UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II
Scuola di Dottorato in Ingegneria dell'Informazione
Dottorato di Ricerca in Ingegneria Informatica ed Automatica

Comunità Europea
Fondo Sociale Europeo

EXTRACTING AND SUMMARIZING INFORMATION FROM LARGE DATA REPOSITORIES

MASSIMILIANO ALBANESE

Tesi di Dottorato di Ricerca

Novembre 2005

via Claudio, 21- I-80125 Napoli - ☎ *[#39]* (0)81 768 3813 - *[#39]* (0)81 768 3816

UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

Scuola di Dottorato in Ingegneria dell'Informazione
Dottorato di Ricerca in Ingegneria Informatica ed Automatica

EXTRACTING AND SUMMARIZING INFORMATION FROM LARGE DATA REPOSITORIES

MASSIMILIANO ALBANESE

Tesi di Dottorato di Ricerca

(XVIII ciclo)

Novembre 2005

Il Tutore

Prof. Antonio PICARIELLO

_____

Il Coordinatore del Dottorato

Prof. Luigi Pietro CORDELLA

_____

Il Co-Tutore

Prof. V.S. SUBRAHMANIAN

Dipartimento di Informatica e Sistemistica

via Claudio, 21- I-80125 Napoli - ☎ *[#39]* (0)81 768 3813 - *[#39]* (0)81 768 3816

Abstract

Information retrieval from large data repositories has become an important area of computer science. Research in this field is highly encouraged by the ever-increasing rate with which today's society is able to produce digital data. Unfortunately most of such data (e.g. video recordings, plain text documents) are unstructured. Two major issues thus arise in this scenario: i) extracting structured data – information – from unstructured data; ii) summarizing information, i.e. reducing large volumes of information to a short summary or abstract comprising only the most essential facts.

In this thesis, techniques for extracting and summarizing information from large data repositories are presented. In particular the attention is focused onto two kinds of repositories: video data collections and natural language text document repositories. We show how the same principles can be applied for summarizing information in both domains and present solutions tailored to each domain. The thesis presents a novel video summarization algorithm, the Priority Curve Algorithm, that outperforms previous solutions, and three heuristic algorithms, OptStory$^+$, GenStory and DynStory, for creating succinct stories about entities of interest using the information collected by algorithms that extract structured data from heterogenous data sources. In particular a Text Attribute Extraction (TAE) algorithm for extracting information from natural language text is presented. Experimental results show that our approach to summarization is promising.

Acknowledgements

This work would certainly not have been possible without the support of many people. I'd like to acknowledge Prof. Lucio Sansone, who has been my master thesis' advisor, for having given me lots of useful suggestions. I'd like to thank Prof. Antonio Picariello, my master thesis' co-advisor and my Ph.D. thesis advisor, for having guided me during these years. I'd like to thank Prof. V.S. Subrahmanian at University of Maryland, my Ph.D. thesis' co-advisor, for having given me the chance to spend several months at his laboratory and start the challenging projects around which the work presented in this thesis is built. I'd like to acknowledge Prof. Angelo Chianese, the other cornerstone of our research group together with Prof. Sansone, for having helped me to start my experience at University of Naples. I'd like to thank Prof. Luigi Cordella, chair of Naples' Ph.D. School in Computer Science and Engineering, for dedicating so much of his time to the School. I'd like to acknowledge Prof. Carlo Sansone, who has guided me in a research activity that is not mentioned in this work.

I'd also like to acknowledge all friends, colleagues and relatives who supported me in these years. Finally, I'd like to acknowledge myself for having had the willingness to pursue this goal.

Table of Contents

List of Figures

List of Tables

Part I

State of the Art

Chapter 1

Introduction

1.1 Information Retrieval from Large Data Repositories

Information retrieval (IR) deals with the representation, storage, organization of, and access to information items. The representation and organization of the information items should provide the user with easy access to the information in which he/she is interested. Given a user query, the key goal of an IR system is to retrieve information which might be useful or relevant to the user. Unfortunately, characterization of the user information need is not a simple problem. On the other hand, the explosive growth of digital technologies has made available huge amounts of data, making the problem of retrieving information even more complex. Such great amounts of data also require a new capability for any modern information retrieval system: the capability of automatically summarizing large data sets in order to produce compact overviews of them.

Information retrieval is a wide, often loosely-defined term. Unfortunately the word *information* can be very misleading. In the context of information retrieval, information is not readily measured in the technical meaning given in Shannon's theory of communication [57]. In fact, in many cases one can adequately describe the kind of retrieval by simply substituting "document" for "information". Nevertheless, "information retrieval" has become accepted as a description of the kind of work published by Sparck Jones [61], Lancaster [34] and others. A perfectly straightforward defini-

tion along these lines is given by Lancaster [34]: *"Information retrieval is the term conventionally, though somewhat inaccurately, applied to the type of activity discussed in this volume. An information retrieval system does not inform (i.e. change the knowledge of) the user on the subject of his inquiry. It merely informs on the existence (or non-existence) and whereabouts of documents relating to his request."* This definition specifically excludes Question Answering systems and Semantic Information Processing systems. It also excludes data retrieval systems such as those used by, for instance, the stock exchange for on-line quotations.

1.1.1 Information versus Data Retrieval

Data retrieval (DR), in the context of an IR system, consists mainly of determining which documents of a collection contain the keywords in the user query which, most frequently, is not enough to satisfy user information needs. In fact, the user of an IR system is concerned more with retrieving information about a subject than with retrieving data which satisfies a given query. A data retrieval language aims at retrieving all objects which satisfy clearly defined conditions such as those in a regular expression or in a relational algebra expression. Thus, for a data retrieval system, a single erroneous object among a thousand of retrieved objects means total failure. For an information retrieval system, however, the retrieved objects might be inaccurate and small errors are likely to go tolerated. The main reason for this difference is that information retrieval usually deals with natural language text which is not always well structured and could be semantically ambiguous. On the other hand, a data retrieval system (such as a relational database management system) deals with data that has a well defined structure and semantics. Data retrieval, while providing a solution to the user of a database system, does not solve the problem of retrieving information about a subject or topic. To be effective in its attempt to satisfy user information needs, the IR system must somehow "interpret" the contents of the information items (documents) in a collection and rank them according to a degree of relevance to the user query. This "interpretation" of a document content involves extracting syntactic and semantic information from

Feature	**Data Retrieval**	**Information Retrieval**
Matching	Exact match	Partial match, best match
Inference	Deduction	Induction
Model	Deterministic	Probabilistic
Classification	Monothetic	Polythetic
Query language	Artificial	Natural
Query specification	Complete	Incomplete
Items wanted	Matching	Relevant
Error response	Sensitive	Insensitive

Table 1.1: Data Retrieval versus Information Retrieval

the document text and using this information to match user information needs. The difficulty is not only knowing how to extract this information but also knowing how to use it to decide relevance. Thus, the notion of relevance is central to information retrieval. In fact, the primary goal of an IR system is to retrieve all the documents which are relevant to a user query while retrieving as few non-relevant documents as possible.

Table 1.1 lists some of the distinguishing properties of data and information retrieval. Let us now consider each item in the table in more details. In data retrieval we are normally looking for an exact match, that is, we are checking to see whether an item satisfies or not certain properties. In information retrieval we usually want to find those items which partially match the request and then select from them the best matching ones.

The inference used in data retrieval is of the simple deductive kind. In information retrieval it is far more common to use inductive inference; relations are only specified with a degree of certainty or uncertainty and hence our confidence in the inference is variable. This distinction leads one to describe data retrieval as deterministic but information retrieval as probabilistic. Frequently Bayes' Theorem is invoked to carry out inferences in IR, but in DR probabilities do not enter into the processing.

Another distinction can be made in terms of classifications that are likely to be useful. In DR we are most likely to be interested in a monothetic classification, that is, one with classes defined by objects possessing attributes that are both necessary and sufficient to belong to a class. In IR polythetic

classification is used instead. In such a classification each member of a class will possess only some of all the attributes possessed by all the members of that class. Hence no attribute is necessary nor sufficient for membership to a class.

The query language for DR will generally be of the artificial kind, one with restricted syntax and vocabulary, while in IR natural language is preferred although there are some notable exceptions. In DR the query is generally a complete specification of what is wanted, while in IR it is invariably incomplete. This last difference arises partly from the fact that in IR we are searching for relevant documents as opposed to exactly matching items. The extent of the match in IR is assumed to indicate the likelihood of the relevance of that item. One simple consequence of this difference is that DR is more sensitive to errors, in the sense that an error in matching will not retrieve the wanted item, which implies a total failure of the system. In IR small errors in matching generally do not affect performance of the system significantly.

1.1.2 Focus of the Thesis

The topic of this thesis is related to the general area of information retrieval. In particular the work focuses on extracting and summarizing information from large data repositories and presents techniques and algorithms to identify succinct subsets of larger data sets: such techniques are applied to different kinds of data. Two major scenarios are considered throughout the thesis: digital video collections and the world wide web (or any other collection of text documents). In fact it is well known that digital video data represent the most voluminous type of data in the field of multimedia databases. On the other hand, the world wide web represents nowadays a huge and global information repository, counting billions of documents.

From an IR point of view, both digital video and text documents represent unstructured data: the information content embedded in such object is not immediately usable by an IR system. It is easy to access the second paragraph of a text document or the last 5 minutes of a videoclip, but it is not that trivial to access the first paragraph that deals with a certain topic

or the video segment in which a certain action occurs. Section 1.1.1 pointed out that, in order to be effective in its attempt to satisfy user information needs, an IR system must somehow "interpret" the content of the documents in a collection and rank them according to a degree of relevance to the user query. This "interpretation" of a document content involves the extraction of syntactic and semantic information from the document and the ability to use this information to match user information needs. If we assume that the primary goal of an IR system is to retrieve all the documents which are relevant to a user query while retrieving as few non-relevant documents as possible, an overall interpretation of each document may be enough to select the relevant documents (an entire text or an entire video).

However with the exponential growth of the amounts of available data, a second level of abstraction of information from the results of the first round of IR becomes necessary. That is, the large number of documents returned by IR systems need to be summarized, in order to reduce the large volume of information to a short summary or abstract comprising only the most essential items. An overall interpretation of each document is not still enough to perform this new task, but detailed understanding of any single piece of information in a document is required: knowledge/information extraction techniques are thus required to represent the information content of a document in a well structured way. Information extraction also enables other applications such as Question Answering (QA), that allows to get targeted and precise answers to specific questions.

In this work we present knowledge extraction and summarization techniques tailored to each of the two scenarios mentioned above. The reason for dealing with such different scenarios is that both of them require to address similar issues in order to achieve the goal of the summarization task, independently from the inherently different nature of the two kinds of data. In fact, we will show as the same criteria and similar algorithms can be applied to solve the two problems.

In the video databases context, we will show that the knowledge extraction phase requires the segmentation of videoclips into meaningful units (shots and scenes) and the identification of events occurring and objects

appearing in each unit, while, the summarization task requires the selection of a subset of those units, such that certain constraints (e.g. the maximum allowed length for the summary) and properties (e.g. continuity and no repetition) are satisfied.

In the context of text documents we propose a technique to extract structured information from natural language text and use such information to build succinct stories about people, places, events, etc., such that certain constraints and properties are satisfied.

The major contributions of this work are

- the Priority Curve Algorithm (PriCA) for video summarization;
- the Text Attribute Extraction (TAE) Algorithm for extracting structured information from natural language text;
- a Named Entity Recognition algorithm (T-HMM) for recognizing in a set of text documents the entities of interest to a given knowledge domain;
- three heuristic algorithms (OptStory$^+$, GenStory and DynStory) for generating stories out of the information collected by the TAE algorithm.

The thesis is organized as follows. The remainder of this chapter introduces the basic concepts of information extraction end summarization. Chapter 2 describes the state of the art in both video summarization and automatic story creation, also discussing several related issues. Chapters 3, 4 and 5 present the original contributions of this work. Chapter 3 first introduces the CPR model for video summarization, then presents the PriCA framework – based on the Priority Curve Algorithm – that integrates video segmentation, event detection and video summarization capabilities. Chapter 4 describes the theoretical foundation of our story creation framework and presents three heuristic algorithms for building stories, namely OptStory$^+$, GenStory and DynStory. The Text Attribute Extraction (TAE) algorithm and the named entity recognition algorithm (T-HMM) are presented in Chapter 5. An approach to extracting attribute values from relational and XML

data sources is also presented in this chapter. Chapters 6 and 7 describe experiments carried out to validate our approach to video summarization and story creation respectively. Conclusions and discussion about future developments are reported in Chapter 8.

1.2 Information Extraction

Many people and organizations need to access specific types of information in some domain of interest. For example, financial analysts may need to keep track of joint ventures and corporate mergers. Executive head hunters may want to monitor the corporate management changes and search for patterns in these changes. Information about these events is typically available from newspapers and various newswire services. Information retrieval systems can be used to sift through large volumes of data and find relevant documents[1] containing the information of interest. However, humans still have to analyze the documents to identify the desired information. The objective of Information Extraction (IE) is to address the need to collect the information (instead of documents containing the information) from large volumes of unrestricted text or any other kind of data.

The extracted information may be more valuable than the original data in several ways.

- While the documents returned by information retrieval systems have to be analyzed by humans, the database entries returned by information extraction systems can be processed by data processing algorithms.
- Information extraction also allows to answer queries that could be answered by information retrieval systems.

We now briefly compare information extraction with both information retrieval and text understanding. Of course some of the following considerations specifically apply to information extraction from text documents, but similar issues arise for other kinds of data.

[1] We often use the term *document* to denote a generic multimedia document – a text document, a video, etc.

Information extraction is a much more difficult task than information retrieval. In fact it involves:

- accurate recognition of the entities in documents: organizations, persons, locations, time, money, etc.;
- co-reference recognition;
- identification of relationships between entities and events;
- domain specific inference.

On the other hand, information extraction is a much easier and more tractable task than text understanding, because:

- its goal is narrowly focused on extracting particular types of information determined by a set of pre-defined extraction criteria;
- the inferences IE systems are required to make are much more restricted than a general natural language understanding system;
- due to its narrow focus, an IE system can largely ignore words or concepts that are outside its domain of interest.

Message Understanding Conference (MUC) is a DARPA sponsored conference in which participating IE systems are rigourously evaluated. Information extracted by the systems from blind test sets of text documents are compared and scored against information manually extracted by human analysts. Information extraction in the sense of the Message Understanding Conferences has been traditionally defined as the extraction of information from a text in the form of text strings and processed text strings which are placed into slots labeled to indicate the kind of information that they represent. So, for example, a slot labeled `NAME` would contain a name string taken directly out of the text or modified in some well-defined way, such as by deleting all but the person's surname. The input to information extraction is a set of texts, usually unclassified newswire articles, and the output is a set of filled slots. The set of filled slots may represent an entity with

its attributes, a relationship between two or more entities, or an event with various entities playing roles and/or being in certain relationships. Entities with their attributes are extracted in the *Template Element* task; relationships between two or more entities are extracted in the *Template Relation* task; and events with various entities playing roles and/or being in certain relationships are extracted in the *Scenario Template* task.

1.3 Information Summarization

Summarizing is the process of reducing a large volume of information to a short summary or abstract comprising only the most essential information items. Summarizing is frequent in everyday communication, but it is also a professional skill for journalists and scientific writers. Automated summarizing functions are needed by internet users who wish to exploit the information available without being overwhelmed. The primary application of summarization is thus that of summarizing the set of documents returned by an information retrieval system. Many other uses of summarization techniques are possible: information extraction, as opposed to document-retrieval; automatic generation of comparison charts; just-in-time knowledge acquisition; finding answers to specific questions; tools for information retrieval in multiple languages; biographical profiling.

The approach and the end-objective of summarization of documents explain the kind of summary that is generated. For example, it could be indicative of what a particular subject is about, or can be informative about specific details of the same. It can differ in being a "generalized summary" of a document as against a "query-specific summary". Summaries may be classified by any of the following criteria [43]:

Detail: indicative/informative

Granularity: specific events/overview

Technique: extraction/abstraction

Content: generalized/query-based

Approach: domain(genre) specific/independent

1.3.1 Evaluation Strategies and Metrics

Human judgment of the quality of a summary varies from person to person. For example, in a study conducted by Goldstein et al. [20], when a few people were asked to pick the most relevant sentences in a given document, there was very little overlap of the sentences picked by different persons. Also, human judgment usually does not find concurrence on the quality of a given summary. Hence it is difficult to quantify the quality of a summary. However, a few indirect measures may be adopted that indicate the usefulness and completeness of a summary [19, 25, 43, 49], such as:

1. Can a user answer all the questions by reading a summary, as he would by reading the entire document from which the summary was produced?
2. What is the compression ratio between the given document and its summary?
3. If it is a summary of multiple documents with temporal dimension, does it capture the correct temporal information?
4. Redundancy – is any information repeated in the summary?
5. Intelligibility – is the information in the summary easy to understand?
6. Cohesion – are the information items in the summary somehow related to each other?
7. Coherence – is the information in the summary organized according to some logic?
8. Readability (depends on cohesion/coherence/intelligibility)

The latter four qualities of summaries are usually difficult to measure. The last one specifically applies to text summaries while the other ones can be used to evaluate any kind of summary. A metric is said to be intrinsic or

extrinsic depending on whether the metric determines the quality based on the summary alone, or based on the usefulness of the summary in completing another task [19].

