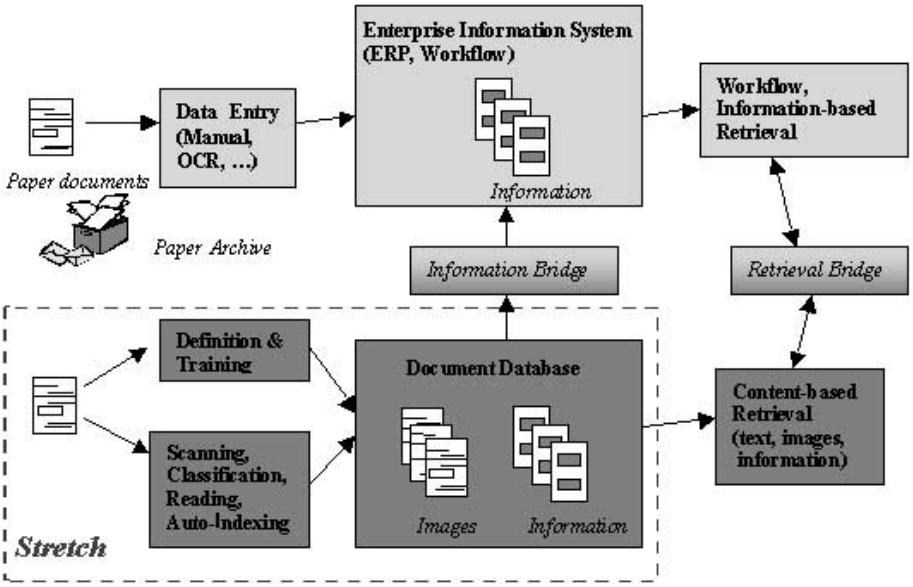


Fig. 1. STRETCH embedded in the corporate workflow

sentation. The objects are enriched with appropriate methods to locate and automatically extract information to index heterogeneous documents with variable layout. This representation is employed in particular by the Document Processing Server which processes the document images to classify and index each document. The core technology employed consists of layout analysis and classification, automatic field location, logo and tag identification, Intelligent Character Recognition (ICR). After layout analysis, the document is classified according to the user's specification and information contents are extracted from the relevant fields.

STRETCH can be integrated into current production lines as a parallel line, without interfering with the traditional workflow or other automated solutions (see Fig. 1 for a general example concerning corporate workflow). Further integrations of the existing workflow management system with document processing modules can increase the overall performance, as explored in the Virtual Office project [15] with the integration of the OfficeMAID System [3].

STRETCH has been tested in various domains that are conspicuous examples of the application fields addressed, namely, an account payable archive, consisting of invoices and related documents [1]; a document archive for the Public Administration; and bank documents [2].

In this paper we describe in particular the document processing core of our system, in terms of both the methods implemented to reach appropriate accuracy and the architectural solution adopted to obtain an efficiency compatible with high-volume application requirements. The paper is organized as follows. In Sect. 2 we briefly summarize the system architecture, introducing the Document Processing Server and presenting some implementation details adopted to increase efficiency. In Sect. 3 the methods employed in the Document Processing Server to perform document classification and document indexing are described in detail. Section 4 presents some experimental results concerning two typical real-world application scenarios: passive invoice workflow and bank document archives. Finally in Sect. 5 some conclusions are drawn.

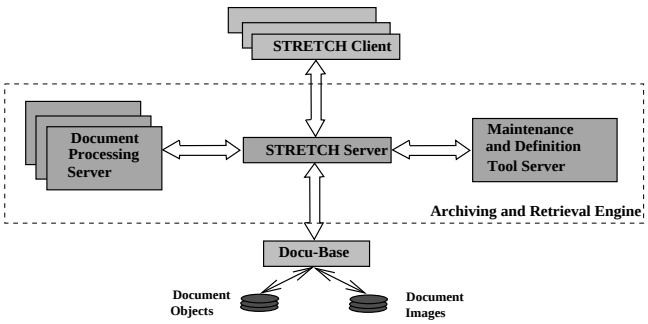


Fig. 2. Main *STRETCH* components and mutual relations

2 STRETCH architecture

From an architectural point of view, the system relies on a networked client/server/database solution. The main achievement consists in the development of an *Archiving and Retrieval Engine (ARE)*, capable of activating appropriate methods to index, archive, and retrieve imaged documents.

Components of both the "Client" and "Server" implement functionalities to achieve system modularity, distributed configuration, and scalability. The "Database" layer, called *Docu-Base* in STRETCH, includes the components to store persistent data. The main components of the STRETCH architecture are shown in Fig. 2. A standard Corba middleware [19] links all the components.

The *STRETCH Client* is the user interface; it is implemented in Java and may have, of course, multiple distributed instances. A main screen, the *GUI Manager*,

provides access to specific screens: *Archive Client*, to select groups of images and submit them as indexing-archiving batches, whose performance can be monitored; *Retrieval Client*, to perform relational queries on any information field and to analyze their results; and *Document Visual Objects*, to browse and edit corresponding documents in the Server, from either retrieved or currently processed documents (an example is presented in Fig. 9).

The Server layer is made of three main components (Fig. 2). The *STRETCH Server* presents the ARE interface to *STRETCH Client* instances, dispatches their requests to the other server components and to the *Docu-Base*, also enforcing the scheduling strategies for parallel processing. The *Document Processing Server (DPS)* carries out the document processing for automatic indexing employing layout classification and flexible reading strategies (Sect. 3). The last server component, *Maintenance and Definition Tool Server*, takes care of maintaining the system configuration and domain-related reading strategies. In addition, users can define new document examples and update the selected domain by a learning process, involving both layout classification and reading strategies.

The *Docu-Base*, linked to a relational database engine, archives and retrieves information extracted from documents with related images. Moving compressed image files reduces the communication load.

The STRETCH architecture employs a three-tiered layering of current software technology,providing the following advantages:
Modularity: a *STRETCH Server* can accept client sessions with only a subset of instantiated components.
Openness: external components may use their own client to interface with the *STRETCH Server*; similarly the relational database and image archive can be replaced.
Scalability: document processing, the most compute-intensive function, can achieve increasing performance and robustness through *DPS* replication on parallel or distributed machines (as detailed below).

The Server includes both Java and native components, the latter used for image processing and OCR functions. Despite Corba's possibility to link components defined with different languages, Java has been employed to define all the front-end Server interfaces, while the pieces of native code are loaded and executed as internal resources for document processing. In fact, with its dynamic class loading and portability, Java has also proven to be a good "wrapper" for native resources, and is easily changed in case of porting to different platforms. The current STRETCH Server platform is Windows NT-4.

2.1 Concurrent document processing

The scalability in document processing has been enforced by allowing DPS replication on the same server machine and/or on different ones.

On archiving, concurrent threads are started by the *STRETCH Client*; they result in corresponding server threads, one for each document to be processed. Each archiving thread in the *STRETCH Server* waits until there is at least one DPS instance available and performs a synchronized (sequential by mutual exclusion) call to its image processing method. During an archiving batch, all the DPS instances are kept busy until the batch is finished. Thanks to this simple but effective dynamic load balancing, every DPS instance receives an operation load proportional to its processing capability, thus well exploiting any processor configuration, including heterogeneous machines. At the end of image processing, the archiving thread sends the information extracted to the *Docu-Base* archive.

The execution of each DPS in a separate Java Virtual Machine, even when running on the same physical machine, allows effective resource separation and exploitation of parallel machines.

This approach has been benchmarked on a cluster of heterogeneous machines (Sect. 4.3).

3 Document processing server methods

Any document can be described with respect to physical and logical aspects. The physical structure of a document, i.e., the layout structure, is the collection of physical objects obtained by the segmentation of the document image. Similarly, the logical structure of a document is obtained by the repeated division of the document content into increasingly smaller parts (logical objects), on the basis of the human understanding of the "meaning" of the content.

Documents can be clustered with respect to their subject or use, according to the users' view: for example, journals, tax forms, business letters, invoices, cheques, and generic bank documents can be regarded as different domains. Since the documents of a domain are used for similar or related functions, they are characterized by some logical and physical similarities. Documents belonging to a given domain can be further characterized by different layout or logical structures. Thus, documents which feature physical similarities can be clustered into classes.