For example, the first one above is an extrinsic metric. An example of intrinsic measure is the cosine similarity of the summary to the document from which it is generated. This particular measure is not very useful, since it does not take into account the coverage of information or redundancy. With such a measure, a trivial way for improving the score would be to take the entire document as its summary. A metric that is commonly employed for extractive text summaries is that proposed by Edmundson [14]. Human judges hand-pick sentences from the documents to create manual-extractive summaries. Automatically generated summaries are then evaluated by computing the number of sentences common to the automatic and manually generated summaries. In information retrieval terms, these measures are called *precision* and *recall*. This method is currently the most used method for evaluating extractive summaries [19].

1.4 Conclusions

This chapter has introduced the general context of the thesis, presenting the basic concepts of Information Retrieval (IR) as opposed to Data Retrieval (DR). The goal of the work has then been introduced, pointing out that the thesis will focus on the extraction and summarization of information from large data repositories. Two major kinds of data repositories have been considered to this aim: digital video collections and the world wide web. The unifying theme of the application of the presented techniques to such different contexts is also described. Finally the basic concepts of both information extraction and summarization have been presented, focusing the discussion mainly on the text documents context, that is the main field where such techniques have been investigated. More attention will be devoted to video data in the next chapter.

Chapter 2

Related works

2.1 Video Databases

An enormous amount of video data is being generated nowadays all over the world. This requires efficient and effective mechanisms to store, access, and retrieve these data. But the technology developed to date to handle those issues is far from the level of maturity required. Video data, as we know, would contain image, audio, graphical and textual data.

The first problem is the efficient organization of raw video data available from various sources. There has to be proper consistency in data in the sense that data are to be stored in a standard format for access and retrieval. Then comes the issue of compressing the data to reduce the storage space required, since the data could be really voluminous. Also, various low-level features of video data have to be extracted – such as like shape, color, texture, and spatial relations – and stored efficiently for access.

The second problem is to find efficient access mechanisms. To achieve the goal of efficient access, suitable indexing techniques have to be adopted. Indexing based on text suffers from the problem of reliability as different individuals can analyze the same data from different perspectives. Also, this procedure is expensive and time-consuming. Nowadays, the most efficient way of accessing video data is through content-based retrieval, but this technique has the inherent problem of computer perception, as a computer lacks the basic capability available to a human being of identifying and segmenting a particular image.

The third problem is the issue of retrieval, where the input could come in the form of a sample image or text. The input has to be analyzed, available features have to be extracted and then similarity would have to be established with the images of the video data for selection and retrieval.

The fourth problem is the effective and efficient data transmission through networking, which is addressed through Video-on-Demand (VoD) and Quality of Service (QoS). Also, there is the issue of data security, i.e., data should not be accessible to or downloadable by unauthorized people. This is dealt with by watermarking technology which is very useful in protecting digital data such as audio, video, image, formatted documents, and three-dimensional objects. Then there are the issues of synchronization and timeliness, which are required to synchronize multiple resources like audio and video data.

2.1.1 Video Data Management

With the rapid advancement and development of multimedia technology during the last decade, the importance of managing video data efficiently has increased tremendously. To organize and store video data in a standard way, vast amounts of data are being converted to digital form. Because the volume of data is enormous, the management and manipulation of data have become difficult. To overcome these problems and to reduce the storage space, data need to be compressed. Most video clips are compressed into a smaller size using a compression standard such as JPEG or MPEG, which are variable-bit-rate (VBR) encoding algorithms. Data compression is an active research field, together with the transmission of video data over networking infrastructures. For instance, Video-on-Demand systems (VoD), which provide services to users according to their conveniences, have scalability and Quality of Service (QoS) issues because of the necessity to serve numerous requests for many different videos with the limited bandwidth of the communication links. To solve these problems, two procedures have been in operation, scheduled multicast and periodic broadcast. In the first one, a set of viewers arriving in close proximity of time will be collected and grouped together, whereas in the second one, the server uses multiple chan-

nels to cooperatively broadcast one video and each channel is responsible for broadcasting some portions of the video.

The abstraction of a long video is quite often of great use to the users in finding out whether it is suitable for viewing or not. It can provide users of digital libraries with fast, safe, and reliable access to video data. There are two ways available for video abstraction, namely, summary sequences, which give an overview of the contents and are useful for documentaries, and highlights, which contain the most interesting segments and are useful for movie trailers. The video abstraction can be achieved in three steps, namely, analyzing video to detect salient features, structures, patterns of visual information, audio and textual information; selecting meaningful clips based on detected features; and synthesizing selected video clips into the final form of the abstract [33]. Synchronization is a very important aspect of the design and implementation of distributed video systems. To guarantee Quality of service (QoS), both temporal and spatial synchronization related to the processing, transport, storage, retrieval, and presentation of sound, still images, and video data are needed [11].

With the enormous volume of digital information being generated in multimedia streams, results of queries are becoming very voluminous. As a result, the manual classification/annotation in topic hierarchies through text creates an information bottleneck, and it is becoming unsuitable for addressing users information needs. Creating and organizing a semantic description of the unstructured data is very important to achieve efficient discovery and access of video data. But automatic extraction of semantic meaning out of video data is proving difficult because of the gap existing between low-level features like *color*, *texture*, and *shape*, and high-level semantic descriptions like *table*, *chair*, *car*, *house*, and so on [75]. Luo et al. [39] have presented a scheme for object-based video analysis and interpretation based on automatic video object extraction, video object abstraction, and semantic event modeling. Although plenty of research works have been devoted to this problem to date, the gap still remains.

2.1.2 Video Information Retrieval

For efficient video information retrieval, video data has to be manipulated properly. The most common techniques applied to video retrieval are:

1. shot boundary detection, where a video stream is partitioned into various meaningful segments for efficient managing and accessing of video data;

2. key frames selection, where summarization of information in each shot is achieved through selection of a representative frame that depicts the various features contained within a particular shot;

3. low-level feature extraction from key frames, where color, texture, shape, and motion of objects are extracted for the purpose of indexing;

4. information retrieval, where a query is provided by the user and then, based on this, a search is carried out through the database to find matchings with the stored information [15].

Content-based image retrieval, which is essential for efficient video information retrieval, is emerging as an important research area with application to digital libraries and multimedia databases using low-level features like shape, color, texture, and spatial locations. Manjunath and Ma [44] focused on the image processing aspects and, in particular, on the use of texture information for browsing and retrieval of large image data. They also present an application for browsing large air photos.

Focusing has been given to the use of motion analysis to create visual representations of videos that may be useful for efficient browsing and indexing in contrast with traditional frame-oriented representations. Two major approaches for motion-based representations have been presented. The first approach demonstrated that dominant 2D and 3D motion techniques are useful for computing video mosaics through the computation of dominant scene motion and/or structure. However, this may not be adequate if object-level indexing and manipulation are to be accomplished efficiently.

The second approach presented addresses this issue through simultaneous estimation of an adequate number of simple 2D motion models. A unified view of the two approaches naturally follows from the multiple model approach: the dominant motion method becomes a particular case of the multiple motion method [56]. The problem of retrieving images from a large database is also addressed using an image as a query. The method is specifically aimed at databases that store images in JPEG format and works in the compressed domain to create index keys. A key is generated for each image in the database and is matched with the key generated for the query image. The keys are independent of the size of the image. Images that have similar keys are assumed to be similar, but the similarity has no semantic [59]. Another paper provides a state-of-the-art account of Visual Information Retrieval (VIR) systems and Content-Based Visual Information Retrieval (CBVIR) systems [45]. It provides directions for future research by discussing major concepts, system design issues, research prototypes, and currently available commercial solutions. Then a video-based face recognition system by support vector machines is presented. Marques and Furht [45] used stereovision to coarsely segment the face area from its background and then used a multiple-related template matching method to locate and track the face area in the video to generate face samples of that particular person.

2.1.3 Video Segmentation

The first step in indexing video databases (to facilitate efficient access) is to analyze the stored video streams. Video analysis can be classified into two stages [55]: shot boundary detection and key frames extraction. The purpose of the first stage is to partition a video stream into a set of meaningful and manageable segments, whereas the second stage aims to abstract each shot using one or more representative frames. In this section we will discuss the problem of shot boundary detection, while the problem of selecting key frames will be discussed in Section 2.1.4.

In general, successive frames in motion pictures bear great similarity among themselves, but this generalization is not true at the boundaries of

shots. A frame at a boundary point of a shot differs in background and content from its successive frame that belongs to the next shot. Two frames at a boundary point will differ significantly as a result of switching from one camera to another, and this is the basic principle upon which most automatic algorithms for detecting scene changes depend. Due to the huge amount of data contained in video streams, almost all of them are transmitted and stored in compressed format. While there are large numbers of algorithms for compressing digital video, the MPEG format [48] is the most famous one and the current international standard. In MPEG, spatial compression is achieved through the use of a Discrete Cosine Transform (DCT) based on an algorithm similar to the one used in the JPEG standard [52]. In this algorithm, each frame is divided into a number of blocks (8×8 pixel), then the DCT transformation is applied to each block. The produced coefficients are then quantized and entropy-encoded, a technique that achieves the actual compression of the data. On the other side, temporal compression is accomplished using a motion compensation technique that depends on the similarity between successive frames on video streams. Basically, this technique codes the first frame of a video stream (I frame) without reference to neighboring frames, while successive frames (P or B frames) are generally coded as differences to the reference frame(s). Considering the large amount of processing power required in the manipulation of raw digital video, it becomes a real advantage to work directly upon compressed data and avoid the need to decompress video streams before manipulating them.

Video data are rich sources of information and in order to model these data, the information content of the data has to be analyzed. As mentioned before, video analysis is divided into two stages. The first stage is the segmentation of the video sequence into a group of shots (shot boundary detection). Generally speaking, there are two trends in the literature to segment video data. The first one works in the uncompressed domain, while the other one works in the compressed domain. The first trend will be discussed first. Methods in the uncompressed domain can be broadly classified into five categories: template-matching, histogram-based, twin-comparison, block-based, and model-based techniques. In template-matching techniques

[26, 74], each pixel at the spatial location (i, j) in frame f_m is compared with the pixel at the same location in frame f_n , and a scene change is declared whenever the difference function exceeds a pre-specified threshold. Using this metric, it becomes difficult to distinguish between a small change in a large area and a large change in a small area. Therefore, template-matching techniques are sensitive to noise, object motion, and camera operations. One example of the use of histogram-based techniques is presented in [66], where the histogram of each video frame and a difference function S between f_n and f_m are computed. A cut is declared if S is greater than a threshold. That technique uses equation 2.1 to compute the difference function and declare a cut if the function is greater than a threshold.

$$S(f_m, f_n) = \sum_{i=1}^{N} |H(f_m, i) - H(F_n, i)| \tag{2.1}$$

The rationale behind histogram-based approaches is that two frames that exhibit minor changes in the background and object content will also show insignificant variations in their intensity/color distributions. In addition, histograms are invariant to image rotation and change slowly under the variations of viewing angle, scale, and occlusion [63]. Hence, this technique is less sensitive to camera operations and object motion compared to template matching based techniques. Another technique that is called twin comparison has been proposed by Zhang, Kankanhalli and Smoliar [74]. This technique uses two thresholds, one to detect cuts and the other to detect potential starting frames for gradual transitions. Unfortunately, this technique works upon uncompressed data and its inefficiency is the major disadvantage. A different trend to detect shot boundary is called block-based [28] and uses local attributes to reduce the effect of noise and camera flashes. In this trend, each frame f_m is partitioned into a set of r blocks and rather than comparing a pair of frames, every sub-frame in f_m is compared with the corresponding sub-frame in f_n. The similarity between f_n and f_m is then measured. The last shot boundary detection technique working upon uncompressed data is termed model-based segmentation [28], where different edit types, such as cuts, translates, wipes, fades, and dissolves are

modeled by mathematical functions. The essence here is not only identifying the transition but also the type of the transition.

On the other hand, methods for detecting shot boundaries that work in the compressed domain have been investigated. The main purpose of works in this trend is to increase efficiency. Again, we can roughly divide these methodologies into three categories. The first category [7, 35, 73] uses DCT coefficients of video-compression techniques (Motion JPEG, MPEG) in the frequency domain. These coefficients relate to the spatial domain, hence they can be used for scene change detection. In [7], shot boundary detection is performed by first extracting a set of features from the DC frame. These features are placed in a high-dimensional feature vector that is called the Generalized Trace (GT). The GT is then used in a binary regression tree to determine the probability that each frame is a shot boundary. Yeo and Liu [73] use the pixel differences of the luminance component of DC frames in MPEG sequences to detect shot boundaries. Lee et al. [35] derive binary edge maps from AC coefficients and measure edge orientation and strength using AC coefficients correlations, then match frames based on these features. The second category makes use of motion vectors. The idea is that motion vectors exhibit relatively continuous changes within a single camera shot, while this continuity is disrupted between frames across different shots. Zhang et al. [74] have proposed a technique for cut detection using motion vectors in MPEG videos. Their approach is based on counting the number of motion vectors M in predicted frames. In P-frames, M is the number of motion vectors, whereas in B-frames, M is the smaller of the counts of the forward and backward nonzero motion. Then, $M < T$ will be an effective indicator of a camera boundary before or after the B and P-frames, where T is a threshold value close to zero. The last category working into the compressed domain merges the above two trends and can be termed hybrid Motion/DCT. In these methods, motion information and the DCT coefficients of the luminance component are used to segment the video [46]. Other approaches that cannot be categorized into any of the above two classes are reviewed below. Vasconcelos and Lippman [69] have modeled the time duration between two shot boundaries using a Bayesian

model and the Weibull distribution, then they derived a variable threshold to detect shot boundaries. A knowledge-based approach is proposed by Meng, et al. [46], where anchorperson shots are found by examining intra-shot temporal variation of frames. In order to increase the robustness of the shot boundary detection, Hanjalic and Zhang [27] proposed the use of statistical model to detect scene changes. In summary, techniques that work upon uncompressed video data lack the necessary efficiency required for interactive processing. On the other hand, although the other techniques that deal directly with compressed data are more efficient, their lack of reliability is usually a common problem.

2.1.4 Video Summarization

The growing availability of large collections of video data creates a strong requirement for efficient tools to automatically summarize videos. Such tools automatically create a short version or subset of key-frames which contains as much information as possible as the original video. Summaries are important because they can provide rapidly users with some information about the content of a large video or set of videos. From a summary, the user should be able to evaluate if a video is interesting or not, for example if a documentary contains a certain topic, or a film takes partly place in certain location. In the corporate arena, there is growing need for video summarization. For instance, a company that uses video technology to secure its buildings may wish to summarize the surveillance videos so that only important events are included in the summary. An online education courseware seller may wish to create brief summaries of educational videos that focus on the most exciting snippets of the course in question. A sports organization such as the National Basketball Association in the USA or the International Federation of Football Association (FIFA) may wish to create summaries consisting of a few game highlights so that these summaries can be shown to potential customers who would subsequently buy the whole video. Military organizations may wish to summarize airborne surveillance video so that high priority events such as missile firings or suspicious vehicle activities can be detected. Large movie databases such as the Internet Movie

Database (IMDb) or movie sellers may wish to automatically create movie trailers.

Automatic summarization is subject to very active research, and several approaches have been proposed to define and identify what is the most important content in a video. However, most approaches currently have the limitation that evaluation is difficult, so that it is hard to judge the quality of a summary, or, when a performance measure is available, it is hard to understand what the interpretation of this measure is.

In general, a summary of a video must satisfy the following three principles first enunciated by Fayzullin et al. [17]. The video summary must contain high priority entities and events from the video. For example, a summary of a soccer game must show goals, spectacular goal attempts, as well as any other notable events such as the ejection of a player from the game, any fistfight and so on. In addition, the summary itself should exhibit reasonable degrees of continuity. Jitter must be absent. A third criterion is that the summary should be free of repetition. For example, it is common in soccer videos for the same goal to be replayed several times. It is not that easy to automatically detect that the same event is being shown over and over again. Even more difficult is to detect events with similar features, that can be considered repetitive. These three tenets, named the CPR (Continuity, Priority and no Repetition), form the basic core of all strong video summarization methods. The video summarization literature contains two broad classes of methods to summarize video. In the first class, that we call the *physical video property based* class, physical properties of the video stream are used to create a summary. The second class, that we call the *semantic video property based* class, tries to use semantic information about the content of the video in order to determine which frames or blocks of the video must be included.

2.1.4.1 Physical Video Property Based Methods

Most existing video summarization systems start with key frame extraction. In this technique, certain properties of the frames are used to identify them as key frames. For instance, one can consider frames with a lot of

motion or abrupt color changes as key frames (while avoiding frames with camera motion and special effects). The detected key frames can either be inserted into a summary as they are or used to segment the video. For example, video segmentation algorithms [37] may be used to "split" the video into homogeneous segments at key frames. One can then construct a summary by selecting a certain number of frames from each of these segments and concatenating them together.

The MoCA [36] system composes film previews by picking special events, such as zooming of actors, explosions, shots, etc. In other words, image processing algorithms are used to detect when selected objects or events occur within a video and then some of the frames in which these events occur end up in the summary. The authors propose an approach to the segmentation of video objects based on motion cues. Motion analysis is performed by estimating local orientations in the spatio-temporal domain using the three-dimensional structure tensor. These estimates are integrated into an active contour model, thus stopping the evolving curve when it reaches the moving objects boundaries. Segmented video objects are then classified by means of the contours of its appearances in successive video frames. The classification is performed by matching curvature features of the video object contour against a database containing preprocessed views of prototypical objects. Object recognition can be performed on different levels of abstraction.

Yahiaoui, Merialdo et al. [72] propose an automatic video summarization method in which they define and identify what is the most important content in a video by means of similarities and differences between videos. They identify in this way what is common, what is unique and how they differ. Comparison and classification representations of the video content is needed to pursuit this goal mainly because the same information often appears in slightly different forms in the different video segments. The elimination of redundant information across the set of videos in a TV series allows to provide concise image summaries. The proposed approach is based on the extraction of feature vectors from the sequence of frames and in particular the set of analyzed features are basically the combination of color histograms

applied on different portions of the frame. These vectors are used for the clustering procedure that produces classes of video frames with expected similar visual content. The frequency of occurrence of frames from each video within classes allows to compute the importance of the various classes. Once video frames have been clustered, the video could be described as sets of frame classes. The global summary is constructed with the representative images of video content selected in the set of most pertinent classes. They also suggest a new criterion to evaluate the quality of the summaries that have been created, through the maximization of an objective function.

Shao et al. [58] propose an approach to automatically summarize music videos, based on an analysis of both video and audio tracks. The musical track is separated from the visual track and is analyzed in order to evaluate the linear prediction coefficients, the zero crossing rates, and Mel-Frequency Cepstral Coefficients (MFCCs). Based on the computed features, and using an adaptive clustering method, they group the music frames and generate a structure of the music content. The results of the previous procedure are crucial for the generation of the summaries that are built in terms of the detected structure and in terms of a domain-based music knowledge. After the music summarization, they turn the raw video sequences into a structured data set in which boundaries of all camera shots are identified and visually similar shots are grouped together. Each cluster is then represented by the shot with the longest length. A video summary is generated by collecting all the representative shots of the clusters. The final step is the alignment operation that aims to partially align the image segments in the video summary with the associated music segments. The authors evaluated the quality of the summaries through a subjective user study and compared the results with those obtained by analyzing either audio or video track only. The subject enrolled in the experiments rated conciseness and coherence of the summaries on a 1 to 5 scale. Conciseness pertains to the terseness of the music video summary and how much of the music video captures the essence of the music video. Coherence instead pertains to the consistency and natural drift of the segments in the music video summary.

DeMenthon et al. [13] represent a changing vector of frame features

(such as overall macroblock luminance) with a multi-dimensional curve and applied a curve simplification algorithm to select key frames. In particular they extend the classic binary curve splitting algorithm, that recursively splits a curve into curve segments until these segments can be replaced by line segments. This replacement can occur if the distance from the curve of the segment is small. They show how to adapt the classic algorithm for splitting a curve of dimension N into curve segments of any dimension between 1 and N. The frames at the edges of the segments are used as key frames at different levels of detail. While this approach works well for the key frame detection, it does not consider the fact that certain events have higher priorities than others, and that continuity and repetition are important.

Ju et al. [32] propose another key frame selection approach that chooses frames based on motion and gesture estimation. Focusing the attention on the constrained domain of video sequences showing presentations in which the camera is focused on the speaker's slides, they estimate the global image motion between every two consecutive frames using a robust regression method. The extracted motion information is used to evaluate if a sequence of consecutive frames represents the same slide. The detected frame sequences are processed to extract the key frames used to represent the slides shown during the presentation. Computing a pixel difference between the key frames and the correspondent frames in the "stabilized" image sequences, they are able to detect the image regions containing potential gestures. Tracking these gestures by means of a deformable contour model and analyzing the shape and the motion over the time, they recognize the pointing gestures and recover the location on the slide to which the speaker is referring.

Zhou et al. [70] attempt to analyze video content, extract and cluster features to classify video semantically. Using an interactive decision-tree learning method, they define a set of if-then rules that can be easily applied to a set of low-level feature matching functions. In particular, the set of low level features used in the proposed framework are motion, color and edge features, that are automatically extracted from a video clip. Sample video

clips from the different semantic categories are used to train the classification system by means of the chosen low level features. The set of rules in the decision tree is defined as a combination of appropriate features and the relative thresholds that are automatically defined in the training process. They then apply their rule-based classification system to basketball videos.

Ma et al. [41] present a generic framework for video summarization based on estimated user attention. They construct video summaries modeling how user's attention is attracted by motion, objects, audio and language when he/she is watching a video program. For each frame in a video an attention value is computed and the result for a given video is an attention curve that allows to determine which frame or which sequence of frames is more likely to attract the user's attention. In this way, the optimal number of key frames in a video shot is determined by the number of wave crests on the attention curve. They then include in their summaries frames to which users pay a good deal of attention.

2.1.4.2 Semantic Video Property Class

This class of video summarization algorithms tries to perform elementary analysis of the video's semantic content and then to use this data, in conjunction with information about the user content preferences, to determine exactly which frames should be included into the summary and which frames should not.

The Video Skimming System [71] from Carnegie Mellon University finds key frames in documentaries and news-bulletins by detecting important words in the accompanying audio. The authors propose a method to extract the significant audio and video information and create a "skim" video which represents a very short synopsis of the original. The goal of this work is to show the utility of integrating language and image understanding techniques for video skimming by extraction of significant information, such as specific objects, audio keywords and relevant video structure. The resulting skim video is much shorter, where compaction is as high as 20:1, and yet retains the essential content of the original segment.

In contrast to the above works, the CPR system of Fayzullin et al. [17]

provides a robust framework within which an application developer can specify functions that measure the continuity $c(S)$, and the degree of repetition $r(S)$ in a video summary S. The developer can also specify functions to assess the priority of each video frame $p(f)$ (or more generally, if the video is broken up into a sequence of blocks, the priority $p(b)$ of each block b). The priority of the summary $p(S)$ can then be set to the sum of the priorities of all blocks in S. The developer can assign weights to each of the three functions and create an objective function $w_1 \cdot c(S) + w_2 \cdot p(S) - w_3 \cdot r(S)$. Given the maximal desired size k of a summary, the system tries to find a set S of video blocks such that the size of S is less than or equal to k blocks and such that the objective function $w_1 \cdot c(S) + w_2 \cdot p(S) - w_3 \cdot r(S)$ is maximized. Authors show that the problem of finding such an "optimal" summary is NP-complete and proceed to provide four algorithms. The first is an exact algorithm that takes exponential time but finds an S that does in fact maximize the value of the objective function. Other three algorithms may not return the best S but run in polynomial time and find summaries that are often as good as the ones found by the exact algorithm. Some examples of the continuity functions, repetition functions, and priority functions provided by Fayzullin et al. [17] include:

- Continuity can be measured by summing up the numbers of common objects shared by adjacent summary blocks, divided by the total numbers of objects in adjacent blocks. Thus, the more objects are shared between adjacent summary blocks, the more continuous the summary is. To measure continuity (or, rather, discontinuity) one can also sum up color histogram differences between adjacent blocks. The lower is this sum, the more continuous is the summary.

- Repetition can be computed as the ratio of the total number of objects occurring in the summary to the number of distinct objects. Alternatively, one can consider repetition to be inversely proportional to standard deviation of the color histogram in summary blocks. The less color changes occur in a summary, the more repetitive this summary is going to be.

- Priority of a block can be computed as the sum of user-defined priorities for objects occurring in the block or based on a set of rules that describe desired combinations of objects and events.

2.1.4.3 An Alternative Classification of Summarization Methods

Another way to classify the approaches to video summarization is to distinguish between *Reasoning Based* and *Measure Based* Summarization, as described in the following.

- *Reasoning Based Summarization.* Reasoning based approaches use logic or neural algorithms to detect certain combinations of events based on the information from different sources (audio, video, natural language). Examples of such approaches are video skims from the Informedia Project by Wactlar et al. [71] and movie trailers from the MoCA project by Lienhart et al. [36]. Sometimes multiple features of a video stream are employed simultaneously: the video analysis is combined with the audio analysis (speech, music, noise, etc.) and even with the textual information contained in closed captions. The heuristics used to identify video segments of interest with respect to all these features are encoded with logical rules or neural networks.

- *Measure Based Summarization.* Measure based approaches use various importance and similarity measures within the video to compute the relevance value of video segments or frames. Possible criteria include duration of segments, inter-segment similarities, and combination of temporal and positional measures. These approaches can be exemplified by the use of SVD (Singular Value Decomposition) by Gong and Liu [21], or the shot-importance measure by Uchihashi and Foote [67].

It is worth noting that most systems summarize video by key-frame extraction. For example, the *Video Skimming System* [71] finds key frames in documentaries and news-bulletins by detecting important words in the accompanying audio. Systems like *MoCA* [36] compose film previews by picking special events, such as zooming of actors, explosions, shots, etc.

In conclusion, despite the vast amount of work on video databases, and the existing work on summarizing video, there is no commonly accepted solution to the problem of automatically producing video summaries that take both content and user interest into account and scale to massive data applications.

2.2 Text Documents

2.2.1 Text Summarization

The goal of text summarization is to take one or more textual documents, extract content from them and present the most important content to the user in a concise way. Text summaries may be roughly classified into two main categories. They may be a collection of sentences carefully picked from the document or can be formed by synthesizing new sentences representing the information in the documents.

Sentence extraction methods for summarization normally work by scoring each sentence as a candidate to be part of summary, and then selecting the highest scoring subset of sentences. Some features that often affect the candidacy of a sentence for inclusion in a summary are listed in the following [14, 19].

Keyword occurrence Sentences containing keywords that are most often used in the document usually represent the topic of the document.

Title keyword Sentences containing words that appear in the title are also indicative of the topic of the document.

Location heuristic In newswire articles, the first sentence is often the most important sentence; in technical articles, the last couple of sentences in the abstract or those from conclusions are informative of the findings in the document.

Indicative phrases Sentences containing key phrases like "this report..." usually give an overview of the document.

Short-length cutoff Short sentences are usually not included in a summary.

Upper-case word feature Sentences containing acronyms or proper names are usually included in a summary.

Pronouns Pronouns such as "she, they, it" cannot be included in a summary unless they are expanded into corresponding nouns.

Redundancy in summary Anti-redundancy was not explicitly taken into account by earlier systems, but forms a part of most of the current summarizers. This score is computed dynamically as the sentences are included in the summary, to ensure that there is no repetitive information in the summary. The following are two examples of anti-redundancy scoring, when a new sentence is added to the summary:

- Scale down the scores of all the sentences not yet included in the summary by an amount proportional to their similarity to the summary generated so far [20, 53].
- Recompute the scores of all the remaining sentences after removing the words present in the summary from the query/centroid of the document [22].

Abstraction of documents by humans is complex to model as is any other information processing by humans. The abstracts differ from person to person, and usually vary in the style, language and details. The process of abstraction is complex to be formulated mathematically or logically [31]. In the last decade some systems have been developed that generate abstractions using the latest natural language processing tools. These systems extract phrases and lexical chains from the documents and fuse them together with generative tools to produce a summary (or abstraction). A relatively less complex approach is to create an extractive summary in which sentences from the original documents are selected and presented together as a summary.

Both extractive and abstractive methods give rise to some problems. In the first case:

- Extracted sentences usually tend to be longer than average. Due to this, part of the segments that are not essential for summary also get included, consuming space.
- Important or relevant information is usually spread across sentences, and extractive summaries cannot capture this (unless the summary is long enough to hold all those sentences).
- Conflicting information may not be presented accurately.

In the case of abstractive methods:

- It has been shown that users prefer extractive summaries instead of glossed-over abstractive summaries. This is because extractive summaries present the information as it is by the author, and would allow the users to read between the lines information.
- Sentence synthesis is not a well-developed field yet, and hence the machine generated automatic summaries would result in incoherence even within a sentence. In case of extractive summaries, incoherence occurs only at the border of two sentences.

2.2.2 Automatic Story Creation

A completely different approach to present information derived from textual data sources is the generation of narrative stories. According to the Oxford English Dictionary a story is *"a narrative, true or presumed to be true, relating to important events and celebrated persons of a more or less remote past; a historical relation or anecdote"*. From a computational point of view a story may be thought as a collection of known facts about a given entity – a person, an event, etc. – that may be delivered to the user in the form of an interactive presentation or rendered as natural language text.

There is a rich body of work on creating stories in the non-database literature, mainly in the Artificial Intelligence field. Many of the proposed approaches focus on specific audiences (e.g. children) and many authors aim at creating virtual environments where virtual characters interact, thus creating stories.

The Virtual Storyteller [65] is a framework for story creation by co-operating intelligent agents. In this framework, a collection of agents is responsible for the creation of different story levels: plot, narrative, and presentation. In the Virtual Storyteller, plots are automatically created based on the actions of autonomous characters whose plot creation is only constrained by general plot requirements. This approach lacks the disadvantages of pure character-based plot development, where the characters are fully autonomous, and of scripted approaches, where the plot content is predefined and the characters have no autonomy at all.

Szilas [64] proposes an approach to interactive drama where a "virtual narrator" chooses the actions to be performed in the story, based on several narrative criteria including consistency and progression ("how much the action makes the intrigue evolve, rather than stagnate" [64]). This is similar to the task of the director in the Virtual Storyteller. An important difference is that the approach of Szilas is not character-based. Instead, the candidate actions originate from a story grammar ('narrative logic' in Szilas' terms). The narrative logic ensures that the candidate actions fit into the general plot structure, and the virtual narrator judges their effect on the user.

The Teatrix system for virtual drama [42] is designed for collaborative story creation by children. In Teatrix, some of the story characters are controlled by the children using the system; the other characters are autonomous agents. There is also an omniscient director agent which can insert new items and characters into the story world, and which can control the characters' actions on behalf of the story coherence. The director cannot control the children's characters. The main difference with the Virtual Storyteller is that in Teatrix, the character and director agents function as aids in the children's story creation process, rather than creating the story by themselves.

In [68], Bers et al. present the SAGE system, a computational storytelling environment, that allowed young cardiac patients at the Boston's Children's Hospital to tell personal stories and create interactive characters, as a way of coping with cardiac illness and hospitalizations. In order to support children in creating their own characters, a visual programming

language was developed to design and program: (1) the scripts that are used by the storyteller, (2) the conversational structure or flow of the interaction, (3) the body behaviors of the interactive toy, which behaves as the pet assistant of the storyteller, and (4) the database of tales that are offered in response by the character. SAGE also has multimedia capabilities allowing children to record their own stories and to draw their own characters.

In conclusion, all the approaches discussed above have a very different goal than ours, because they either focus on having humans create a story, or provide some kind of action specification for the agents involved in the story and allow the story to develop having a non-deterministic outcome. Our goal is to collect all the available information about given entities from different data sources and create stories about them by selecting proper subsets of the known facts and possibly rendering those facts in English or any other language. Our approach to story creation is thus more similar to text summarization than automatic storytelling.

Part II

Theory

Chapter 3

Extraction and Summarization of Information from Video Databases

3.1 Introduction

The video summarization algorithm proposed in this thesis retains the core idea from [17] that the CPR criteria are important to achieve good quality summaries. However, the algorithm uses a completely different approach to the problem of finding good summaries fast. It completely eliminates the objective function upon which the previous algorithms were based, but captures the same intuitions in a compelling way. Section 3.2 introduces the CPR model, while Section 3.3 presents our Priority Curve Algorithm (PriCA). In particular Section 3.2.2 describes the three CPR-based algorithms presented in [16]. We describe the details of these algorithms for two reasons: i) we have implemented such algorithms in our video summarization framework and compared the performance of the Priority Curve Algorithm against them; ii) the algorithms we designed for automatically creating stories use similar approaches.

Information extraction issues are discussed as part of the description of the PriCA framework, that integrates all the video management functionalities discussed in Chapter 2 – namely video segmentation, video event

detection, information extraction and video summarization. Experiments on a prototype of the PriCA system are discussed in Chapter 6.

3.2 Video Summarization: the CPR Model

3.2.1 Formal Model

The model assumes that every video v has a *length* len_v describing the number of frames in the video – the frames in a video of length len_v are labeled $1, \ldots, \mathsf{len}_v$. In many cases, a sequence of contiguous frames might be considered as a *block* and then summaries might be created based on determining which blocks (rather than frames) to include in the summary. The advantage of this approach is that the number of blocks in a video is much smaller than the number of frames. The CPR model applies to both frames and blocks.

3.2.1.1 Summarization Content Specification

One of the most important issues of video summarization is to specify what should be included in the summary. In this section, the concept of *summary content specification* is presented.

Definition 3.1 (k-summary)**.** Suppose v is a video, $k \geq 0$ is an integer, and $S \subseteq \{1, \ldots, \mathsf{len}_v\}$ is a set of frames, whose cardinality is $card(S) \leq k$. Then S is called a *k-summary* of v.

In other words, a k-summary of a video is any set of k or fewer frames from the video. The desired content of a summary is specified using a logical language which contains a unary predicate called insum that takes a frame as input. If f is either a frame number or a variable ranging over frames, then $\mathsf{insum}(f)$ is called an insum-*atom*. When $\mathsf{insum}(f)$ is true, this means that frame f is *of interest* for inclusion in the summary. Of course, not all frames of interest may be included in the summary. Furthermore, some frames may be included in the final summary even if there is no interest in them because they may be required to ensure continuity of the produced summary.

The model assumes that the video database on top of which the summarization tools are built supports the following API functions:

- $\mathsf{findframe}(v, X)$: when X is either an object or an activity, this function returns the set of frames in the video v containing X.
- $\mathsf{findobj}(v, f)$: given a video v and a frame f, this returns the set of all the objects occurring in frame f of video v.
- $\mathsf{findact}(v, f)$: this is similar to the previous function except that it returns all activities occurring in frame f of video v.