Document classification and indexing is performed in DPS by a cascade execution of three methods (Fig. 3):

- A top-down *segmentation* method (Sect. 3.1) aimed at extracting a hierarchical description of the input document.
- A *physical classification* of the document's layout is performed by comparing its MXY tree representation with pre-stored representations of documents of known class. In order to have an efficient comparison an appropriate decision tree (the Document Decision Tree, DDT), is taken into account (Sect. 3.2).
- After the physical classification step, it is possible to activate an *automated indexing strategy* (Sect. 3.3). The strategy aims at applying a proper reading procedure to extract the information pertaining to the document class. Since the strategy is based on the location of appropriate tags (keywords) and logos in class-dependent areas, the application of the strategy allows us to obtain an implicit *logical classification*

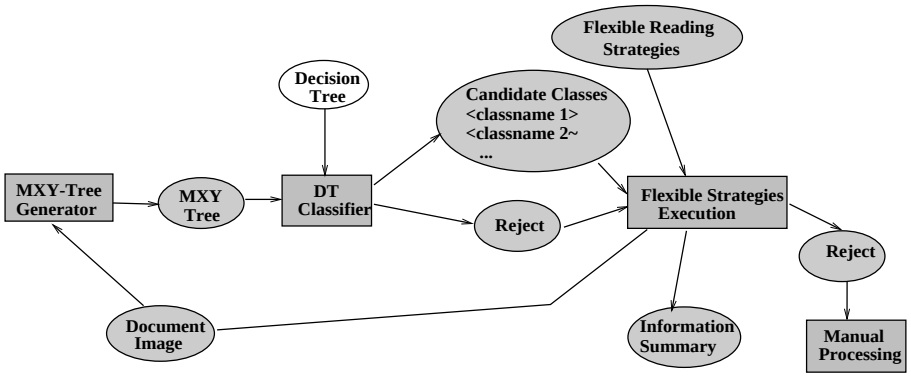


Fig. 3. Document processing strategy

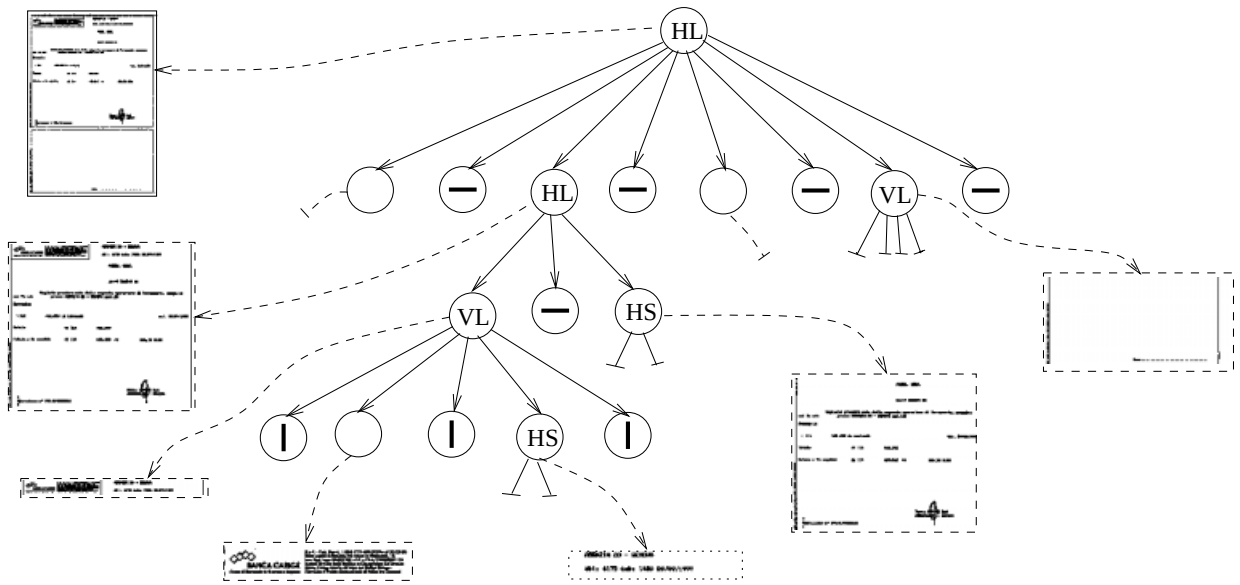


Fig. 4. A portion of the MXY tree of a page. Dotted lines point out to images of regions described in the corresponding nodes. VL (HL) denotes Vertical (Horizontal) cutting Line; HS denotes Horizontal cutting Space. Nodes with a line indicate leaves corresponding to line separators

that can confirm the physical classification made by the DDT.

Both DDT and indexing strategies are learnt by examples during an offline training phase. In particular indexing strategies can be tuned either to read information of a given class or to read information of more classes, automatically discriminating among them on the basis of their content (OCR, logo). By training both class-dependent and class-independent strategies it is possible to take advantage of all the information provided by the DDT-based classification. Three alternative results can be provided by the DDT-based classifier: (i) If the document is classified with enough confidence, a class-dependent strategy is activated; (ii) in case of rejection, a general class-independent strategy can be applied; and (iii) when n top candidate classes have higher recognition confidence, then more strategies, corresponding to the n top candidate classes, can be applied in sequence, till the reading operation is successfully completed. In all the cases when a reading strategy is activated, the strategy can either provide all the required information or reject the document for further manual processing.

3.1 Document layout analysis: modified X-Y tree

The X-Y tree [17,18] is a top-down data-driven method for page layout analysis. The basic assumption behind the X-Y tree segmentation is the fact that structured elements of the page (columns, paragraphs, titles, figures, lines of text, printed symbols) are generally laid out in rectangular blocks. Furthermore, the blocks can almost always be divided into groups in such a way that blocks that are adjacent to one another within a group have one dimension in common [17]. The Modified X-Y tree (MXY tree) [6] is an improvement of the X-Y tree approach where the splitting of regions into sub-parts can be obtained by means of cuts along horizontal and vertical separators (mostly lines), in addition to the classical cuts along white spaces. Each node of an MXY tree is associated either to a region of the page or to a separator. The MXY tree is well suited to the segmentation and the subsequent physical representation of documents with complex layout containing some lines as separators. Notable examples of these documents are invoices and bank documents.

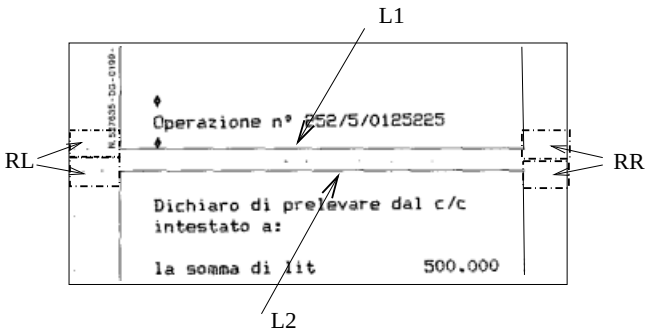


Fig. 5. Definition of regions RL and RR for lines. Line L1 cannot be used as a cutting line, since its RL rectangle is not white

In the basic X-Y tree algorithm the root of the tree is associated to the whole document which is then split into regions separated by white horizontal spaces. Each region corresponds to a node, a child of the root of the tree, and the algorithm is recursively applied on each subregion. Horizontal and vertical white spaces are alternatively used to further divide a region. The splitting process is stopped when a cutting space (either horizontal or vertical) cannot be found or when the area of the current region is smaller than the pre-defined resolution of the algorithm.

When dealing with documents containing lines, the X-Y tree decomposition algorithm can fail in splitting regions contained inside regions delimited by lines. This problem can be overcome by dealing with bottom-up segmentation algorithms (e.g., RLSA algorithm). However, in the latter case we lose the hierarchical description of the document that is very useful for document classification tasks. In order to improve segmentation capabilities of the classical X-Y tree approach, the MXY tree algorithm extracts the structure of the document by splitting along either white spaces or lines. Let C_h be the average height of characters in the region to be split. A line, L, is used to split the region R if:

- the line is longer than a given threshold (typically 20–25% of the region's width);
- the two rectangles RL and RR are white, where RL (RR) is the rectangle of height $2 \cdot C_h$ covering the area from the left (right) extrema of L to the left (right) border of R (Fig. 5).

The first condition avoids cutting along short lines at the first steps of the decomposition, whereas the second one ensures that only lines actually used as separators are taken into account. If both a line and a space can be used to split a region, the line has priority over the space. Figure 4 supplies an example of MXY tree generation.

When using the X-Y tree decomposition for noisy images an uneven segmentation can occur. These errors are either due to extraneous elements in the page (e.g., salt-and-pepper noise) or to broken objects. In addition, seriously skewed images can significantly reduce the usefulness of such an approach. In order to face these problems we followed various approaches. First, connected components smaller than the average size of characters in the document are removed from the image before applying the MXY decomposition. In this way some broken characters and dots from 'i's and 'j's are lost. However, since the MXY tree is used for document classification (Sect. 3.2) and not for field location (Sect. 3.3) this problem is less critical, since differences in MXY trees are managed by the Decision Tree. The line location algorithm (based on RLSA algorithm) is designed so as to be able to locate slightly skewed and broken lines. Moreover, documents taken into account have a reduced skew.

For the task of document classification each node of the MXY tree contains a vector with the following information:

1. a flag indicating whether the father has used a line or a space during region splitting;
2. four values describing the position of the sub-region associated with the node;
3. the average grey level of the sub-region associated with the node.

These values are used by the decision tree in order to compute the similarity among nodes of different MXY trees.

3.2 Document classification

Decision trees have been widely used for pattern classification (e.g., [20]). In a decision tree the classification is carried out in an iterative process. During this process the set of classes the pattern can belong to, is restricted, till a single class is left.

In [8,9] a decision tree (the so-called Geometric Tree, GTree) is used to model the knowledge about more than one document layout by describing a hierarchy for possible logical object arrangements. Similarly to XY trees the documents are represented by a set of recursive cuts performed over white spaces. Each example is a leaf of the GTree and intermediate nodes are recursively inserted to gather cuts, which are shared by different documents. Cuts shared by many documents are stored in nodes close to the root, whereas document-specific cuts are stored in nodes close to the leaves. In the classification process a path, leading from the root node to a leaf, is followed. At each step the input document is compared with the cuts stored in the children of the last node inserted in the path. This approach was tested on business letters without taking lines into account.

The extension of the GTree approach to documents containing lines (such as invoices and bank documents) is based on two factors: first, the document is split also along lines; second, the recursive splitting of the document is done explicitly during a segmentation phase, instead of being searched on the document image during labeling. To this purpose we introduce a decision tree (Document Decision Tree, DDT) whose nodes contain MXY trees. Both DDT building and traversal are based on an appropriate matching among sub-trees extracted from MXY trees corresponding to training samples and to the incoming document.

In the following we briefly introduce the formalism needed to describe our framework. The children of every

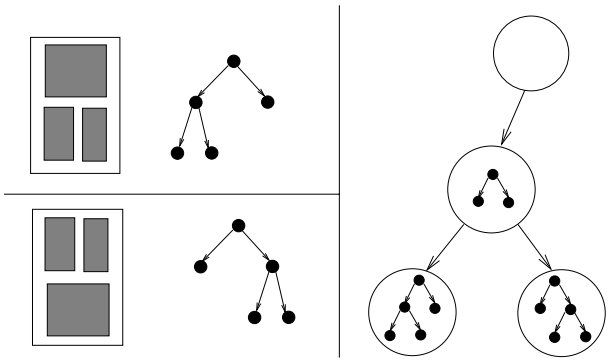


Fig. 6. Example of DDT. On the left, two documents with the corresponding MXY trees. On the right, the associated DDT containing the MXY trees of training samples in its leaves

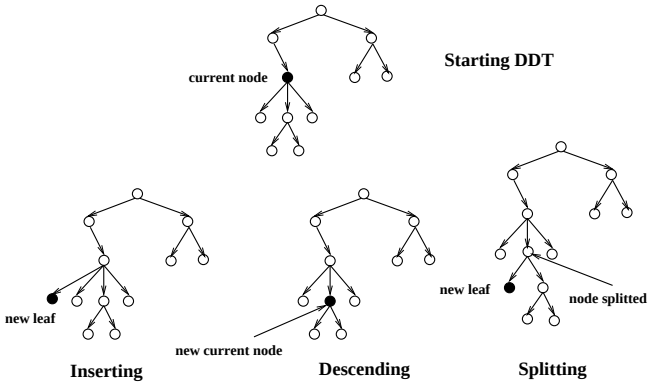


Fig. 7. The three admitted operations which are performed over document decision tree during the insertion of a new example

node are ordered. Two nodes match if they have equal label, and all nodes of the trees are supposed to be labeled with a vector of variables (defined in Sect. 3.1) assuming a set of discrete values. In order to apply matching criteria between nodes, either the labels must be quantized or a flexible match between labels, based on the Euclidean distance, must be taken into account. T_h matches T_k if their roots match, the ith child of the root of T_h matches the ith child of the root of T_k (where i varies between 0 and the number of children of the root of T_h) and this recursively holds for each node of T_h and T_k. A common sub-tree between two trees T_1, T_2 is a tree matching both a sub-tree of T_1 and a sub-tree of T_2.

Definition 1 *$CT(T_1, T_2)$ is the set of all common sub-trees between the trees T_1, T_2.*

Definition 2 *$CTN(T_1, T_2, N_i, N_j) \in CT(T_1, T_2)$ is the set of all common sub-trees between the trees T_1, T_2 which include both the node N_i of T_1 and the node N_j of T_2.*

For example $CTN(T_1, T_2, root(T_1), root(T_2))$ is the empty set if the root node of T_1 and the root node of T_2 have two different labels.

Definition 3 *The maximum common sub-tree containing a node N_i of T_1 and a node N_j of T_2 is a tree in $CTN(T_1, T_2, N_i, N_j)$ with the maximum number of nodes. We indicate this tree as $SM(T_1, T_2, N_i, N_j)$.*

We can generalize these definitions to deal with more than two trees. Let $CTN(T_1, \ldots, T_n, N_{i_1}, \ldots, N_{i_n})$ be the set of common sub-trees among the trees $T_1, \ldots, T_n$, containing the nodes N_{i_1} of T_1, $\ldots$, N_{i_n} of T_n. It can be shown that the CTN among n trees is the intersection among the CTN of each couple of trees. The maximum common sub-tree among n trees $SM(T_1, \ldots, T_n, N_{i_1}, \ldots, N_{i_n})$ is the tree $T \in CTN(T_1, \ldots, T_n, N_{i_1}, \ldots, N_{i_n})$ with maximum number of nodes. We indicate with $SMR(T_1, \ldots, T_n) = SM(T_1, \ldots, T_n, root(T_1), \ldots, root(T_n))$ the maximum common sub-tree among the trees $T_1, \ldots, T_n$, containing the root nodes of $T_1, \ldots, T_n$.

Definition 4 *A Document Decision Tree (DDT) is a tree structure with the following features:*

1. *in each node of the DDT a MXY tree is stored;*
2. *an MXY tree, representing one document instance, is stored in each leaf of the DDT;*
3. *if a node i of the DDT is the father of n nodes $j_1, \ldots, j_n$, where the trees $T_{j_1}, \ldots, T_{j_n}$ are stored, then in i the tree $SMR(T_{j_1}, \ldots, T_{j_n})$ is stored.*
4. *for each couple of trees T_1, T_2 stored inside nodes of the DDT at level i, exists a node n at a level smaller then i such as $SMR(T_1, T_2)$ is stored in node n.*

The DDT is a tree data structure, built in order to perform an efficient MXY classification, which stores the MXY trees of the example patterns. As in traditional decision trees, patterns sharing similar features are pushed close in the decision tree, while dissimilar patterns are driven apart. In our framework MXY trees sharing large SMR are forced to stay close to each other in the DDT, while trees sharing a small SMR are pushed far away in the DDT hierarchy.