Definition 3.2 (video call)**.** Suppose vc is a video database API function, and $t_1, \ldots, t_n$ are either arguments to vc (of the right type) or variables ranging over the values of the appropriate type. Then $vc(t_1, \ldots, t_n)$ is called a *video call*.

Definition 3.3 (video atom)**.** If $vc(t_1, \ldots, t_n)$ is a video call and X is either a constant or a variable of the same type as vc's output, then $(X \in vc(t_1, \ldots, t_n))$ is called a *video atom*. Likewise, if X, Y are either frames or variables ranging over frames and d is an integer, $\mathsf{after}(X, Y, d)$, $\mathsf{before}(X, Y, d)$, and $\mathsf{near}(X, Y, d)$ are video atoms.

Membership predicates are used to require the presence of a certain object or activity in a frame, or to bind a variable to objects or activities in a frame. For example, $X \in \mathsf{findact}(v, f)$ allows the variable X to be bound to any activity in frame f of video v. The before, after, and near predicates are used to ensure continuity by requiring that some frames occur near each other and in a certain order. Intuitively, a frame X satisfies $\mathsf{before}(X, Y, d)$ iff X occurs in the interval of frames starting at $Y - d$ and ending at Y. $\mathsf{after}(X, Y, d)$ is equivalent to $\mathsf{before}(Y, X, d)$ and $\mathsf{near}(X, Y, d)$ is equivalent to $\mathsf{after}(X, Y, d) \vee \mathsf{before}(X, Y, d)$.

Definition 3.4 (video condition)**.** If $va_1, \ldots, va_n$ are video atoms and E is a conjunction of equalities , then $\varrho = va_1 \wedge \ldots \wedge va_n \wedge E$ is a *video condition*.

For example, $X \in \mathsf{findobj}(v, f) \wedge X \in \mathsf{findobj}(v, f')$ is true for all objects X that appear *both* in frame f and frame f' of video v.

Definition 3.5 (summarization rule)**.** A *summarization rule* is an expression of the form $A \leftarrow \varrho \wedge A_1 \wedge \ldots \wedge A_m$, where ϱ is a video condition, and $A, A_1, \ldots, A_m$ are insum-atoms.

Intuitively, the above rule says that if ϱ is true and $A_1, \ldots, A_m$ are of interest for inclusion in a k-summary, then A is also of interest for inclusion in the k-summary.

A *video summary content specification* $\mathcal{V}$ is a finite set of summarization rules. Based on these rules, a finite set $\mathsf{Der}(\mathcal{V})$ of instantiated insum-atoms can be derived. These atoms are the ones deemed to be of interest for inclusion in a summary.

Definition 3.6 (valid k-summary)**.** Suppose $\mathcal{V}$ is a video summary content specification. A k-summary S is *valid* w.r.t. $\mathcal{V}$ iff $S \subseteq \mathsf{Der}(\mathcal{V})$.

The above definition says that for a k-summary to be valid, the inclusion of each frame must be justified by some rule in the summary specification.

3.2.1.2 Priority Specification

The most important consideration when computing a summary is how *appropriate* the summary content is to the user. To express this characteristic, the model introduces the concept of *priority*.

Definition 3.7 (priority function)**.** Suppose v is a video, $k \geq 0$ is an integer, and Σ is the set of k-summaries of v. A *priority function* w.r.t. v is a mapping $\mathsf{pri} : \Sigma \rightarrow \Re^+$.

Priority functions can be explicitly stated in many ways. The simplest way to define a priority function is to assign a priority $\mathsf{pri}(f)$ to each frame f and then compute the priority of a set S of frames as $\mathsf{pri}(S) = \Sigma_{f \in S} \mathsf{pri}(f)$.

3.2.1.3 Continuity Specification

Continuity is an important criterion to be taken into account when computing an appropriate summary. For example, consider a soccer match with

one goal. To show the goal effectively, a summary should probably include a segment of video both just before and just after the goal.

Definition 3.8 (continuity function)**.** Suppose v is a video, $k \geq 0$ is an integer, and Σ is the set of k-summaries of v. A *continuity function* w.r.t. v is a mapping $\chi : \Sigma \rightarrow \Re^+$.

Different summarization applications may use different instances of this general definition. For example, a notion of distance between frames can be used to define the continuity function. The more similar any two contiguous frames in the summary are, the more continuous the summary is.

3.2.1.4 Repetition Specification

The third important property of a summary is that it must not contain repetitive information. A video spanning 90 minutes will probably have at least a few key scenes. Summaries should probably show clips of each of these scenes, rather than just one. The goal of a repetition specification is to avoid repetitions.

Definition 3.9 (repetition function)**.** Suppose v is a video, $k \geq 0$ is an integer, and Σ is the set of k-summaries of v. A *repetition function* w.r.t. v is a mapping $\rho : \Sigma \rightarrow \Re^+$.

As in the case of continuity functions, repetition functions are very general in nature and can be defined in several ways. For example, a notion of distance between frames can be used to define the repetition function. The more distant frames in the summary are, the less repetitive the summary is. Alternatively the frequency of occurrence of objects/actions in the summary can be used to evaluate the degree of repetition: the more often the same object/action appears in the summary, the more repetitive the summary is.

3.2.1.5 Optimal Summary

It is easy to see that the continuity, priority, and repetition criteria may be in conflict with each other. For example, while choosing more adjacent frames improves continuity, it may also lead to increased repetition. In order

to define what an optimal summary is, we first need a way of evaluating a summary that allows the user to specify relative importance of these three criteria.

Definition 3.10 (summary evaluation)**.** Suppose $\mathcal{V}$ is a video summary content specification, $\mathcal{S}$ is the set of all the summarizations of a given video v, and $\alpha, \beta, \gamma \geq 0$ are integers. A *summary evaluation* is a function $eval : \mathcal{S} \rightarrow \Re$, of the form $eval(S) = \alpha \cdot \chi(S) + \beta \cdot \mathsf{pri}(S) - \gamma \cdot \rho(S)$. The summary evaluation *eval* is called *monotonic* iff whenever $S \subseteq S^{'}$, $eval(S) \leq eval(S^{'})$.

In the above definition, the constants α, β, γ denote the respective importance to be given to continuity, priority, and repetition criteria.

Definition 3.11 (k-summary computation problem)**.** Suppose $\mathcal{V}$ is a video summary content specification. A k-summary S is *optimal* w.r.t. $\mathcal{V}$ and a summary evaluation $eval(S)$ iff (i) it is valid w.r.t. $\mathcal{V}$ and (ii) there is no other valid k-summary $S^{'}$ w.r.t. $\mathcal{V}$ such that $eval(S) < eval(S^{'})$.

Theorem 3.1. Computing an optimal k-summary is NP-complete.

The proof is by a reduction of the knapsack problem [10] to the optimal k-summary computation problem.

3.2.2 Summarization Algorithms

This section describes the three CPR based algorithms presented in [16]. We first introduce a summarization algorithm called CPRopt which finds an optimal k-summary without making any assumptions about the priority, continuity, and repetition functions. However, as the optimal k-summary computation problem is NP-complete, this algorithm takes an exponential amount of time (w.r.t. the length of a video) which is clearly unacceptable. As a consequence, three alternative heuristic k-summarization algorithms have been designed and implemented. The first algorithm (CPRdyn) is based on dynamic programming, the second (CPRgen) is based on genetic programming, while the third algorithm called the Summary Extension Algorithm (SEA for short) is based on the *summary extension* concept.

3.2.2.1 The Optimal Summarization Algorithm

The CPRopt algorithm starts by computing the set of all insum-atoms in $\mathsf{Der}(\mathcal{V})$. This step can be executed in time linear to the number of frames in the video v. The algorithm then considers all subsets of $\mathsf{Der}(\mathcal{V})$ that contain k or less frames. Each of these subsets is a valid k-summary w.r.t. $\mathcal{V}$. The *eval*() function is then applied to these subsets and the one with the maximal *eval*() value is chosen. As the set of all subsets of size k need to be stored, the CPRopt has exponential space and time complexity – this is not a surprise as the problem of finding an optimal summary has been shown to be NP-complete.

Procedure CPRopt($\mathcal{V}$,k)
  $\mathcal{V}$ is a video summary content specification
  k is a desired summary length
begin
  $V := \emptyset$
  $\Delta := \emptyset$
  repeat
    // $A, A_1, \ldots, A_n$ are variable-free
    $V := V \cup \Delta$
    $\Delta := \emptyset$
    for each rule $A \leftarrow \varrho \wedge A_1 \wedge \ldots \wedge A_n$ in $\mathcal{V}$ **do**
      if $\varrho \wedge \{A_1, \ldots, A_n\} \subseteq V$ **then** $\Delta := \Delta \cup \{A\}$
    end for
  until $\Delta \setminus V = \emptyset$
  $\Sigma := \{S \mid S \subseteq V \wedge card(S) \leq k\}$
  $BestS := S \in \Sigma$ such that $\alpha \cdot \chi(S) + \beta \cdot \mathsf{pri}(S) - \gamma \cdot \rho(S)$ is maximal
  return $BestS$
end.

3.2.2.2 The CPRdyn Algorithm

The CPRdyn algorithm is based on *dynamic programming* [10]. The algorithm maintains a variable $v_{current}$ describing the best solution found so far. Initially, $v_{current}$ consists of k randomly chosen frames which are derivable from $\mathcal{V}$. The algorithm changes $v_{current}$ in each iteration by checking to see whether replacing a frame in $v_{current}$ by a frame which is not in $v_{current}$ will lead to a better summary. CPRdyn can be summarized as follows. The

space complexity of CPRdyn is linear in the number of frames, while the time complexity is exponential in k (which is much better than being exponential in the number of frames).

```
Procedure CPRdyn(𝒱,k)
    𝒱 is a video summary content specification
    k is a desired summary length
begin
    // Fill v_current with k randomly selected frames from Der(𝒱).
    v_current := {f_i | i ∈ [1,k] ∧ f_i ∈ Der(𝒱)}
    // Put the remaining frames into v^c.
    v^c := Der(𝒱) − v_current
    while v^c ≠ ∅
        subs := false
        r := 1
        while r ≤ k and subs = false
            // Build a new tentative solution by replacing f_r with a frame from v^c.
            v_tentative := (v_current \ {f_r}) ∪ {first(v^c)}
            if eval(v_current) < eval(v_tentative) then
                v_current := v_tentative
                add f_r to the tail of v^c
                subs := true
            else
                r := r + 1
            end if
        end while
        remove first(v^c) from v^c
    end while
    return v_current
end.
```

It is important to note that the CPRdyn algorithm will only consider summaries whose length is exactly k frames. While this may look like a serious limitation, it is not, as long as the $eval()$ function used in the algorithm is monotonic.

3.2.2.3 The CPRgen Algorithm

The CPRgen algorithm uses a genetic programming approach [10] to compute a k-summary. The algorithm starts by creating a random population of summaries and rates population members according to the value of $eval()$. A mutation operator is applied to a random population member and the member with the smallest $eval()$ value is eliminated. The algorithm

stops when the variation of the $eval()$ values within the population is less than a threshold δ or when the maximal number of iterations is reached. CPRgen's time complexity is $\mathrm{O}\left(\frac{\mathsf{len}_v{}^2}{k^2} \times N^2\right)$, while its space complexity is $\mathrm{O}\left(\frac{\mathsf{len}_v{}^2}{k}\right)$.

Procedure CPRgen($\mathcal{V}$,k,N,δ)
  $\mathcal{V}$ is a video summary content specification
  k is a desired summary length
  N is the desired number of iterations
  δ is the desired fitness threshold
begin
  $R := \lceil \frac{\mathsf{len}_v}{k} \rceil$
  compute an initial population of random solutions $V := (v_i)_{i=1...R}$ with frames from Der($\mathcal{V}$)
  for $j \in [1, N]$
    for $i \in [1, R]$
      $\overline{v}$:= a solution randomly chosen among the ones in V
      select a frame $\overline{f}$ from the video
      if $\overline{f} \in \overline{v}$ **then**
        choose another frame from the video
        insert the new frame in $\overline{v}$ eliminating $\overline{f}$
        add $\overline{v}$ to the population of solutions V
        eliminate from V the solution with the smallest fitness
        if $max_{v_1,v_2 \in V} |eval(v_1) - eval(v_2)| \leq \delta$ **then**
          return best solution from V
        end if
      end if
    end for
  end for
  return the best solution from V
end.

3.2.2.4 The Summary Extension Algorithm (SEA)

SEA extends the CPR model as follows:

- rules in a content specification $\mathcal{V}$ are assigned a weight;
- summaries are sets of *frame coverage* pairs (f, p), where $p \in [0, 1]$ quantifies how well the frame f satisfies the summary content specification $\mathcal{V}$;

- video calls return sets of frame coverage pairs, instead of just frames.

The CPR model is a special instance of the SEA model, in which all the specification rules have weight 1, and all the frame coverage pairs in a summary have $p = 1$. The Summary Extension Algorithm is based on a complex model. Furthermore, differently from the CPRgen and CPRdyn algorithms, we do not have any story creation algorithm based on similar principles. We thus omit a detailed description of SEA. We just mention that SEA is a greedy breadth-first search algorithm with the branching factor limited to N. It is based on the concept of *Valid Summary Extension*. Given a summary S a Valid Summary Extension of S w.r.t. a video summary content specification $\mathcal{V}$ – $VSE(\mathcal{V}, S)$ – is the set of frames f such that the value of inserting f into S is greater than 0. A function is defined to compute the value of inserting a frame into a summary. Such function takes into account the weight of the rule that justify the insertion of the frame in the summary and coverage of the frame itself. Frames in the valid summary extension are used for attempting to improve current solutions.

The space complexity of SEA is linear w.r.t. the number of new (i.e. not occurring in S) answers returned by $VSE()$ and exponential w.r.t. N. The time complexity is exponential w.r.t. both N and the number of new answers returned by $VSE()$.

3.3 The Priority Curve Algorithm (PriCA) for Video Summarization

In this section, we describe the PriCA (Priority Curve Algorithm) system for video summarization. The proposed summarization algorithm retains the core ideas on which the CPR model is based but uses a completely different approach to the problem of finding good summaries fast. We first provide an overview of PriCA components and then describe the components in detail. The term PriCA will be use to denote both the algorithm and the whole framework.

Given an input video v containing len_v frames, and an integer $0 < k \leq \mathsf{len}_v$ which indicates the maximal summary length that the user wishes to

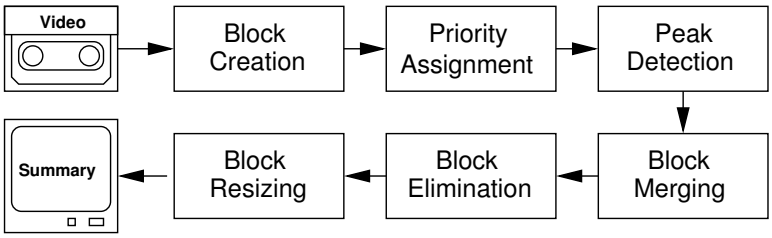


Figure 3.1: Architecture of the PriCA framework

see, PriCA finds a set of exactly k frames from the video. It attempts to ensure that these frames contain high priority objects and events in them (for example, when summarizing a soccer video, it might attempt to find frames containing goals, red cards, and other notable events from the game). It also attempts to ensure that the summary is not full of jitter by picking continuous portions of the video. Last, but not least, it attempts to eliminate repetition.

3.3.1 Overview of PriCA

PriCA is a complex system consisting of many parts. Fortunately, many of these parts can be implemented using standard image processing algorithms. Figure 3.1 shows the key components of PriCA. In the rest of this section, we describe the overall functions of the PriCA system. In particular, we will show how some of these components (such as block creation and priority assignment) can be built directly using standard image processing methods, while others (such as block merging, elimination, and resizing) require new contributions. Section 3.3.2 describes the details of the new components. In certain places, we will see that a mix of video and audio (and if available accompanying text) can be profitably used. We will use an application we have built for summarizing soccer video to illustrate our techniques. PriCA components are presented in the following.

Block creation The first part of PriCA is a *block creation* component that takes a video file as input and automatically splits the video file into blocks. This can be done in one of many ways. In the first way, the person interested in summarizing a collection of videos simply says

that each block is a certain number of frames (e.g. he/she may say that a block is a collection of 1800 frames, representing one minute of the video at a playback rate of 30 frames per second).

Alternatively, we may use *any* classical video segmentation algorithm to split the video into a set of blocks. Each block is a segment returned by the segmentation algorithm. The video is thus represented as a sequence of blocks, possibly of varying sizes. In our video summarization application, video segmentation is done by adopting the shot detection approach proposed by Boccignone et al. [5], that improves the formal model for video summarization introduced in [3].

A third method is to use audio streams associated with video in order to perform the desired segmentation. For example, every time we detect a new speaker, we may create a new segment. Thus for example, if person p_1 speaks during the first 100 frames of the video followed by person p_2 for another 40 frames followed by person p_3 for 200 frames, we would have three segments of video segmented by *audio*. There are numerous speaker recognition algorithms [9] in the literature.

A fourth method for video segmentation is to use the audio once again: the key difference this time is that we associate a vector with each audio-slice (which is a fixed duration of audio – usually a very small duration). This vector captures important audio properties such as pitch, intensity, loudness, etc. When two consecutive audio-slices have vectors whose Euclidean distance is below a distance threshold, we merge the slices together and evaluate the vector of the merged slice. We then check whether the merged slice can be merged with the next audio slice and so on. When this is not possible, we have a completed segment and the new slice forms the initial part of a new segment.