Learning phase. The DDT is built from the set of MXY trees, composing the learning set, inserting one tree at a time, and updating the DDT structure in order to respect the definition of document decision tree (Definition 4). At the beginning the DDT is composed by a single empty node. The first inserted document will be stored inside the root node of the DDT. We denote with $NewXY$ the new MXY tree which must be inserted in the DDT. Let N_a be a node of DDT, $Ch_i(a)$ be the ith child of N_a, and $XY(Ch_i(a))$ be the MXY tree stored in $Ch_i(a)$. Starting from the root of DDT a path on DDT is followed until the right insertion point is found. Let L be the last node of the path. At each step three operations are admitted (Fig. 7):

1. *Descending.* If a child i exists such as: $SMR(NewXY, XY(Ch_i(L)) = XY(Ch_i(L))$ then the node $Ch_i(L)$ is added to the path followed on the DDT $(L \leftarrow XY(Ch_i(L)))$.
2. *Inserting.* If for each child i it holds that: $SMR(NewXY, XY(Ch_i(L)) = XY(L)$ then a new child is added to L. The $NewXY$ is stored in the new child. This node is a new leaf of the DDT. The insertion of the new XY tree terminates.

3. *Splitting.* If there exists a child node $Chi(L)$, such that $XY(L)$ is a proper sub-tree of $SMR(NewXY, XY(Ch_i(L))$ and $SMR(NewXY, XY(Ch_i(L))$ is a proper sub-tree of $XY(Ch_i(L))$, then a new node is added as the ith child of L where $SMR(NewXY, XY(Ch_i(L)))$ is stored. This new node has two children: the first one is a new leaf where $NewXY$ is stored and the second one is the old ith child of L. The insertion of the new XY tree terminates.

A single XY tree is taken into account at each step, till all the learning set is inserted. If the descending operation is performed, then a node expressing all the needed (at that level) information about the new example exists. If the inserting operation is performed, then a node expressing any further information about the new example cannot be found. If the splitting operation is performed, then a node expressing further information about the new example can be found, but that node does not contain all the needed information.

Classification phase. During this phase, as in traditional decision trees, an input document is associated with one DDT leaf, which represents an instance of a document inserted into the decision tree during the learning phase. The input document belongs to the class of the document stored in the selected leaf of the DDT. The classification is performed by following a path from the root to a leaf. The maximum number of steps is equal to the maximum depth of the decision tree. At each step a new node is added to the path. The selection of a child is performed by a two-stage process:

- the similarities between the XY tree representing the input document and the XY trees, stored in each child of the last node of the path, are computed;
- the child storing the XY tree most similar to the input document is selected.

When a leaf is reached the algorithm terminates. To increase the system's reliability, many paths can be followed in parallel. In this case a list of classes, sorted by their similarity with the input document, is obtained. This ranked list allows us to start an iterative information extraction procedure, by first applying the reading strategy relevant to the top scoring candidate class, and, in case of failure, by backtracking to the second top class, and so on. Only very few classes have to be considered (see Sect. 4).

Similarity criteria between trees. During the classification phase a similarity function is needed to compare MXY trees. We introduce the similarity function which has been applied in the experiments and we discuss why we think that this similarity function is more appropriate than tree metrics to guide the DDT navigation. In [10–12,21] techniques to construct a metric among trees are shown. These algorithms cannot be successfully applied to the navigation of decision trees because all these metrics are highly sensitive to the difference of dimension of the trees.

During a generic step of the classification phase the system compares the input XY tree to the XY trees stored in the children of a decision tree node. Unfortunately, the DDT construction technique above described cannot guarantee that the numbers of nodes of the XY trees, stored in the children of a node, are the same. According to tree metrics, the distance among trees composed of a non-homogeneous number of nodes is large. Since the input MXY tree is usually larger than the MXY trees stored in the intermediate nodes of the DDT, the classifier would more likely follow paths in the decision tree leading to nodes where larger trees are stored. This bias can strongly affect the performance of the classifier.

A criterion has been designed in order to compare the structural similarities between MXY trees even if the trees do not feature a homogeneous number of nodes. The nodes of the MXY trees store vectors of labels with real values, and then the node matching is modified in order to deal with continuous values: given two XY trees T_1, T_2, and a threshold t, we say that a node i of T_1 with label $l_1 \in \mathcal{R}^n$ and a node j of T_2 with label $l_2 \in \mathcal{R}^n$ match if: $|l_1 - l_2| < t$. Let $T(i)$ indicate the sub-tree of the tree T descending from the node i of T. For each node i of T_1 and each node j of T_2 we construct the tree $SMR(T_1(i), T_2(j))$. The similarity between a tree T_1 and a tree T_2 is defined as:

$$SIM = \frac{\sum_{i=1}^{|T_1|} \sum_{j=1}^{|T_2|} [|SMR(T_1(i), T_2(j))| \cdot g(T_1, T_2, i, j)]}{|T_1| \cdot |T_2|} \quad (1)$$

where $|T_1|$ is the number of nodes of tree T_1, $|T_2|$ is the number of nodes of tree T_2, and the function $g(T_1, T_2, i, j)$ weights the level difference between the node i of T_1 and the node j of T_2. Function g is a monotonic decreasing function: if the level drop between two nodes is large then the function g assumes a small value.

This similarity function scans all the nodes of the trees T_1, T_2 to find all maximum common sub-trees. The number of nodes of all these sub-trees are summed and the result is normalized by the product of the number of nodes of the two trees. This similarity function takes into account all the internal similarities between the trees and, after the normalization step, is no longer highly sensitive to the dimensionality of the input trees. Since the trees are ordered, the maximum sub-tree extraction between two trees can be performed with complexity linear to the number of nodes of the smaller tree.

3.3 Indexing strategy

The automatic indexing strategy employed in STRETCH makes use of a template structure named correlation graph [4], which allows the implementation of information extraction strategies based on image processing and reading techniques. The structure of such a graph is very similar to Form Graphs introduced in [5] for form registration and reading. The main difference is that correlation graphs are learned from

examples, instead of being explicitly built by users as in [5]. The correlation graph describes a document understanding strategy which first computes the search area for fields to be read, possibly based on the position of other already found fields. Afterwards, it uses some predefined template elements to "read" the field contents inside the search area according to the field type (for instance string, graphical element, date, total amount). Meaningful fields are of three main types: (i) fields to be read as text (ASCII strings); (ii) textual or geometric tags used by the recognition strategy to understand the document structure; and (iii) image fields that can be recognized by suitable methods (for instance, logos). Each information field is assigned a basic type that defines how its value is interpreted during retrieval. The search areas for the information fields of a given document class are automatically learnt upon the presentation of appropriate document examples, and can be further refined by the presentation of new examples. The location of fields to be read on the basis of tags has been previously explored for instance in [5,13]; however, it is important to observe that in the STRETCH approach mutual relationships are automatically learnt.

Learning and application development

The framework for strategy definition is endowed with a learning tool, which, starting from examples supplied by users, automatically generates the Java code to solve the application. The tool is a graphic environment for the development of document reading applications, which has three components:

- a graphical interface that allows users to locate the relevant fields on a set of document examples;
- a learning algorithm that performs a clustering of the examples and detects the spatial cross-correlation between fields;
- a synthesising algorithm that generates the Java code of the reading strategy.

The user has to prepare a training set for the learning algorithm, based on a set of positive examples of documents of a given class. The position of an information field is simply marked by the user on the image of each example by drawing the surrounding rectangle with the mouse. Then the user has to choose a type for the field, among a set of predefined ones. Different examples can have a different number and different types of fields. Other specific parameters can be set as well (e.g., a dictionary of textual tags).

The learning algorithm performs a clustering of the document examples marked up with the relevant fields. The clustering algorithm is based on the minimization of the information required to describe the examples themselves [4]: this amounts to a sort of Simulated Annealing. The number of clusters is not predefined: it is learnt from the examples along with the structure of the clusters. Reducing the "temperature" parameter slowly and adopting a rather small number of examples (a few tens for the applications so far explored) causes stable convergence of the clusters, apart from their order. This algorithm can be replaced by other clustering methods, notably by LVQ2 [14].

Each cluster of documents corresponds to a substrategy capable of locating and reading interesting pieces of information in the document image. The location areas are computed in terms of the overlap of the locations in the single examples where a given information field has been marked. For instance, if the field "total amount" can be located either bottom down or midway left, the documents will be partitioned into two clusters and two sub-strategies will be generated. The complete strategy for reading a specific document class typically is a *choice* among sub-strategies. The full class-dependent strategy executes the various sub-strategies sequentially, until one sub-strategy is successful; if none is, the strategy fails. The sub-strategy execution follows an automatically assigned priority which reflects an overall optimization, based in particular on the sample distribution in the training set. The output of the class strategy is a string containing all the extracted information, i.e., the content of each information field, interpreted according to its type, and its location in the document image.

The Java code implementing a recognition strategy is synthesised using pre-defined building blocks which perform the following functions:

- locate the fields to be read by computing the search areas;
- extract the field content inside a search area on the image, taking into account the field type.

Basic methods implement the elementary functions, such as reading a textual field by invoking an OCR library, classifying a logo, recognizing a tag against a dictionary, and interpreting a date. Alternative methods can be employed; for instance, for reading printed text we have actually experimented with both in-house developed solutions [7] and commercial OCR packages. A logo can be located and recognized [16] as an auxiliary confirmation of document classification. The program synthesis algorithm also finds the optimal ordering of execution of field processing in order to minimize backtracking in case of failure of the current sub-strategy.

4 Applications and experimental results

We have thoroughly tested STRETCH in two application scenarios, concerning invoices and bank documents, respectively. In both applications classes are user-defined; in the former, they correspond to suppliers and do not have one-to-one correspondence with document layout; in the latter, classes correspond to document types. In the following, we detail the goals of either application, its implementation in STRETCH, and present classification and indexing results on selected test sets of real documents.

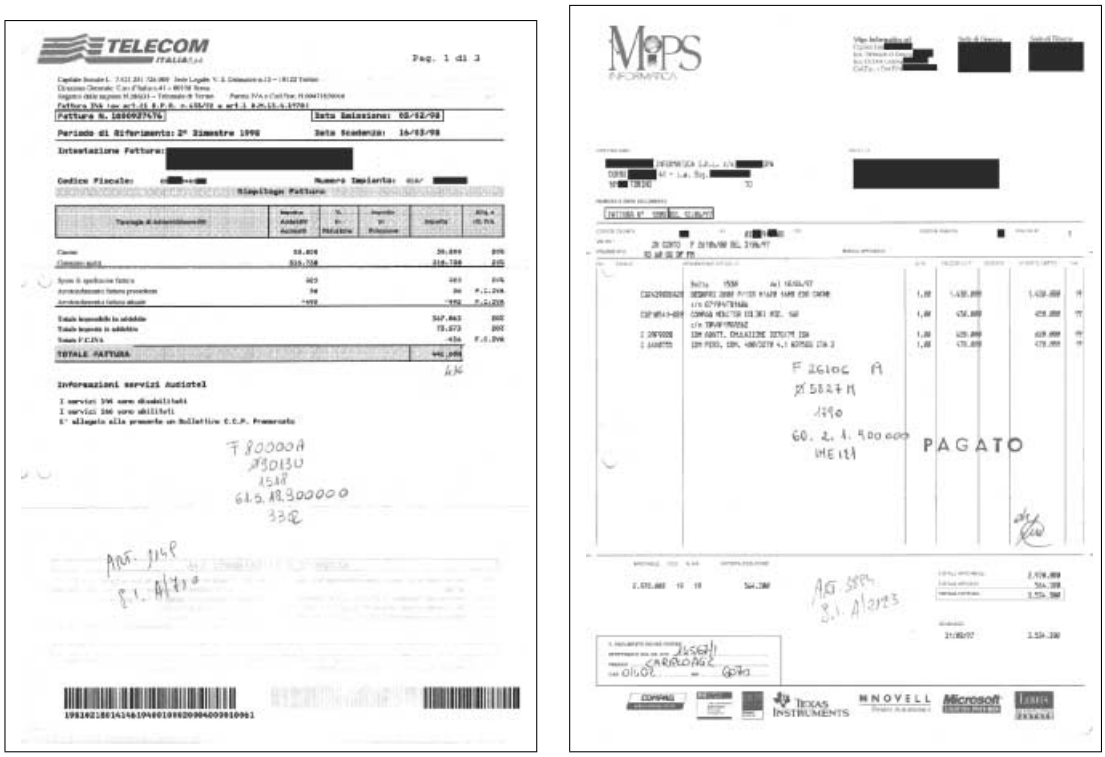


Fig. 8. Two examples of invoices. Personal data are blackened for privacy reasons

4.1 Passive invoices workflow

In a Company or Institution, passive invoices are the documents issued by suppliers or consultants which require payments relevant to given services or goods. In this scenario, STRETCH aims at providing both a digital archive and automated data entry for Value Added Tax (VAT) recording, as required by tax regulations. Two examples of invoices processed by STRETCH are depicted in Fig. 8.

The goals of the application are:

- Classify the documents according to user-defined classes which correspond to the supplier who issued the invoice.
- Locate and read the specific information fields, whose contents are used used both as an index and for VAT recording, that is: *Supplier* name; *Date of issue*; *Invoice number*; *Total amount* of the invoice; *IVA*: the total amount of Italian VAT tax.

The ICR reader embedded in the reading strategy reads the information written on invoices issued by a given supplier. If the supplier identification fails, a general reading strategy for the invoice domain is applied. The results of the indexing procedure are exemplified in Fig. 9, where the contents of the information fields (Information Summary) are presented together with the corresponding areas in the document image.

Classification results. The selected data set consists of 753 binary images (300×300 dpi) of real passive invoices of a company of the Finmeccanica Group, issued by nine different suppliers (nine classes). These images (containing complete invoices) have a skew less than 7 degrees, and the noise is small. No specific filtering or enhancement was applied to the images. Twenty documents per class were randomly selected to create a training set and the other 573 images constituted the test set. A document is considered as "correctly classified" when the most probable class assigned by the decision tree is the correct one. In the performed tests 560 documents (97.8%) were correctly classified; for nine documents (1.5%) the second candidate is the correct one; and for the remaining four documents (0.7% cases) the third candidate is the correct one. The correct class never ranks beyond the third position.

Indexing results. The documents used as a test set for indexing (250 invoices) were a subset of the above described data set. They show different layouts, various styles, and many different fonts and font sizes. All the invoices show a company logo, usually in one-to-one correspondence with the supplier, the unique exception are the invoices issued by one supplier that have neither a fixed layout, nor a unique "standard" filling style.

The indexing performance was the following: the complete reading strategy (field location, proprietary neural OCR and tag dictionary) produced a total of 31 misclassification errors (96.9% correct on information fields, 100% on tags). Errors were mainly encountered with very noisy images, dot matrix, and italic fonts.