Priority assignment The segmented video is then fed into a *priority assignment* module which examines each block and assigns a priority to it. For example, the person summarizing a soccer video may specify priorities as follows: goal → 10, red card → 7, yellow card → 6, corner kick → 3, fight → 10, and so on. The priority assignment component

can also be implemented in many ways. In our soccer video summarization application, for example, the priority assignment is done by using image processing algorithms for events such as goal shot detection or red card detection [8]. Alternatively, in a military surveillance application that wants to assign high priority to gunshot detection in a video, an image analysis algorithm that identifies gunshots or explosions may be used to assign high priorities to such events. In a civilian surveillance application, high priority might indicate events denoting the entry or exit of a person into or from a monitored site. As a last resource, a video can also be annotated by a human being.

Peak detection Once the priorities have been assigned to each block, block IDs (increasing with time), together with their priorities, are shipped to the *peak detection* module. This module creates a graph whose x axis consists of block IDs, and whose y axis shows the priority of each block. The *Peaks()* algorithm we have developed can find *peaks* in this priority curve. Figure 3.2 shows an example graph and the peaks involved. Intuitively, a peak consists of a sequence of blocks containing high priority events.

Block merging The set of blocks thus identified in each peak is then shipped to the *block merging* module that examines these blocks and tries to determine if any of them can be merged. For example, it may turn out that there may be three blocks – the first containing the play just before a goal, the second containing the goal itself, while the third shows the post goal celebration. The block merging algorithm uses *rules* to determine conditions under which multiple contiguous blocks can be merged together into a new block (whose priority equals the sum of the priorities of the blocks being merged).

Block elimination The set of blocks produced after merging is then shipped to a *block elimination* module. This module eliminates blocks whose priority is too low. For example, it may turn out that 10 merged blocks are returned after the block merging algorithm and these 10 blocks have a total of 5000 frames. If we want a summary consisting

of just 3600 frames, we may want to re-examine whether a block of relatively low priority should be eliminated. For example, if we compute the average priority of the 10 blocks above to be 25 and the standard deviation to be 3, then we may want to eliminate all blocks with a priority under 16 (this is the classical statistical model which says that for a normal distribution, most objects in the distribution must occur within 3 standard deviations of the mean). Other statistical rules can also be used here.

Block resizing The next component is the *block resizing component*. For example, even after eliminating "low value" blocks above, it may turn out that we still have 8 blocks containing a total of 4500 frames. This must somehow be reduced to 3600. The block resizing component eliminates frames from the blocks in proportion to the priorities of the blocks involved. For example, let us say that the total priorities of all the 8 blocks is 800, and that a particular block (containing 1500 frames) has priority 200. Thus, this block accounts for a quarter of the entire priority of the 4500 frame sample. As a consequence, the block should be allowed to contribute a quarter of the 3600 frames allowed in the summary, i.e. 900 frames should be chosen from this block for inclusion in the summary. Our block resizing algorithm will show *how* to select the best 900 frames from the 1500 frame block. The block resizing component sequences the resized blocks together to create the final summary.

3.3.2 Details of PriCA Components

In this section, we describe how to implement each module of the PriCA algorithm. Some of these modules, such as *block creation* and *priority assignment*, can be implemented using classical image processing and recognition algorithms, while others require original algorithms.

3.3.2.1 Block Creation Module

The block creation module can be implemented using any number of methods including classical segmentation algorithms which have been ex-

tensively studied in the literature, as discussed in Chapter 2. Hence, we are only going to describe our implementation of this module.

We have been working with soccer videos and have chosen to segment them by shot boundaries. In spite of the long research history, the problem of the shot boundary detection has not been completely solved yet. Sports video is arguably one of the most challenging domains for robust shot boundary detection due to

1. strong color correlation between shots, due to a single dominant background color (soccer field);
2. large camera and object motions;
3. cuts and gradual transitions (such as fades and dissolves) often present in sports video clips.

To detect shots in soccer videos, we have adapted an algorithm from [5], based on the observation that frames belonging to the same shot are more similar than frames from different shots. The heart of the algorithm is based on the biological mechanisms of *visual attention*. The term "attention" captures the cognitive functions that are responsible for filtering out unwanted information and bringing to consciousness what is relevant for the observer [29]. Boccignone et al. [5] propose a novel similarity function based on a combination of the scanpath structure (which describes how the eye focuses on different parts of an image – the so called Focuses of Attention, FOAs for short) and color, texture, and shape features of FOAs of each single frame.

The proposed scheme allows the detection of both cuts and dissolves between shots using a single technique, rather than a set of dedicated methods. Also, it is well grounded in visual perception theories and allows us to overcome usual shortcomings of many other techniques proposed so far. Further, the proposed focus of attention representation is robust with respect to smooth view changes.

3.3.2.2 Priority Assignment Module

The priority assignment module can be implemented using classical object/event recognition algorithms [4] which have been extensively studied in the literature. Hence, we will limit the following discussion to our implementation of this module.

Automatic event detection in soccer videos is an open problem actively addressed by several sports institutions. In this paper, we are interested in a simple detection of such events as "goal", "celebration", "yellow card", and "red card". To detect these events, we have used the strategy proposed in [8]. These algorithms aim to aggregate the shots, extracted in the preliminary shot detection stage, in order to form scenes characterized by the presence of goal actions and match highlights. Each frame of the extracted shot is analyzed using a feed-forward neural network with a back-propagation algorithm. The network has been trained considering geometric features, colors and texture, the presence of players (foreground, portrait or whole figure), detection of the ball, of the field and of the goal-mouth, detection of red and yellow cards. Particular attention is given to the audio analysis in order to improve the detection or to detect relevant events such as goals and celebration: in fact, it is not difficult to believe that in a scene containing a "celebration" following a goal, the audio signal is inevitably higher than in normal actions of the game. In particular we have adopted a simple but efficient RMS calculus. Eventually, the *goal* event has been characterized through simple reasoning about the conjunctive simultaneous presence of several detected events, such as the presence of the player (foreground, portrait or whole figure), of the ball, of the field and of the goal area and of the celebration.

3.3.2.3 Peak Identification Module

Let $b_1, \ldots, b_n$ be the blocks in the video (e.g. after the segmentation process). Let p_i denote the priority of block b_i.

Definition 3.12 ((r,s)-peak)**.** Suppose $b_1, \ldots, b_n$ is a video, $r \in (0, n/2]$ is an integer and $s \in [0,1]$ is a real number. Blocks $b_j, \ldots, b_{j+r}$ are said to be

an (r, s)-*peak* iff

$$\frac{\sum_{j \leq i \leq j+r} p_i}{\sum_{j-\frac{r}{2} \leq i \leq j+\frac{3r}{2}} p_i} \geq s$$

Suppose we wish to check if a sequence of r blocks $S_1 = b_j, \ldots, b_{j+r}$, constitutes a peak. The above definition looks at $\frac{r}{2}$ blocks before the sequence as well as $\frac{r}{2}$ blocks after the sequence, i.e. the sequence $S_2 = b_{j-\frac{r}{2}}, \ldots, b_j, \ldots, b_{j+r}, \ldots, b_{j+\frac{3r}{2}}$ is considered. This latter sequence S_2 is of width $2r$. We sum up the priorities of all blocks in S_2 – let us call this sum s_2. Likewise, we sum up the priorities of all blocks in S_1 and call this priority s_1. Clearly, $s_1 \leq s_2$. If $\frac{s_1}{s_2}$ exceeds or equals s, then we decide that the contribution of the priorities of the peaks in S_1 is much larger than that in S_2 and so S_1 constitutes a peak.

It is important to note that r and s must be chosen by the application developer[1]. We discuss four cases below:

- r, s **both large:** When r and s are both large, we find segments that are big (i.e. consist of a large number of blocks) where the priority is very high throughout this large segment. The disadvantage of having a large r is that "local" peaks in a sequence of r blocks may get missed.

- r **large,** s **small:** This option is not a very good one. If one chooses s to be small, the number of sequences of blocks recognized as (r, s)-peaks will be very large.

- r **small,** s **large:** In contrast to the first case above, peaks in this case will consist of a relatively small number of blocks, but the peaks are unlikely to contain further subpeaks within them (which can happen in the first case above). When r is small, one may be tempted to infer that lots of (r, s)-peaks will be found – however, this really depends on s. When s is large, it seems unlikely that lots of (r, s)-peaks will be found unless the priority of blocks is more or less even throughout the video.

[1]Sometimes we use the term *application developer* to denote the person who is in charge of setting up the prototype for a particular application. We do not expect the end user to tune certain parameters.

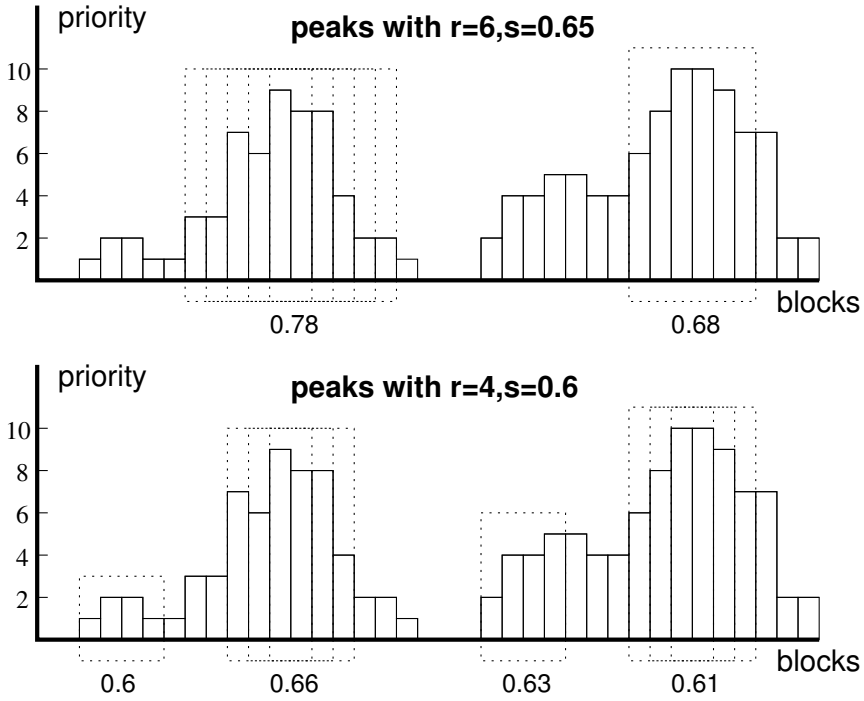


Figure 3.2: Example of peaks in the priority function

- r, s **both small:** We do not recommend this option – small values of both r and s are likely to produce a vast number of (r, s)-peaks.

Our recommendation is to pick r small (but not too small) and s large. Figure 3.2 shows two examples of peaks corresponding to (r, s) values of $(6, 0.65)$ and $(4, 0.6)$ respectively. Dotted rectangles signify peaks, with s-values shown for the most significant peaks. As seen from the figure, peaks often occur in clusters. While the upper graph corresponds to wide ($r = 6$) peaks, parameters in the lower graph allow for narrower ($r = 4$) and slightly lower ($s = 0.6$ as opposed to $s = 0.65$) peaks. This is consistent with our discussion above that when s drops in value, the number of peaks goes up. As result, the lower graph contains more peaks and smaller clusters.

Here is a simple algorithm that, given a sequence of video blocks and r, s values, will find all blocks that belong to (r, s)-peaks:

Algorithm Peaks(v,r,s)

v is a sequence of $card(v)$ block-priority pairs

```
    r is the peak width
    s is the peak height
begin
    Res := ∅
    for each j ∈ [r, card(v) − r] do
        center := 0
        total := 0
        for each ⟨ b_i, p_i ⟩ ∈ v such that i ∈ (j − r, j + r] do
            total := total + p_i
        end for
        for each ⟨ b_i, p_i ⟩ ∈ v such that i ∈ (j − r/2, j + r/2] do
            center := center + p_i
        end for
        if center/total ≥ s then
            Res := Res ∪ {⟨ b_i, p_i ⟩ ∈ v | i ∈ (j − r/2, j + r/2]}
        end if
    end for
    return Res
end.
```

The $Peaks()$ algorithm slides a $2r$-wide window along a sequence of blocks, computing the total sum of block priorities in that window ($total$). It then computes the sum of block priorities in a narrower r-wide window in the middle of the $2r$-wide window ($center$). When the ratio of these two sums $\frac{center}{total}$ exceeds the threshold s, all blocks in the r-wide window are picked as a *peak*.

Example 3.1. Consider the very small fragment shown in Figure 3.3. At some time, the $Peaks()$ algorithm will focus its window of length $2r$ on the segment from $j - \frac{r}{2}$ to $j + \frac{3r}{2}$ shown in the figure. It will compute the sum of the priorities of the blocks in the entire window of length $2r$ (which is $5 + 41 + 8 = 54$) as well as the sum of the priorities of the window of length r in the center of the window of length $2r$ – the priority there is 41. As a consequence, the ratio of these is $\frac{41}{54} = 0.76$. If 0.76 exceeds the threshold s that has been chosen, then the sequence of blocks from j to $j+r$ is considered a peak.

Example 3.2. Consider the 35 block sequence shown in Figure 3.4. We now describe how the $Peaks()$ algorithm finds the peaks in this figure. Suppose $r = 6$ and $s = 0.8$.

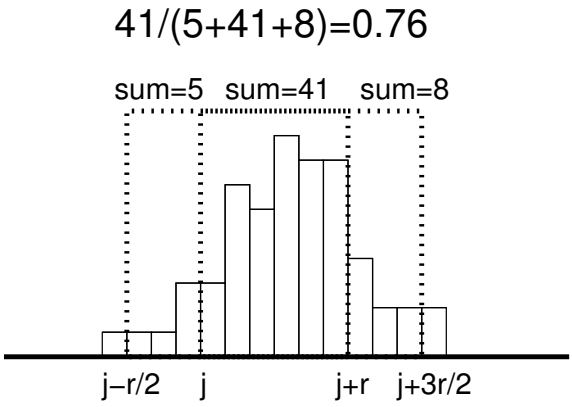


Figure 3.3: Peaks() algorithm analyzing a peak

- **Window from** $1-12$**:** We initially start by looking at the first 12 blocks. The sum of the priorities of these 12 blocks is 59. If we look at the window of size 6 centered at the middle of the first 12 blocks (these are the blocks $4-9$), the sum of the priorities is 36. The ratio, $\frac{36}{59}$ is below $s = 0.8$.

- **Window from** $2-13$**:** We now slide the window of length $2r$ one place to the right. At this time, the sum of the 12 block window is 60 and the sum of the 6 center blocks (blocks $5-10$) is 44. The ratio is therefore $\frac{44}{60}$ which is below $s = 0.8$.

- **Window from** $3-14$**:** We now slide the window of length $2r$ one place to the right. At this time, the sum of the 12 block window is 58 and the sum of the 6 center blocks (blocks $6-11$) is 48. The ratio is therefore $\frac{48}{58}$ which is greater than s. Therefore, blocks $6-11$ are returned as a peak.

The algorithm continues in a similar fashion, finding peaks in blocks $18-23$ ($r = \frac{25}{29}$) and $28-33$ ($r = \frac{39}{48}$).

Example 3.3. Let us now see what happens with Example 3.2 when the threshold s is dropped to 0.4.

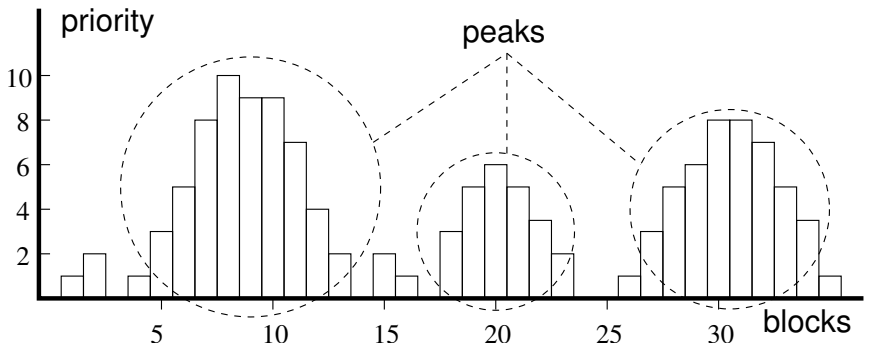


Figure 3.4: Result of running Peaks() Algorithm

- **Window from** $1-12$**:** The ratio, $\frac{36}{59}$, for the first 12 blocks is above $s = 0.4$. Hence, blocks $4-9$ are returned as a peak.
- **Window from** $2-13$**:** We now slide the window of length $2r$ one place to the right. The ratio $\frac{44}{60}$ also indicates a peak, as it is bigger than $s = 0.4$. Thus, we also return blocks $5-10$.
- **Window from** $3-14$**:** We now slide the window of length $2r$ one place to the right. Again, the ratio $\frac{48}{58}$ indicates a peak, returned as $6-11$.

As one can see, the peaks detected by the algorithm in this case are much wider than the peaks in Example 3.2. In fact, most blocks shown in Figure 3.4 will be selected as peaks. This is a typical result of lowering the detection threshold s.

The $Peaks()$ algorithm has complexity of $O(r \cdot card(v))$ – hence it is linear with respect to the number of input blocks.

Note that the performance of the $Peaks()$ algorithm can be improved by avoiding computation of *center* and *total* iteratively in each iteration of the outer loop. After the first iteration of the outer loop, these values can be updated in constant time. Though including these optimizations complicates the algorithm somewhat, it is well worth doing – as shown in the algorithm $OptPeaks()$ below.

Algorithm OptPeaks(v,r,s)
  v is a sequence of block-priority pairs
  r is the peak width
  s is the peak height
begin
  $Res := \emptyset$
  $center := 0$
  $total := 0$
  for each $\langle b_i, p_i \rangle \in v$ such that $i \in [1, 2 \cdot r]$ **do**
    $total := total + p_i$
  end for
  for each $\langle b_i, p_i \rangle \in v$ such that $i \in \left(\frac{r}{2}, \frac{3r}{2}\right]$ **do**
    $center := center + p_i$
  end for
  if $\frac{center}{total} \geq s$ **then**
    $Res := Res \cup \{\langle b_i, p_i \rangle \in v \mid i \in \left(\frac{r}{2}, \frac{3r}{2}\right]\}$
  end if
  for each $j \in [2 \cdot r + 1, card(v)]$ **do**
    $total := total + p_j - p_{j-2 \cdot r}$
    $center := center + p_{j-\frac{r}{2}} - p_{j-\frac{3r}{2}}$
    if $\frac{center}{total} \geq s$ **then**
      $Res := Res \cup \{\langle b_i, p_i \rangle \in v \mid i \in \left(j - \frac{3r}{2}, j - \frac{r}{2}\right]\}$
    end if
  end for
  return Res
end.

The $OptPeaks()$ algorithm has complexity of $O(card(v))$ – hence while its complexity is linear with respect to the number of input blocks, it does not depend on the window size r as in the case of $Peaks()$'s complexity.

3.3.2.4 Block Merging Module

The peak identification algorithm discards blocks that are not (r, s)-peaks for the selected r, s values. Let $Peaks(v, r, s)$ be the set of all blocks from the original video that contain peaks.

Consider the set $\{(b_i, b_{i+1}) \,|\, b_i, b_{i+1} \in Peaks(v, r, s)\}$ of all pairs of blocks that are adjacent to each other. In general when adjacent blocks are peaks, there is some possibility that they may describe the same event. For example, in our soccer video summarization application, we may have one block (peak block) that describes a goal event. The camera may have been switched to another block that also describes the same goal – however, the

segmentation algorithm creating the blocks may treat these events as different events when in fact they describe the same event. This may be due to the fact that the segmentation algorithm is based on shot detection algorithms or other similar algorithms that solely use visual features to detect segments. The main goal of the block merging module is to merge adjacent blocks that may be very similar, so that repeating blocks can be treated as a single block in the later processing steps (such as resizing).

A *block similarity function* is a function *sim* that takes two blocks as input and returns a non negative real number as output. The bigger the number returned, the more similar the blocks are considered to be. There are many ways in which we could implement block similarity functions. Here are a few examples:

1. sim_{idiff}. One possibility is that we could use any classical image differencing algorithm $idiff$ [23] to return the similarity between two frames and we could set the similarity between the two blocks to be the similarity between the two most similar frames, drawn from each block.

2. sim_{random_idiff}. An alternative is that the similarity algorithm randomly selects k frames from each block (where k is set to some small number) and then finds two frames, one from each block, that are maximally similar. This is a variant of the above algorithm.

3. sim_{audio}. We could use an audio detection algorithm that extracts the audio associated with each frame and then uses audio features such as pitch, loudness, etc. to associate a vector with the entire block. The similarity between the two blocks is some function that is inversely proportional to the Euclidean (or other) distance measure between the two blocks.

4. sim_{text}. In the event that the videos in question have an accompanying text transcript, we could identify the text blurb associated with each of the two blocks and set the similarity of the two blocks to be equal to the similarity between the two text transcripts using any classical

method to evaluate similarities between text documents.

5. $sim_{keywords}$. Suppose we have a given set $\mathcal{K}$ of keywords of interest. For each block b, we associate a vector $\vec{b}$ of length $|\mathcal{K}|$ – the i'th entry in the vector denotes the frequency of occurrences of the i'th keyword (or its synonyms). We then merge two adjacent blocks b_1, b_2 iff the Euclidean distance between their associated vectors is below a given threshold. Note that the keyword vector can be replaced by other vectors traditionally used in information retrieval [62].

6. sim_{vec}. As is often common in image processing, we could associate a color and/or texture histogram with each block and return the similarities between the histograms using root mean squared distance or the L_1 metric [63].

To define the block merging process, let us remark that $Peaks(v, r, s)$ returns a set of block-priority pairs of the form $\langle\, b_i, p_i \,\rangle$ – as opposed to a set of blocks – and assume that adjacent blocks can be concatenated with the $\oplus$ operator. The block merging algorithm then takes as input, *any* block similarity function between blocks (those listed above are just a few examples, many others are also possible), together with a set of block-priority pairs, and returns a new set of merged blocks-priority pairs, as follows.

Algorithm Merge(v,$sim()$,d)
  v is a sequence of block-priority pairs
  $sim()$ is a similarity function on blocks
  d is the merging threshold
begin
  $Res := \emptyset$
  $B :=$ first block-priority pair $\langle\, b_1, p_1 \,\rangle \in v$
  for each $\langle\, b_j, p_j \,\rangle, \langle\, b_{j+1}, p_{j+1} \,\rangle \in v$ **do**
    if $sim(b_j, b_{j+1}) \geq d$ **then**
      $B := \langle\, B.b \oplus b_{j+1}, B.p + p_{j+1} \,\rangle$
    else
      add B to the tail of Res
      $B := \langle\, b_{j+1}, p_{j+1} \,\rangle$
    end
  end for
  add B to the tail of Res

return Res
end.

The $Merge()$ algorithm considers all pairs of blocks b_j, b_{j+1}, concatenating them together into a bigger block $B.b$, as long as $sim(b_j, b_{j+1})$ value stays above the threshold d. The priority $B.p$ of the newly merged block is computed as the sum of individual priorities of its parts.

Example 3.4. Let us continue with Example 3.2. The peaks identified are $6-11$, $18-23$, and $28-33$. The $Merge()$ algorithm merges these blocks as follows. For the sake of this example, let us define $sim(b_1, b_2) = 1 - \frac{|p_1-p_2|}{p_1+p_2}$, where p_1 and p_2 are priorities of blocks b_1 and b_2 respectively, and set the threshold $d = 0.9$. Given these parameters, blocks $8-10$ will be merged into a single new block with $p = 28$ and so will blocks $19-21$ ($p = 16$), $28-29$ ($p = 11$), and $30-32$ ($p = 33$). Thus, the total number of blocks decreases from 18 to 11 after merging.

The $Merge()$ algorithm has linear complexity with respect to the number of blocks in its input.

3.3.2.5 Block Elimination Module

Suppose $\mathcal{B}$ is the set of blocks from the original video after the block merging step has been applied to the set of blocks in $Peaks(v, r, s)$. In the block elimination module, we would like to remove from this set all blocks whose priorities are less than a certain threshold. In addition, we would like to consider eliminating blocks that are repetitive. For example, in our soccer application, we may have replays of a goal long after the goal was scored. Both the original goal and the later replay may have high priorities, but our summary should probably not include both of them.

The block elimination module may use a similarity function similar to those used in the block merging module to first identify similar blocks. Any similarity function sim designated by the application developer may be used here in the following manner.

We say that blocks b_1, b_2 *are equivalent, denoted* $b_1 \sim b_2$*, w.r.t.* sim iff $sim(b_1, b_2) \geq t$ for some threshold t. In other words, the blocks are

considered similar if the similarity function assigns them a similarity score that exceeds a given threshold.

One may think that the $\sim$ is an equivalence relation, but in general it may not be so. The reason for this is that the fact that the similarity between b_1 and b_2 exceeds threshold t and the fact that the similarity between b_2 and b_3 exceeds threshold t does not imply that the the similarity between b_1 and b_3 exceeds threshold t. As a consequence, we need to define the concept of a cluster w.r.t. a threshold t.

Definition 3.13 (t-cluster)**.** Suppose $\mathcal{B}$ is a set of blocks and *sim* is a similarity function. A *t-cluster* of $\mathcal{B}$ is any set $B \subseteq \mathcal{B}$ such that for all $b_1, b_2 \in B$, $sim(b_1, b_2) \geq t$.

In other words, a t-cluster consists of blocks that are highly similar to each other (i.e. have similarity level t or more according to the selected similarity function).

Given a set of blocks $\mathcal{B}$ returned by the block merging module, our goal is to split $\mathcal{B}$ into t-clusters. The key idea is that when a cluster has lots of blocks, it may be possible to just retain one of those blocks rather than keeping all of them as these blocks are all deemed to be similar. This introduces the concept of a t-partition given below.

Definition 3.14 (t-partition)**.** A *t-partition* of $\mathcal{B}$ is a set $\mathcal{B}_1, \ldots, \mathcal{B}_r$ where $\mathcal{B}_1 \cup \ldots, \cup \mathcal{B}_r = \mathcal{B}$ and $\mathcal{B}_i \cap \mathcal{B}_j = \emptyset$ and for all $1 \leq i \leq r$, $\mathcal{B}_i$ is a t-cluster.

Intuitively, a t-partition splits a set of blocks into clusters $\mathcal{B}_i$ such that each cluster is a t-cluster. It is easy to see that a valid t-partition of $\mathcal{B}$ simply consists of the set $\{ \{b\} \mid b \in \mathcal{B} \}$. In other words, if we simply split $\mathcal{B}$ by taking each element of $\mathcal{B}$ and making it into a singleton set cluster, we would have a valid t-partition. Clearly, this defeats our intent to group multiple blocks together. To ensure this, we need to define the concept of a *maximal t-partition*.

Definition 3.15 (maximal t-partition)**.** Suppose $\mathcal{B}_1, \ldots, \mathcal{B}_r$ is a t-partition of $\mathcal{B}$. We say that $\mathcal{B}_1, \ldots, \mathcal{B}_r$ is a *maximal t-partition* iff there are no $1 \leq i < j \leq r$ such that $\mathcal{B}_i \cup \mathcal{B}_j$ is also a t-cluster.

A maximal t-partition forces clusters that can possibly be merged to in fact be merged.

Our goal now is to first find a maximal t-cluster of $\mathcal{B}$ where $\mathcal{B}$ is the set of blocks returned by the block merging step. This is relatively easy: suppose $\mathcal{B} = \{b_1, \ldots, b_m\}$. We first put b_1 into a cluster by itself and then we check if b_2 has similarity level t or more with this cluster. If so, we add b_2 into the cluster – if not, we go on to b_3. This process is repeated till we add as many blocks to the cluster associated with b_1 as possible while ensuring that this cluster is a t-cluster. We then continue to repeat this process to create clusters with the blocks not in this initial cluster. Finally, the process is completed by selecting one block from each cluster and eliminating all other blocks, as shown in the following algorithm.

```
Algorithm Cluster(v,t)
    v is a sequence of block-priority pairs
    t is the clustering threshold
begin
    Res := ∅
    while v ≠ ∅ do
        B := ∅
        for each b ∈ v do
            if min_{b'∈B} sim(b, b') ≥ t then
                B := B ∪ {b}
                v := v \ {b}
            end if
        end for
        Res := Res ∪ {highest priority block from B}
    end while
    return Res
end.
```

It is easy to see that the complexity of the $Cluster()$ algorithm above is proportional to the number of blocks in the input to the algorithm – as such $Cluster()$ runs very fast indeed.

Once the clusters are identified, our block elimination module picks one element from each cluster. In a sense, this one element is a representative of that cluster. Selecting an element from a given cluster can be done in many possible ways:

1. *Random selection.* One option is to randomly choose any member of the cluster.

2. *Prioritized selection.* Another option is to select a member of the cluster that has maximal priority. This strategy has the advantage that even though all blocks in a cluster are similar, one may have slightly higher priority than another and we might as well choose it.

3. *Prioritized ratio selection.* Another option is to select a member of a cluster that has the maximal priority vs. size ratio. As blocks can have varying sizes, it may turn out that the block with the largest priority is also pretty large – in this case, it may be better to choose a smaller block with fairly high priority as this smaller block contributes less frames towards the overall summary, thus allowing blocks from other clusters to be utilized.

4. *Size-oriented selection.* Another option is to merely be parsimonious and say that the block with the smallest size in each cluster will be selected.

Other strategies are also possible: rather than selecting one block from each cluster, we may be able to select multiple blocks. It could well be the case that a given $\mathcal{B}$ has only five clusters. In this case, any of the mechanisms to select a single block from each cluster yields a total of five blocks and it is conceivable that these five blocks don't jointly account for the total summary length. Consider any strategy to select blocks from a cluster, and suppose we have split $\mathcal{B}$ into clusters $\mathcal{B}_1, \ldots, \mathcal{B}_r$. In this case, we could iteratively make one pass through the clusters $\mathcal{B}_1, \ldots, \mathcal{B}_r$ and select a block from each cluster. If at the end of this, the total size of the selected blocks is below $k \times sf$ where $sf \geq 1$ is some scaling factor, then we continue to select blocks from the clusters $\mathcal{B}_1, \ldots, \mathcal{B}_r$ till this condition is violated. At this point, we stop and return all the selected blocks – let S denote this set. It is admissible for the scaling factor to be greater than or equal to 1 because the last component of our architecture may resize blocks if needed.

After removing repetitions, our block elimination module computes the

mean μ and standard deviation σ for the priorities of blocks in S. Given a real number $m \geq 0$, let us define a function $Drop(S, m)$ that drops from S all blocks whose priorities are less than $\mu - m\sigma$. Thus, the result of

$$Drop(Cluster(Merge(Peaks(v, r, s), d), t), m)$$

will be a set of all non-repeating high-priority merged peaks taken from v, with respect to the r, s, d, m parameters. Alternatively we may apply the $Drop()$ algorithm to the result of $Merge()$, before clustering blocks.

Example 3.5. Let us continue with Example 3.4 and assume that there are no repetitive blocks. The average priority of the peaks is $\mu = \frac{122}{11} = 11$ and the standard deviation is $\sigma = \sqrt{\frac{1089}{11}} = 10.4$. If we choose $m = 0.25$ and thus delete all blocks whose priorities are less than 8.4, the remaining blocks will be $8 - 10$, $19 - 21$, $28 - 29$, and $30 - 32$. Notice that these are merged blocks whose priorities have been bumped up during the merging process.

Due to its iterative nature, the $Drop()$ algorithm has complexity of $O(card(v))$ – hence it is linear with respect to the number of input blocks.

3.3.2.6 Block Resizing Module

Even after eliminating some low-priority blocks in the previous step, the total frame count of the remaining blocks may still exceed the limit k imposed in the beginning of this paper. In this case, we have to truncate some blocks to fit the limit. Clearly, blocks with higher priorities must have more prominence in the summary and thus occupy a larger percentage of frames. We then devise an algorithm that allocates to each block a number of frames proportional to its priority and truncates blocks to fit the limit of k frames.

Algorithm Resize(v,k)
v is a sequence of block-priority pairs
k is the desired summary length
begin
$Res := \emptyset$
$p_{total} := \sum_{\langle b,p \rangle \in v} p$
$p' := 0$

$k' := 0$
for each $\langle b, p \rangle \in v$ **do**
  if $len(b) \leq \frac{p \cdot k}{p_{total}}$ **then**
    $Res := Res \cup \{\langle b, p \rangle\}$
    $v := v \setminus \langle b, p \rangle$
    $p' := p' + p$
    $k' := k' + len(b)$
  end if
end for
$p_{total} := p_{total} - p'$
$k := k - k'$
for each $\langle b, p \rangle \in v$ **do**
  $alloc := round\left(\frac{p \cdot k}{p_{total}}\right)$
  $b' :=$ b truncated to $alloc$ frames
  $Res := Res \cup \{\langle b', p \rangle\}$
  $p_{total} := p_{total} - p$
  $k := k - alloc$
end for
return Res
end.

The $Resize()$ algorithm collects output blocks in Res and starts by copying all blocks whose length $len(b)$ is smaller than the number of frames they would be allocated in the summary $\left(\frac{p \cdot k}{p_{total}}\right)$. It then computes the remaining number of unallocated frames in k. All remaining blocks are truncated proportionally to their priorities to fit into remaining k frames.

Example 3.6. Let us continue with Example 3.5. There are four blocks that have survived merging and elimination:

- blocks $8 - 10$ with priority $p = 28$,
- blocks $19 - 21$ withe priority $p = 16$,
- blocks $28 - 29$ with priority $p = 11$,
- blocks $30 - 32$ with priority $p = 33$.

Notice that all four are merged blocks, hence there are ranges instead of single numbers. Assuming that each "original" block corresponds to a single frame and the user requested a summary of 5 frames, let us see what the $Resize()$ algorithm does to our summary. First of all, given the total

priority $p_{total} = 88$, all blocks will have to be resized. Block $8 - 10$ has an allocation of $\frac{5 \cdot 28}{88} = 1.59$ frames. As we cannot split frames, this block has to be truncated to two frames $8 - 9$. Block $19 - 21$ has an allocation of $\frac{3 \cdot 16}{60} = 0.8$ and therefore gets truncated to a single frame 20. By repeating this process, we also obtain frames 28 and 31. Thus, the final summary is made of frames $8, 9, 20, 28, 31$.

The $Resize()$ algorithm has complexity of $O(card(v))$ – hence it is linear with respect to the number of input blocks.