4.2 Bank account notes archive

This scenario concerns the management of a high-volume document repository, produced day-by-day by

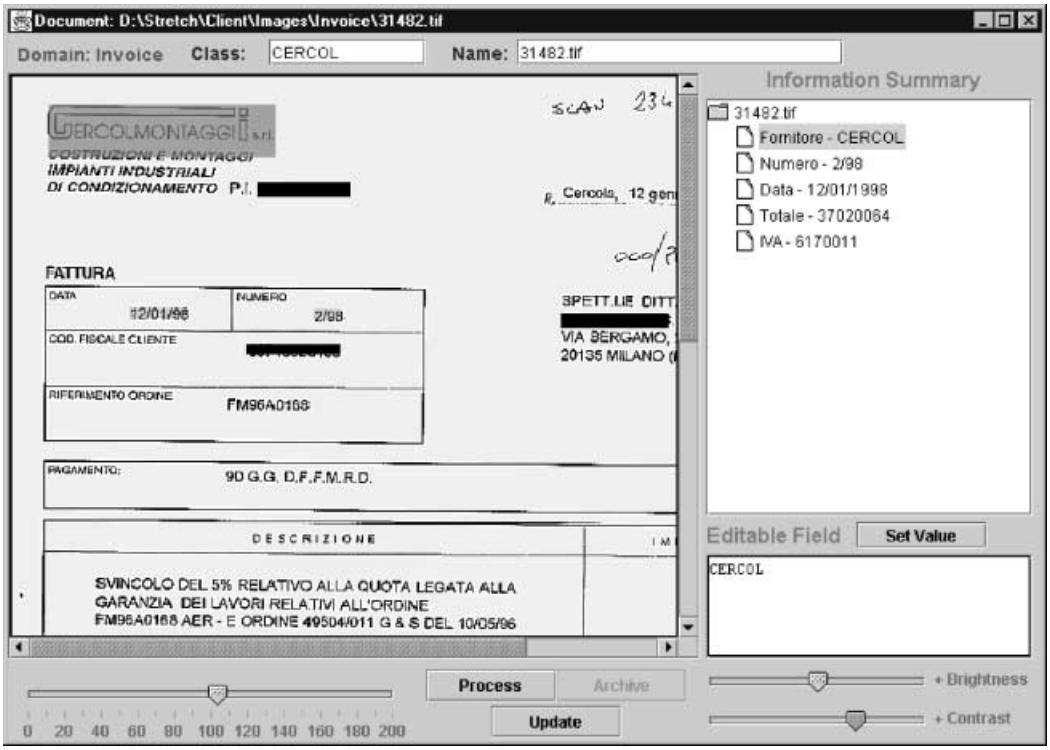


Fig. 9. Summary of the information ("indexes") extracted from a document (invoice) with the corresponding location in the document image. Personal data are masked

the branches of a Bank. The documents can be of several different types (see Fig. 10): account notes, cheques, document batch headers, and different types of enclosures. The most important are the account notes, which describe a cash operation, often together with enclosures. All the account notes and related enclosures must be stored in their original paper form for legal reasons. The physical storage and the effort spent in document searches represent a significant cost for every bank[2]. The document storage service, performed by an appointed Service Bureau, consists in physically gathering all the account notes and related documents, scanning both the front and rear sides, indexing and archiving the images, then performing physical paper storage whose location is included in the document indexes in the archive.

The goals of the application are:

- Classify the documents according to their type, in particular account notes (class "Contabile"), cheques (class "Assegno"), and batch headers (class "Testapacco"); all the other documents are generically regarded as Enclosures.
- For documents of either class "Contabile" or class "Assegno" locate and read a specific information field, corresponding to the operation number or the cheque number, respectively.

The top-level strategy designed for this application is described in [2], along with some preliminary experimental results.

[2] A medium size bank with some hundred branches produces from 30,000 up to 100,000 account notes and enclosures a day.

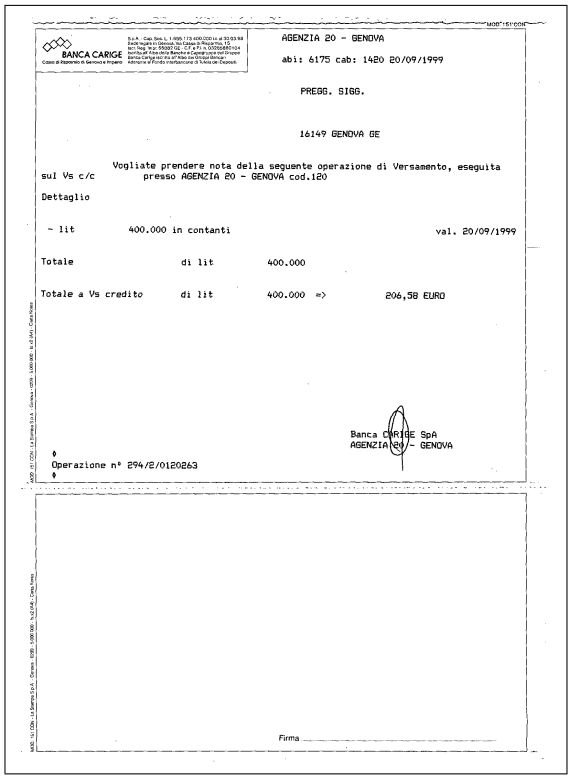
Classification results. In the preliminary experiments [2], the test set consisted of 541 heterogeneous documents issued by a large Italian bank group. These documents belong to any of the possible classes involved in the application and show several different layouts, where information fields are located with high variability. Black and white images (200×200 dpi resolution) were produced by high volume scanners. No specific filtering or enhancement was applied to the images.

The Decision Tree was generated using a total of 67 documents: six Testapacco, five Assegno, 24 Contabile, and 32 mixed Enclosures (the latter class is actually under-represented, as in the test set it contains more than 20 different form types and also rear sides).

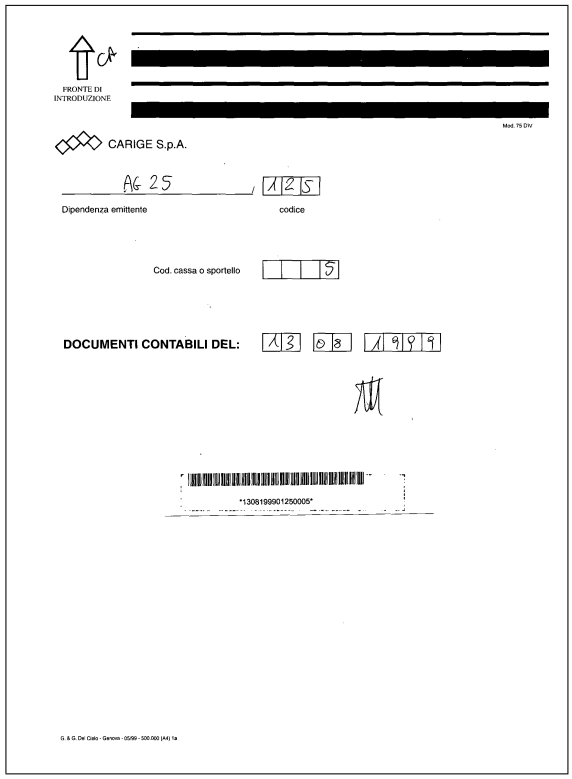
The document classification results, summarized in Table 1, are quite encouraging: the Decision Tree-based classification achieved 99% document classification accuracy[3] over the three main classes (Table 1a), or 91% also taking all the enclosures into account (Table 1b). By adding classification confirmation performed by appropriate reading strategies looking for some pre-defined tags (which, in particular, is necessary to discriminate one type of Enclosures from Contabile), 100% on the three main classes and 96% on all the documents was achieved.

Further tests were conducted on larger document sets, without re-training the original Decision Tree. In particular, classification of all the documents in live batches run at the Service Bureau servicing the

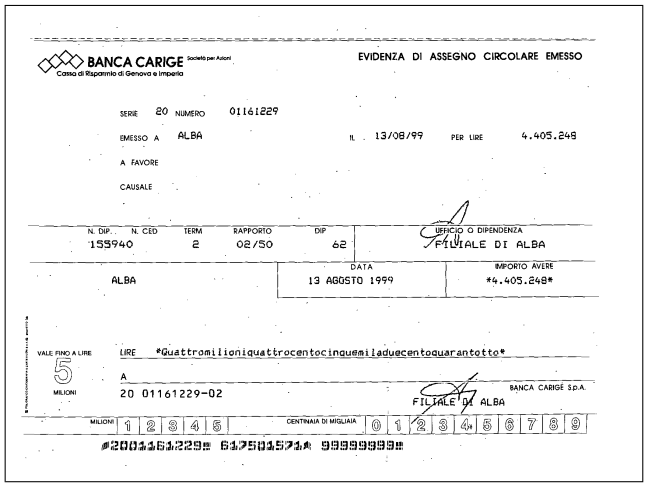
[3] Accuracy is defined as the fraction of non-rejected documents which are correctly processed.



a



b



c

Fig. 10a–c. Examples of three types of bank documents: **a** *account note*, **b** *batch header*, and **c** *cheque*. Personal data have been removed in these examples

Bank was performed. In this case, the classification "groundtruth" of each document was available, not the reading groundtruth (i.e., the real content of each information field). Therefore, classification accuracy can be measured, but not reading accuracy. The results we obtained in a typical batch are reported in Table 2. They correspond to 98.2% of correct classification on the three main classes (batch header, cheque, account note), with 0.7% rejection, i.e., the accuracy is 98.9%.