Chapter 4

Automatic Creation of Stories

4.1 Introduction

There are numerous applications where there is a need to rapidly infer a story about a given subject – a person, an event, an artifact or a place – from a given set of potentially heterogeneous data sources. For example, consider a person walking through the archaeological site at Pompeii who encounters an painting labeled with a simple statement such as "Death of Pentheus". Though a casual tourist may be satisfied with the knowledge that there is a beautiful painting depicting some unfamiliar event, a student or a person with a deeper interest in culture may be unsatisfied. He/she may want to know more about *Pentheus*, events surrounding his death, etc. In the same vein, consider a police officer who has to serve a warrant at a particular address. The police officer may want to get the quick story on this address: (i) who lives there? (ii) what can be said about these people? (iii) who lives in the neighborhood? (iv) what is their background? And so on.

What constitutes a story may vary dramatically from one example to another. In the case of Pompeii, users may be interested in cultural, historical, mythological, and artistic aspects of the entities about whom stories are being woven. On the other hand, these aspects may not be of great interest to the police officer who instead may want to assess threats existing in the

area. Thus, what goes into a story depends not only on the basic facts about the subject of interest, but also on the subject's domain and user's interests.

There are two other important aspects of stories: they must be *succinct* and they must allow the user to *explore* different facets of the story that are of interest to her. The police officer probably wants just the pertinent facts, not a long complex story about the genealogy of the residents of the house he is going to.

In this work, we formally define a story to be a set of facts about a given subject that satisfies a "story length" constraint. An optimal story is a story that maximizes the value of an objective function measuring its quality and its ability to satisfy user's needs. We present algorithms to extract stories from text and other data sources. We also develop an algorithm to compute an optimal story, as well as three heuristic algorithms to rapidly compute a suboptimal story. The proposed STORY framework supports the goals of succinctness and exploration and creates stories with respect to three important parameters analogous to those presented in Chapter 3 for the video summarization: the priority of the story content, the continuity of the story, and the non-repetition of facts covered by the story.

We have implemented a prototype STORY system and applied it to several scenarios, among which the archaeological site of Pompeii. Our STORY system allows us to deliver stories over both wired and wireless networks to multiple heterogeneous devices such as computers, internet terminals, and PDAs. We run experiments to show that constructing stories can be efficiently performed and that the stories constructed by these heuristic algorithms are high quality stories.

The organization and key contributions of this chapter are as follows:

1. In Section 4.2, we present the concept of a *story schema* and *story instance*. Story schemas and instances can be applicable to diverse data sources. Informally speaking, a story is a set of facts obtained from a set of data sources.

2. Section 4.3 introduces optimal stories and the story computation problem and shows that optimal story computation is NP-complete.

3. In Section 4.4, we develop the OptStory algorithm which is guaranteed to find an optimal story. As the optimal story computation problem is NP-complete, this algorithm is inefficient. We therefore develop three heuristic algorithms – OptStory$^+$, GenStory, and DynStory.

4. In Section 4.5, we describe how an ordered collection of facts is rendered into an actual narrative in English or other language.

Furthermore, in Chapter 5, we show how to extract data (i.e. facts) from heterogeneous data sources including text sources and relational/XML sources, while Chapter 7 presents experiments that attempted to measure subjective qualities of the stories produced by our algorithms and discusses experimental results.

4.2 Story Schema and Instance

In this section, we describe concepts of a *story schema* and a *story instance*. We assume the existence of some set $\mathcal{E}$ whose elements are called entities.

Intuitively, entities describe the objects of interest. In a museum, the objects of interest could be all the known people depicted by images or sculptures shown in the museum, as well as all the other people related to those people in some way. Additionally, the set of entities could include all places depicted. In the case of the police officer, entities of interest could include all people about whom the police have information. *Note that there is no need to explicitly enumerate this set of entities – they could, for example, be discovered by an algorithm seeking entities* [6].

Definition 4.1 (ordinary attributes)**.** Suppose $\mathcal{E}$ is a set of entities. We assume the existence of a universe $\mathcal{A}$ whose elements are called *ordinary attributes*. Each attribute A has an associated domain $dom(A)$. We say that $\mathcal{A}$ is a *set of ordinary attributes associated with the set $\mathcal{E}$ of entities* if $\mathcal{E} \subseteq \bigcup_{A \in \mathcal{A}} dom(A)$.

The above requirement merely ensures that each entity can be characterized by the values of ordinary attributes.

Intuitively, the story about *Pentheus* may have many ordinary attributes. One ordinary attribute might be `mother` – the domain of `mother` could be the set of all alphabetical strings representing female first names. The *value* of this attribute could be the string "Agave". Attributes do not need to be elementary types. An attribute such as `persons` could have as its domain, the powerset of the set of names of people known in Greek Mythology. In the *Pentheus* example, the value could be {"Pentheus", "Agave", "Maenads"} – note that *Maenads* is not one person, but rather a collective name for a group of people.

Notice that it is entirely possible that the value of an attribute could be an entity by itself and there may be a story about this entity which involves other entities as well. In the *Pentheus* example, *Agave* (his mother) could be an entity about whom many attributes have known values. However, the attribute `occupation` for *Pentheus* may have the value "king" which is in $dom($`occupation`$)$ but is not an entity.

There are many cases where we may have multiple values for an attribute and want to generalize them into one.

Definition 4.2 (generalization function)**.** Suppose A is an ordinary attribute. Then a *generalization function* for attribute A is a mapping Γ_A from $2^{dom(A)}$ to $dom(A)$.

For example, suppose we have an attribute called `occupation` whose domain is the set of all strings representing an occupation. A generalization function $\Gamma_{occupation}$ may map a set of strings of the form "king of ..." to a single string "king" This is just one example of a generalization function – many more are possible.

In the above discussion, attributes have invariant values. However, there are many situations where attributes may have time-varying values. For example, *Pope Paul III* may have an `occupation` attribute with the value "Cardinal" from 1493 to 1533 and "Pope" from 1534 to 1549 – we have ignored exact dates of ascension here and just approximated the years. In order to express this kind of information, we introduce the concept of a time-varying attribute.

Definition 4.3 (time-varying attribute)**.** A *time-varying attribute* is a pair $(A, dom(A))$ where A is the name of the attribute and $dom(A)$ is the domain of values for the attribute.

A time-varying attribute looks just like an ordinary attribute. However, a value for a time varying attribute associates an interval.

Definition 4.4 (timevalue)**.** A *timevalue* for a time-varying attribute $(A, dom(A))$ is a set of triples (v_i, L_i, U_i) where $v_i \in dom(A)$ and L_i, U_i are either integers or the special symbol $\perp$ (denoting unknown). A timevalue is *fully specified* iff there is no triple of either the form $(v_1, \perp, U_i)$ or $(v_i, L_i, \perp)$ or $(v_i, \perp, \perp)$ in it.

Intuitively, if an object has a time-varying attribute $(A, dom(A))$ with a timevalue of $\{(v_1, 15, 20), (v_2, 25, 30)\}$ this means that the attribute has value v_1 between times 15 and 20 and value v_2 between times 25 and 30. In the case of *Pope Paul III*, the timevalue of `occupation` is given by {("Pope",1534,1549),("Cardinal",1493,1533)}.

Definition 4.5 (consistent timevalue)**.** A timevalue tv for a time-varying attribute $(A, dom(A))$ is *consistent* iff there is no pair (v_1, L_1, U_1), (v_2, L_2, U_2) in tv such that $v_1 \neq v_2$ and $L_1, U_1, L_2, U_2 \neq \perp$ and such that the intervals $[L_1, U_1] \cap [L_2, U_2]$ intersect.

Intuitively, consistency of a timevalue ensures that the attribute does not have two distinct values at the same time (e.g. Pope Paul III could not be both pope and cardinal at the same time). Thus, the timevalue {("Pope",1534,1549),("Cardinal",1493,1533)} for Pope Paul III's `occupation` attribute is consistent.

Note, however, that had we wanted to allow a person to have multiple occupations at the same time, we could simply have defined the domain of `occupation` to be the *powerset* of the set of all strings rather than the set of all strings.

Note 4.1. Throughout the rest of this paper, we will abuse notation and use the term attribute to refer to both ordinary and time-varying attributes. The context will determine the usage.

Just as we have defined generalization functions for ordinary attributes, we can also define generalization functions for time-varying attributes.

Definition 4.6 (generalization function (cntd.))**.** Suppose $(A, dom(A))$ is a time-varying attribute. A *generalization function* for A is a mapping Γ_A from a timevalue for attribute $(A, dom(A))$ to a singleton timevalue for attribute $(A, dom(A))$ such that if $\Gamma_A(X) = \{(v, L, U)\}$ then $[L, U] \subseteq \left[min_{(v_i,L_i,U_i)\in X} L_i, max_{(v_i,L_i,U_i)\in X} U_i\right]$ and there exists $(v_j, L_j, U_j) \in X$ such that $[L, U] \cap [L_j, U_j] \neq \emptyset$.

For example, a generalization function may map the set of time values {("Bishop of Massa",1538,1552),("Bishop of Nice",1533,1535)} to the singleton timevalue {"Bishop",1533,1552)}.

Note that if the definition of generalization function for time varying attributes did not require that $\exists(v_j, L_j, U_j) \in X \mid [L, U] \cap [L_j, U_j] \neq \emptyset$ then we would have a problem. The generalization function could, for example, return {("Bishop",1536,1537)} even though its input said nothing about the time interval from 1536 to 1537.

The concept of a story schema below specifies the entities of interest, and the attributes of interest for a given story application.

Definition 4.7 (story schema)**.** A *story schema* consists of a pair $(\mathcal{E}, \mathcal{A})$ where $\mathcal{E}$ is a set of entities and $\mathcal{A}$ is a set of attributes associated with $\mathcal{E}$. We will use $\mathcal{A}^o$ and $\mathcal{A}^{tv}$ to denote the ordinary and time-varying attributes in $\mathcal{A}$ repectively.

For example, if the *Sito Archaeologico di Pompei* wishes to allow visitors to learn everything about archaeological ruins at Pompeii, then the set of entities could be defined as follows: (i) the set of all objects in Pompeii that are of interest (including paintings, sculptures, etc.), plus (ii) the objects and events depicted in those paintings, plus (iii) any entities related to entities in the previous two categories. Clearly, the museum workers can only define the first two items in the above list, while the story creation system will automatically derive all other entities using the third criterion.

Definition 4.8 (story instance)**.** A *story instance* w.r.t. story schema $(\mathcal{E}, \mathcal{A})$ is a partial mapping $\mathcal{I}$ which takes an entity $e \in \mathcal{E}$ and an attribute $A \in \mathcal{A}$

and returns as output, a value $v \in dom(A)$ when A is an ordinary attribute, and a timevalue $\{(v, L, U) \,|\, v \in dom(A)\}$ when A is a time-varying attribute.

We use the notation $\mathcal{I}(e, A) = \perp$ to indicate that $\mathcal{I}(e, A)$ is undefined. Attributes like `daughter` may be set valued. For example, *Agamemnon* had several daughters – amongst them *Elektra* and *Iphigenia*. In this case, the attribute `daughter` should have a set valued domain as its type – and thus, an instance would say that the `daughter` attribute of the entity *Agamemnon* has the value {"Elektra","Iphigenia"}. Thus, requiring that each attribute has at most one value per entity leads to no loss of generality as the attribute can assume set values.

Example 4.1 (Pentheus painting)**.** The table below shows other entities related to a Greek Mythology character called *Pentheus* who is depicted in a stunning painting in Pompeii.

Entity	**Attribute**	**Value**
Bacchus	occupation	"god"
	enemy	{("Pentheus",⊥,⊥)}
	friends	{("Maenads",⊥,⊥)}
Maenads	occupation	{("priestess",⊥,⊥)}
	friends	{("Bacchus",⊥,⊥)}

Note 4.2. It is often convenient to think of a story instance as a set of *entity-attribute-value* triples (EAV for short). An EAV triple w.r.t. an instance $\mathcal{I}$ of a story schema $(\mathcal{E}, \mathcal{A})$ is a triple $\langle e, A, v \rangle$ such that $e \in \mathcal{E}$, $a \in \mathcal{A}$ and $v = \mathcal{I}(e, A)$. Let $\mathsf{EAV}_{\mathcal{I}}$ denote the set of all the EAV triples w.r.t. $\mathcal{I}$. We will often abuse notation and switch between these two representations.

4.3 Story Computation Problem

The notion of a story instance does not allow us to include generalized tuples or to handle conflicts. To introduce these features, we will need to define several specialized instances.

In this section, we first define the concepts of a *valid instance* and a *full instance* based on a set of data sources. Intuitively, these concepts are used to collect all facts reported by a set of data sources. We then define

a closed instance by allowing generalization. However, given any topic or entity, there may be many stories that can be associated with that topic or entity. To address this, we define continuity, priority, and non-repetition criteria to test if one story is better than another. Later, in Section 4.4, we will present several algorithms for story computation.

4.3.1 Valid and Full Instances

In order to create a story from a story schema, one may need to access a variety of sources. In the case of Pentheus, we may need to access electronic Greek texts to find out more about him. Let us assume that our data sources have an associated application program interface (this is a reasonable assumption as most commercial programs do have APIs). The *source access table* describes how to extract an attribute's value using a source's API.

Definition 4.9 (source access table)**.** A *source access tuple* sat is a triple $(A, s, f_{A,s})$ where A is an attribute name, s is a data source, and $f_{A,s}$ is a partial function (body of software code) that maps objects to values in $dom(A)$ when A is an ordinary attribute, and to time values over $dom(A)$ when A is a time-varying attribute. A *source access table* SAT is a finite set of source access tuples.

The source access table does not, of course, need to be populated with a function for each source and each attribute. Some sources may provide some information, while others may not. The functions $f_{A,s}$ are *partial functions* because some sources may not have information about certain entities. When implementing the STORY system, we created several $f_{A,s}$ functions capable of extracting data from relational tables, XML hierarchies, and HTML documents returned by the Google search engine[1]. In Chapter 5, we will describe in detail our algorithms to extract entity-attribute-value triples from text (e.g. Web) sources. We will also outline algorithms to extract entity-attribute-value triples from relational and XML sources.

[1]Our algorithm can be used in conjunction with any algorithm for topic discovery [2] and can be applied to any corpus of documents whatsoever, rather than applying them to web documents whose accuracy is questionable.

Note 4.3. The developer of an application requiring stories about a certain domain needs to specify the functions $f_{A,s}$ in the source access table. Such a function must return timevalues when A is a time-varying attribute. This can be quite difficult. For instance, determining when Pentheus was killed from a text document is a nontrivial task. Had we allowed timevalues to be more general (e.g. to say Pentheus was killed after some other event, or to say Pentheus was killed within 5 years of yet another event), then the functions $f_{A,s}$ would need to infer this even more complex information from textual sources. This incredibly challenging problem is beyond the scope of this work.

Definition 4.10 (valid instance)**.** Suppose $(\mathcal{E}, \mathcal{A})$ is a story schema, SAT is a source access table, and $\mathcal{I}$ is an instance. $\mathcal{I}$ is said to be *valid* w.r.t. SAT iff for every entity $e \in \mathcal{E}$ and every attribute $A \in \mathcal{A}$, if $\mathcal{I}(e, A)$ is defined, then there is a triple of the form $(A, s, f_{A,s})$ in SAT such that $f_{A,s}(e) = \mathcal{I}(e, A)$.

Intuitively, the above definition says that an instance is valid w.r.t. some source access table if every fact (i.e. every assignment of value to an attribute for an entity) is supported by at least one source. *Note that different sources may disagree on the value of a given attribute for a given entity.* For instance, one source may say Pentheus' mother is Agave, while another may say it is Hera. We now define the concept of a *full instance* that collects together the set of all values for attribute A of entity e from all sources.

Definition 4.11 (full instance)**.** Suppose $(\mathcal{E}, \mathcal{A})$ is a story schema and SAT is a source access table. Suppose $\mathcal{I}$ is an instance w.r.t. $(\mathcal{E}, \mathcal{A}')$ where the attributes in $\mathcal{A}'$ are the same as the attributes in $\mathcal{A}$ with one difference – if an attribute $A \in \mathcal{A}$ has $dom(A) = 2^{2^S}$, then the corresponding attribute $A' \in \mathcal{A}'$ has $dom(A') = dom(A)$. Otherwise, $dom(A') = 2^{dom(A)}$, i.e. the powerset of the original domain. $\mathcal{I}$ is said to be the *full instance* w.r.t. $(\mathcal{E}, \mathcal{A})$ and SAT iff for all entities $e \in \mathcal{E}$ and attributes $A \in \mathcal{A}$,

$$\mathcal{I}(e, A) \quad = \quad \begin{cases} \bigcup_{\forall s \,|\, (A,s,f_{A,s}) \in \mathsf{SAT}} f_{A,s}(e) & if\ dom(A) = 2^{2^S} \\ \{f_{A,s}(e) \mid (A, s, f_{A,s}) \in \mathsf{SAT}\} & otherwise \end{cases} .$$

Intuitively, the above definition says that an instance is full when it accumulates all the facts reported by various sources, independently of whether

these facts are conflicting or not. We will describe how conflicts may be resolved later on in this chapter (Definition 4.14).

4.3.2 Stories

In this section, we define how to generalize information contained in full instances, resolve conflicts in this information, and create stories out of instances. A generalized story schema is a story schema, together with an equivalence relation on attribute domains and a generalization function.