Indexing results. On the same test set employed for the preliminary classification experiments, where the reading groundtruth was available, we also performed indexing tests. Two different commercial OCR's were tested within the system for text reading. Table 3 presents the reading results for the two document classes requiring full indexing, obtained with the two different OCRs. With OCR1, the information fields were correctly located in 96% of cases. With both OCR1 and OCR2 the errors in reading the field values were partly due to lack of context processing (numeric-only fields were not forced, hence alphanumeric characters were confused, such as 'S' for '6', 'B' for '8', and so on). Serious problems were mainly encountered with very noisy or corrupted images, which are very common in this domain because of poor printing quality: in fact, the documents are al-

Table 1. Preliminary results on 541 Bank documents for document classification (top-rank class only). **a** main classes ("Testapacco", "Assegno", "Contabile"); **b** the three main classes plus Enclosures

	Total	Accuracy	Correct		Error		Reject		Confusion Matrix			
Class			#	%	#	%	#	%	Test.	Ass.	Cont.	Rej
Testapacco	19	100	19	100	0	0	0	0	19	0	0	0
Assegno	36	100	36	100	0	0	0	0	0	36	0	0
Contabile	277	99	275	99	1	1	1	0.4	1	0	275	1
Total	332	**99**	330	**99**	1	**1**	1	**0.3**	20	36	275	1

a

	Total	Accuracy	Correct		Error		Reject		Confusion Matrix				
Class			#	%	#	%	#	%	Test.	Ass.	Cont.	Encl.	Rej
Testapacco	19	100	19	100	0	0	0	0	19	0	0	0	0
Assegno	36	97	35	97	1	3	0	0	0	35	0	1	0
Contabile	277	99	273	99	3	1	1	0	1	0	273	2	1
Enclosures	209	76	134	64	42	20	33	16	2	13	27	134	33
Total	541	**91**	461	**85**	46	**9**	34	**6**	22	48	300	137	34

b

Table 2. Document classification results on 1,390 Bank documents (three main classes) from a live batch of the Service Bureau

Output class Input class	Contabile	Assegno	Testapacco	Rejected	Total
Contabile	**1191**	15	0	8	1214
Assegno	0	**118**	0	0	120
Testapacco	0	0	**56**	0	56
Total	1191	133	56	10	**1390**

Table 3. Results on Bank documents: field location and reading

	OCR 1			OCR 2		
	Assegno	Contabile	Total	Assegno	Contabile	Total
Correct	19	111	130	30	72	102
1 error	10	43	53	1	58	59
2 errors	4	37	41	1	26	27
≥ 3 errors	–	75	75	3	39	42
Inserted char.	1	1	2	–	7	7
Field not located	2	10	12	1	75	76
Total	36	277	313	36	277	313

ways second copies of the original (which is given to the bank customers). To solve this problem some appropriate image pre-processing appears to be mandatory.

4.3 Timing performance

Full processing time (classification and indexing) was 1.6 s per document on the average (documents A4 or smaller) on a 500 Mhz Pentium III with 512 MB RAM, running a single Document Processing Server. This is comparable with the product currently adopted in the Service Bureau, a high-end commercial ICR product, which employs 1.4 s per document running on a similar configuration. However, the latter is not endowed with rejection capabilities, hence all the documents must be verified manually, while with STRETCH only a small percentage of documents are rejected. The time for manual verification is therefore greatly reduced and the total average time required to process each document is smaller than with the commercial product. Moving to inexpensive parallel configurations, well affordable by Service Bureaus, will make the system even more efficient.

To evaluate the advantages of a parallel execution of more DPS on different processors we performed some speedup benchmarks. Figure 11 represents the speedups ($speedup = \frac{sequential-time}{parallel-time}$, considering the elapsed time measured by the client for executing the whole batch) ob-

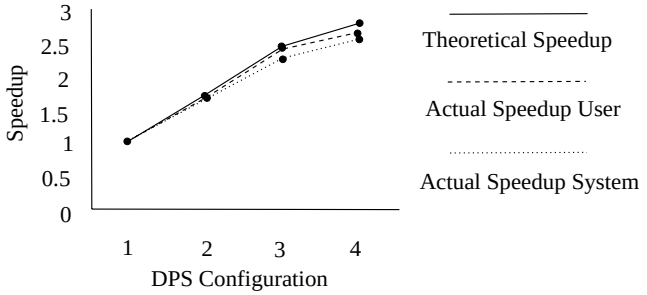


Fig. 11. Benchmark of document processing speedup

tained with up to 4 DPS instances on four heterogeneous processors. "Actual Speedup User" means with no image archiving, while "Actual Speedup System" includes it. The theoretical speedup is computed by taking into account the different clocks of the Pentium processors added one by one to the configuration (500, 500, 450, and 200 Mhz, respectively), with respect to the 600 Mhz power alone. The test was performed on a set of 270 documents of the bank application.

The best solution, currently under specification, will probably be the integration of the STRETCH Document Processing Server for document classification with a set of high-end ICR's specialized for reading different document types. In fact, reliable document classification would allow us to select among a set of different OCR/ICR systems, each one well parameterized for a specific document type, improving the whole service productivity. More specifically, the same OCR can be tuned according to different reading strategies suited for each document type.

5 Conclusions

In this paper we have presented STRETCH, an archiving system featuring a Document Processing Server based on a novel approach for classification and indexing of heterogeneous documents. In the preliminary experiments on invoices and bank documents, very encouraging results have been obtained; in particular, document classification accuracy is satisfactory.

Future work will aim at improving the overall system performance and accuracy to enable its engineering as a solution for high-volume document processing. In particular, the integration in STRETCH of a more accurate ICR system with proper image pre-processing and contextual analysis will also yield an acceptable reading accuracy. As regards pre-processing, we intend to work in two directions: improve de-skewing capabilities, in order to make the line extraction algorithm more robust and more efficient, and introduce character smoothing and completion algorithms, to overcome the poor image quality of second copy documents. As regards contextual post-processing, we will first of all introduce a"numeric only" attribute for textual fields. We will also explore the application of specific dictionaries for some fields and cross-checking between fields when possible (for instance, between"VAT" and "total amount").

Another line of improvement concerns alternative clustering methods for the Application Learning tool. LVQ-based methods will be introduced to accommodate larger training sets while still reaching learning convergence.

An interesting prospective application of STRETCH aims at avoiding document pre-sorting in the production line of service bureaus in charge of document acquisition and storage. In fact, employing STRETCH even just for document classification is expected to be highly beneficial. High-end commercial tools can show up to 97% document classification accuracy, but with no rejection capabilities: all the documents in a batch must therefore be verified by operators. Productivity can hence be greatly enhanced by exploiting a system featuring both rejection capabilities and high document classification accuracy. In this case, manual verification can be applied to the rejected documents only. A rejection of few percent will result in conspicuous cost reduction and improved performance. Another possibility is the use of STRETCH document classification module as a front-end to high performing OCR solutions.

Finally, a reliable document classification system will allow the selection of either a specific tuning of the same OCR or even different OCR's, each tuned to a specific document (sub-)class, to improve the whole service productivity.

Acknowledgements. We would like to acknowledge Sandra Bruzzo, Pietro Pedrazzi, Christian Pisani, and Rosa Martino (Elsag) for their valuable support in implementing and testing STRETCH; Enrico Francesconi (Dip. Sistemi e Informatica University of Firenze) for useful discussions and suggestions. This Project has been supported by the European Community under a grant in Programme ESPRIT - RTD n. 24977.

References

1. E. Appiani, L. Boato, S. Bruzzo, A.M. Colla, M. Davite, D. Sciarra: STRETCH: a system for document storage and retrieval by content. In: Proc. of DAUDD '99 Workshop, Florence, Italy, September 1999, pp. 588–592, 1999
2. E. Appiani, A.M. Colla: Automatic analysis and indexing of variable-layout documents. In: Proc. RIAO2000, Paris, France, April 12-14, 2000, pp. 980–987, 2000
3. S. Baumann, M.B H. Ali, A. Dengel, T. Jager, M. Malburg, A. Weigel, C. Wenzel: Message extraction from printed documents - a complete solution. In: Proc. Fourth International Conference on Document Analysis and Recognition, pp. 1055–1059, 1997
4. L. Boato, E. Cattani, M. Davite, B. Villa: Automatic programming of variable layout image documents reading applications based on minimum description length induction. In: AI*IA Workshop on Automatic Learning and Natural Language, Torino (Italy) September 1997
5. F. Cesarini, M. Gori, S. Marinai, G. Soda: INFORMys: a flexible invoice-like form reader system. IEEE Trans. PAMI, 20(7):730–745, 1998
6. F. Cesarini, M. Gori, S. Marinai, G. Soda: Structured document segmentation and representation by the modi-

fied X-Y tree. In: Proc. Fifth International Conference on Document Analysis and Recognition, pp. 563–566, 1999
7. A.M. Colla, P. Pedrazzi: Single and coupled neural handprinted character classifiers. In: International Conference on Artificial Neural Networks, pp. 969–972, Sorrento, Italy, May 1994. Springer, Berlin Heidelberg New York
8. A. Dengel: Initial learning of document structure. In: Proc. Second International Conference on Document Analysis and Recognition, pp. 86–90, 1993
9. A. Dengel, F. Dubiel: Clustering and classification of document strcture – a machine learning approach. In: Proc. Second International Conference on Document Analysis and Recognition, pp. 587–591, 1993
10. M.A. Eshera, K.S. Fu: A graph distance measure for image analysis. IEEE Trans. Syst. Man Cybern., 14(3):398–408, 1984
11. M.A. Eshera, K.S. Fu: An image understanding system using attributed symbolic representation and inexact graph-matching. IEEE Trans. Pattern Anal. Mach. Intell., 8(5):604–617, 1986
12. F. Esposito, D. Malerba, G. Semeraro: A syntactic distance for partially matching learned concepts against noisy structural object descriptions. Int. J. Expert Syst., pp. 409–451, 1992
13. T.M. Ha, H. Bunke: Model-based analysis and understanding of check forms. Int. J. Pattern Recognition Artif. Intell., 8(5):1053–1081, 1994
14. T. Kohonen: The self-organizing map. Proc. IEEE, 78(9):1464–1480, 1990
15. H. Maus: Towards a functional integration of document analysis and understanding in workflow management systems. In: Proc. of Workflow Management '99: workflow-based applications, pp. 83–97. University of Muenster, 1999
16. M. Corvi, E.Ottaviani: Multiresolution logo recognition. In: Proc. Int. Workshop on Advances in Visual Form Analysis, pp. 110–119, Capri, Italy, May 1997
17. G. Nagy, S. Seth: Hierarchical representation of optically scanned documents. In: Proc. 7th International Conference on Pattern Recognition, pp. 347–349, 1984
18. G. Nagy, M. Viswanathan: Dual representation of segmented technical documents. In: Proc. First International Conference on Document Analysis and Recognition, pp. 141–151, 1991
19. A. Pope: The Corba Reference Guide. Addison-Wesley, Reading, Mass., USA, 1998
20. J. Quinlan: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco, Calif., USA, 1993
21. A. Sanfeliu, H. Bunke: Syntactic pattern recognition and application. World Scientific, Singapore, 1994
22. S.L. Taylor, R. Fritzson, J.A. Pastor: Extraction of data from preprinted forms. Mach. Vision Appl., 5(5):211–222, 1992

Enrico Appiani received his degree in Electronic Engineering in 1981 from the University of Genova, Italy. Since 1981 he has been with Elsag SpA, formerly joining the Central R&D department for the engineering of Elsag's parallel embedded computers. From 1991 to 1995 he was responsible for system software design and development. He is now in charge of technological innovation on systems and software architectures for network-based applications with image processing, joining the Image Recognition Technology department. His current interests include distributed systems architectures, application performance evaluation, middleware, communication protocols, remote sensors, Internet-based solutions. Enrico has been project leader of Esprit-funded, and presently of IST-funded, R&D projects for high-performance image processing, document indexing, intelligent surveillance. He has authored 10 papers in international conferences and magazines and is a member of IEEE.

Francesca Cesarini Francesca Cesarini received her degree in mathematics from Università di Firenze, Italy, and since 1983 she has been Associate Professor of Computer Science at the Department of Systems and Computer Science of the Università di Firenze. Her research interests include data structures and algorithms for database management systems, object data model, and techniques for document representing and understanding.

Anna Maria Colla received her degree in Mathematics in 1976 and her Ph.D. in Physics in 1979 from the Università di Genova, Italy. Since 1981 she has been with Elsag SpA, Genova, formerly with the Central Research department and then with the Image Recognition Technology department of the Large Systems business unit. Her current research interests include neural networks, pattern recognition, document processing and intelligent surveillance systems. She has been project manager of Italian and Esprit-funded R&D projects, currently of the Italian Research Programme on Micro- and Bio-Electronics, Theme 5: "Neural Systems for Automation of Services and Plants". She has been a member of the organizing committees of several international conferences, authored more than 40 scientific papers, and is a member of the Italian Neural Network Society (executive board), of the European Neural Network Society and IEEE.

Michelangelo Diligenti received his M.Sc. degree in Tele-communication Engineering in 1998 at the University of Siena, Italy. Currently he is a Ph.D. student at the Università di Firenze, Italy. He has collaborated with the University of Wollongong and the Nec Research Institute. His main research interests are in pattern recognition, document classification, visual databases and machine learning applied on the world wide web.

Marco Gori received the Laurea in electronic engineering from the Università di Firenze, Italy, in 1984, and his Ph.D. degree in 1990 from Università di Bologna, Italy. From October 1988 to June 1989 he was a visiting student at the School of Computer Science (McGill University, Montreal). In 1992, he became an associate professor of computer science at the Università di Firenze and, in November 1995, he joined the Università di Siena, where he is currently professor of computer science. His main research interests are in neural networks, pattern recognition, and applications of machine learning to information retrieval on the Internet. He organized many scientific events, including the NIPS'96 post-conference workshop on "Artificial Neural Networks and Continuous Optimization: Local Minima and Computational Complexity". He was the co-Director of the international summer school on "Adapting Processing of Sequences" held in Salerno on September 1997 and is currently acting as the program chair of the International Joint Conference on Neural Networks (July 24-28, 2000). Dr. Gori serves as an Associate Editor of a number of technical journals related to his areas of expertise, including the IEEE Trans. Neural Networks, Neurocomputing, Neural Computing Survey, Pattern Analysis and Application, the International Journal on Computer Research, and the International Journal on Pattern Recognition and Artificial Intelligence. He is the Italian chairman of the IEEE Neural Network Council and is a senior member of the IEEE.

Simone Marinai received the Laurea in Electronic Engineering in 1992, from the Università di Firenze, Italy. He obtained his PhD degree in computer science in 1996 defending a thesis on the extraction of information from structured documents. Currently he is Assistant Professor at Università di Firenze. His main research interests are in pattern recognition, neural networks, and document processing applications. He was the chairman of the workshop "Document Analysis and Understanding for Document Databases" (DAUDD) held in Firenze in 1999. He is a member of IAPR.

Giovanni Soda received his degree in Mathematics from the Università di Firenze, Italy, in 1969. From 1971 he was researcher at the National Council of Research, where his activity included formal systems for language manipulation. Since 1975 he has been at the Università di FIrenze where he is presently Professor of Artificial Intelligence at the Department of Systems and Computer Science (DSI) of the Università di Firenze. Giovanni Soda's current research interests include Knowledge Representation Systems, integration of Artificial Intelligence techniques with Neural Networks, pattern recognition and Document Processing, in particular. He was the general chairman of the AI*IA95 held in Firenze in 1995 and was Program Chairman of the DEXA99 Conference. He is a member of the executive board of AI*IA (the Italian Association of Artificial Intelligence) and is a member of the IEEE, ACM, IAPR Societies.