Definition 4.12 (generalized story schema)**.** A *generalized story schema* is a quadruple $(\mathcal{E}, \mathcal{A}, \sim, \mathcal{G})$ where $(\mathcal{E}, \mathcal{A})$ is a story schema, $\sim$ is a mapping which associates an equivalence relation on $dom(A)$ with each attribute $A \in \mathcal{A}$ and $\mathcal{G}$ is a mapping which assigns, to each attribute $A \in \mathcal{A}$, a generalization function Γ_A for attribute A.

Intuitively, a generalized story schema consists of a regular story schema, a function that associates an equivalence relation with each attribute domain and a function that associates a generalization function with each attribute domain. An equivalence relation on the domain $dom(A)$ of attribute A specifies when certain values in the domain are considered equivalent. For example, we may consider string values "king" and "monarch" to be equivalent in $dom(\texttt{occupation})$. For a time-varying attribute, we may consider ("king", L, U) and ("monarch", L', U') to be equivalent independently of whether $L = L' \wedge U = U'$ is true or not. Likewise, in the example of Pope Paul III, the equivalence relationship may say that the triplet ("Bishop of ...",-,-) is always equivalent to other triplets of the form ("Bishop of ...",-,-) independently of whether the bishops governed different places. Our system uses WordNet [47] to infer equivalence relationships between terms.

The definition of a closed instance below takes a full instance associated with a source access table and closes it up so that generalization information can be included.

Definition 4.13 (closed instance)**.** Suppose ($\mathcal{E}$, $\mathcal{A}$, $\sim$, $\mathcal{G}$) is a generalized story schema and $\mathcal{I}$ is the full instance w.r.t. ($\mathcal{E}$, $\mathcal{A}$). The *closed instance* w.r.t. a source access table SAT and generalized story schema ($\mathcal{E}$, $\mathcal{A}$, $\sim$, $\mathcal{G}$)

is defined as $\mathcal{I}'(e, A) = \mathcal{I}(e, A) \cup \{\Gamma_A(X') \mid X' \text{ is a } \sim_A\text{-equivalence class of } \mathcal{I}(e, A)\}$.

Intuitively, here is how we find the closed instance associated with a given source access table and a given generalized story schema. For each entity e and each attribute A of the entity:

1. We first compute the set $\mathcal{I}(e, A)$ where $\mathcal{I}$ is the full instance associated with our source access table.

2. We then split $\mathcal{I}(e, A)$ into equivalence classes using the equivalence relation $\sim_A$ on $dom(A)$. Suppose the equivalence classes thus generated are $X_1, \ldots, X_n$.

3. For each equivalence class X_i, we compute $\Gamma_A(X_i)$ – this is the generalization of the equivalence class X_i using the generalization function Γ_A associated with attribute A. Suppose $\Gamma_A(X_i) = v_i$.

4. We insert the tuple $\langle e, A, v_i \rangle$ into the full instance.

This process is repeated for all entities e and all attributes A. After all tuples of the form shown above have been inserted into the full instance, it becomes the closed instance.

A story cannot be defined based on a full instance alone. In the real world, the "full story" about any single person or event is likely to be very complex and involve a large amount of unimportant minutiae. For example, consider the story of Pope Paul III. Depending on what items about Pope Paul III are considered important, we may choose to merely say that he served as a bishop from 1538 to 1556 and ignore the details. However, the full instance associated with Pope Paul III may not explicitly say this – rather it might state (as in our example) that he was a bishop of this place for some time, that place for another time period, and so on. Generalization is needed for this.

Note that so far we have not tried to resolve possible conflicts between attribute values obtained from different sources. However, such conflicts need to be resolved before we can create a story. In other words, suppose $\mathcal{I}'$

is a closed instance. Whenever $\mathcal{I}'(e, A)$ is of cardinality two or more, some mechanism is required to get rid of all but one member in $\mathcal{I}'(e, A)$.

Definition 4.14 (conflict management policy)**.** Given an attribute A such that $dom(A) = 2^{2^S}$, the *conflict management policy* χ_A is a mapping from $dom(A)$ to $dom(A)$ such that $\chi(X) \subseteq X$. For any other attribute A, χ_A is a mapping from $2^{dom(A)}$ to $dom(A)$ such that $\chi(X) \in X$.

There is an extensive literature [30] on conflict resolution whose results can be directly plugged in as conflict management policies – three of these are shown below.

1. **Temporal conflict resolution.** Suppose different data sources provide different values $v_1, \ldots, v_n$ for $\mathcal{I}(e, A)$. Suppose value v_i was inserted into the data source at time t_i. In this case, we pick the value v_i such that $t_i = max\{t_1, \ldots, t_n\}$. If multiple such i's exist, one is selected randomly.

2. **Source based conflict resolution.** The developer of a story may assign a *credibility* c_i to each source s_i that provides a value v_i for attribute A of entity e. This strategy picks value v_i such that $c_i = max\{c_1, \ldots, c_n\}$. If multiple such i's exist, one is selected randomly.

3. **Voting based conflict resolution.** Each value v_i returned by at least one data source has a *vote*, $vote(v_i)$. $vote(v_i)$ is the number of sources that return value v_i. In this case, this conflict resolution strategy returns the value with the highest vote. If multiple v_i's have the same highest vote, one is picked randomly and returned.

These are just three example strategies. It is easy to pick hybrids of these strategies as well. For example, we could first find the values for $\mathcal{I}(e, A)$ with the highest votes and then choose the one which is most recent (temporal). A deconflicted instance is one from which conflicts have been removed.

Definition 4.15 (deconflicted instance)**.** Suppose $(\mathcal{E}, \mathcal{A}, \sim, \mathcal{G})$ is a generalized story schema and $\mathcal{I}'$ is the closed instance w.r.t. $(\mathcal{E}, \mathcal{A})$. The *deconflicted instance* w.r.t. a source access table SAT, generalized story schema

$(\mathcal{E}, \mathcal{A}, \sim, \mathcal{G})$, and conflict management policy χ is the instance $\mathcal{I}^\sharp$ such that for all entities $e \in \mathcal{E}$ and all attributes $A \in \mathcal{A}$ if $\mathcal{I}^\sharp(e, A) \neq \perp$ then $\mathcal{I}^\sharp(e, A) = \chi(\mathcal{I}'(e, A))$.

Note that finding any arbitrary strong or deconflicted instance is not enough. The reason is a technical one. The instance $\mathcal{I}_{null}$ which is undefined for all $\mathcal{I}_{null}(e, A)$ has no conflicts – however it is not very useful as it has no information in it.

Definition 4.16 (story)**.** Suppose $\mathcal{I}$ is a closed instance w.r.t. a generalized story schema $(\mathcal{E}, \mathcal{A}, \sim, \mathcal{G})$ and a source access table SAT, and $e \in \mathcal{E}$ is an entity. Then a *story* $\sigma(e, \mathcal{I})$ of size k, is a sequence of attribute-value pairs $\langle A_1, v_1 \rangle, \ldots, \langle A_k, v_k \rangle$ such that for all $1 \leq i \leq k$, $A_i \in \mathcal{A}$ and $v_i = \mathcal{I}(e_i, A_i)$.

A *deconflicted story* w.r.t. a given conflict management policy χ is a sequence of attribute-value pairs $\langle A_1, v_1 \rangle, \ldots, \langle A_k, v_k \rangle$ such that for all $1 \leq i \leq k$, $A_i \in \mathcal{A}$ and $v_i = \mathcal{I}^\sharp(e_i, A_i)$ where $\mathcal{I}^\sharp$ is the deconflicted instance w.r.t. χ.

Note that the above definition of a story only lists the essential facts in a story. Our Story system presents these facts in English (a Spanish version also exists) to the user in one of two ways: if the fact was derived from a relational or XML source, then a template is used to output the fact in English. If the fact was derived from a text document, then either a template or the sentence from which the fact was extracted may be used as the output. We have approximately 400 templates currently in our system.

Note 4.4. Throughout the rest of this chapter, we will use the word "story" to refer to both ordinary and deconflicted stories.

4.3.3 Optimal Stories

There are good stories and bad stories even when they are about the same topic and even when they are derived from the same instance. So, what makes a story good?

First of all, the facts included in the story have to be *relevant* to the user. For example, the fact that Pentheus' mother was Agave is probably more important than the length of Pentheus' big toe. Thus, the first fact

is better be told to the user in the beginning of the story while the second fact can be included at the end or omitted altogether.

Secondly, the story has to be continuous by delivering facts in the order expected by the user. If the facts occur over some period of time, it is logical to tell them in the order of occurrence. But even for such basic facts as the place of birth or parents' names, there is a certain customary order of delivery (i.e. "X has been born in P from Y and Z"). Violating this order will make story less comprehensible.

Finally, we would not want to repeat same or similar facts again and again in the same story. Redundancy is *not* a virtue when it comes to storytelling.

To help us choose stories that are "better" than others from the universe of possibilities, we define a story evaluation function.

Definition 4.17 (story evaluation function)**.** Suppose $\mathcal{S}$ is the set of all possible stories about some entity e w.r.t. the same schema and source access table. The *story evaluation function* $eval(s)$ takes a story s and returns a real value in the $[0, 1]$ range that measures how good s is, with higher values corresponding to better stories.

The reader will note immediately that there are many ways of defining the evaluation function, not limited to the three general criteria outlined above, that are similar to the ones proposed in [16] and discussed in Chapter 3. Our story creation algorithms can work with *any* evaluation function.

Problem 4.1 (optimal story computation)**.** Given a closed instance $\mathcal{I}$, a positive integer k, and an entity $e \in \mathcal{E}$ as input, find a *story* $\sigma(e, \mathcal{I})$ of size $\leq k$ that maximizes the value of a given evaluation function. In this case, $\sigma(e, \mathcal{I})$ is called an *optimal story*.

Theorem 4.1. Given all the parameters listed in the above problem statement, and given a story S, determining if S is optimal is an NP-complete problem.

Membership in NP will be proved by the OptStory algorithm presented in Section 4.4. NP-hardness is proved by a straightforward reduction of the

knapsack problem to that of stories.[2]

4.4 Story Computation

We start this section by presenting an algorithm to find optimal stories. We then present three heuristic algorithms that build upon the work in video summarization algorithms presented in [16] and do not necessarily find an optimal story, but create "good enough" stories in a reasonable time.

Given an entity e, the OptStory algorithm finds an *optimal* story of length k by maximizing the value of the evaluation function.

Algorithm OptStory(e,SAT,k)
  e is an entity
  SAT is a source access table
  k is the requested story size
begin
  $\mathcal{I} := Deconf\mathcal{I}(Closed\mathcal{I}(Full\mathcal{I}(e, SAT)))$[3]
  return RecStory($\emptyset, \mathcal{I}, k$)
end.

The OptStory algorithm first picks all data about the entity e available in $\mathcal{I}$. It then calls the recursive RecStory algorithm that enumerates over all possible stories of k or fewer attributes that can be derived from the given data and returns the best story with respect to the evaluation function $eval()$.

Algorithm RecStory($Story$,$Data$,k)
  $Story$ is the story so far
  $Data$ is the set of attribute-value pairs to assign
  k is the remaining story size
begin
  $\langle\, BestS, BestW \,\rangle := \langle\, Story, eval(Story) \,\rangle$
  if $k > 0$ **then**
    for each $\langle\, A, v \,\rangle \in Data$ **do**
      $S := Story$ with $\langle\, A, v \,\rangle$ attached to the tail
      $\langle\, S, W \,\rangle :=$ RecStory($S, Data \setminus \{\langle\, A, v \,\rangle\}, k - 1$)

[2] Intuitively, given an instance of the knapsack problem involving a knapsack of capacity C and objects $o_1, \ldots, o_k$ of weights $w_1, \ldots, w_k$ and profits $p_1, \ldots, p_k$ respectively, we can consider each o_i to be a fact with the same weights and profits.

[3] $Full\mathcal{I}()$, $Closed\mathcal{I}()$ and $Deconf\mathcal{I}()$ denote functions that return full, closed and deconflicted instances respectively

```
            if W > BestW then ⟨ BestS, BestW ⟩ := ⟨ S, W ⟩
        end for
    end if
    return ⟨ BestS, BestW ⟩
end.
```

4.4.1 Restricted Optimal Story Algorithm

Given n attributes, the RecStory algorithm will have to sort through $\sum_{0 \leq i \leq k} \frac{n!}{(n-i)!}$ stories. Even if we restrict the algorithm to the k-length stories, it will still have to consider $\frac{n!}{(n-k)!}$ stories. To make story creation more manageable, let us consider the following algorithm.

```
Algorithm RecStory+(Story,Data,k,b)
    Story is the story so far
    Data is the set of attribute-value pairs to assign
    k is the remaining story size
    b is the branching factor
begin
    ⟨ BestS, BestW ⟩ := ⟨ Story, eval(Story) ⟩
    Q is a priority queue
    if k > 0 then
        for each ⟨ A, v ⟩ ∈ Data do
            S := Story with ⟨ A, v ⟩ attached to the tail
            Q.add(S, eval(S))
            if length(Q) > b then Q.delete(tail(Q))
        end for
        for each SS ∈ Q do
            ⟨ S, W ⟩ := RecStory+(SS, Data \ SS, k − 1)
            if W > BestW then ⟨ BestS, BestW ⟩ := ⟨ S, W ⟩
        end for
    end if
    return ⟨ BestS, BestW ⟩
end.
```

The RecStory$^+$ algorithm essentially limits the search at each step to the b best stories w.r.t. the evaluation function. Given n attributes, this algorithm only considers $1+ \sum_{0 \leq i < k}(b^i \cdot (n-i))$ stories. We use OptStory$^+$ to denote the algorithm that calls RecStory$^+$.

4.4.2 Genetic Programming Approach

In this section, we present a story creation algorithm GenStory based on genetic programming. GenStory creates suboptimal stories too.

Algorithm GenStory(e,SAT,k,N,δ)
  e is an entity
  SAT is a source access table
  k is the requested story size
  N is the desired number of iterations
  δ is the desired fitness threshold
begin
  $Data := DeconfI(ClosedI(FullI(e, SAT)))$
  $R := \left\lceil \frac{card(Data)}{k} \right\rceil$
  $Q :=$ R random solutions of k attributes from $Data$
  for $j \in [1, N]$ **do**
    for $i \in [1, R]$ **do**
      $S :=$ solution randomly chosen from Q
      choose random $\langle A, v \rangle \in Data$ and $\langle A', v' \rangle \in S$
      replace $\langle A', v' \rangle$ in S with $\langle A, v \rangle$
      $Q := Q \cup \{S\}$
      $Q := Q \setminus \{S'\}$ where $\forall S \in Q\, eval(S) \geq eval(S')$
      if $max_{S_1,S_2 \in Q} |eval(S_1) - eval(S_2)| \leq \delta$ **then**
        return best solution from Q
      end if
    end for
  end for
  return best solution from Q
end.

The GenStory algorithm starts by creating a population Q of $\left\lceil \frac{card(Data)}{k} \right\rceil$ random stories. It will then repeatedly choose a random story S from this population and replace a random attribute in this story with a different attribute not occurring in S. The resulting story is added to Q, and then the story with the lowest $eval()$ value is deleted from Q. The GenStory algorithm will terminate when all story candidates in the population Q have approximately the same worth (w.r.t. the value of δ) or when the maximal number of iterations N is reached.

4.4.3 Dynamic Programming Approach

In this section, we present a story creation algorithm DynStory based on the dynamic programming approach. This algorithm also yields stories that

are suboptimal, yet does it in less time than the the OptStory algorithm.

Algorithm DynStory(e,SAT,k)
  e is an entity
  SAT is a source access table
  k is the requested story size
begin
  $Data := DeconfI(ClosedI(FullI(e, SAT)))$
  $S :=$ random solution of k attributes from $Data$
  $Data := Data \setminus S$
  while $Data \neq \emptyset$
    $subs := false$
    $r := 1$
    while $r \leq k$ and $subs = false$ **do**
      $S' := S$ with $\langle A_r, v_r \rangle$ replaced with $first(Data)$
      if $eval(S) < eval(S')$ **then**
        $S := S'$
        add $\langle A_r, v_r \rangle$ to the tail of $Data$
        $subs := true$
      else
        $r := r + 1$
      end if
    end while
    remove $first(Data)$ from $Data$
  end while
  return S
end.

The DynStory algorithm starts by creating a random solution S and proceeds by trying to replace each attribute in S with the first attributes from the list of candidates $Data$. As soon as a better solution is found, it takes the place of S. The algorithm terminates when the list of candidates is exhausted. The time complexity of DynStory is linear w.r.t. the number of attributes in the instance $Data$.

4.5 Story Rendering

A *story*, as defined so far, is a collection of a given number of known facts about a given entity. Facts to be included in the story are selected based on user's preferences which are somehow taken into account through an objective function.

The last issue to be addressed in any automatic story generation environment is how to present stories to the users. Of course, most users would not be satisfied with a list of facts presented as database entries. They would prefer to get the interesting facts rendered as a narrative text in English – or any other language. Rendering a collection of facts into text that reads well is not a trivial task.

The STORY system renders the set of facts constituting a story in one of two ways described below.

Original sentences from the sources. When extracting facts from text documents, we store sentences from which these facts have been extracted. When narrating a story, these sentences can be used. This solution is not feasible for rendering facts extracted from other types of data sources – such as relational databases and XML files – where a piece of text describing the fact is not available.

Templates. Whenever original sentences are not available a template may be used to construct a sentence out of an EAV triple. A template operates in a way similar to the mechanism offered by most programming languages for generating formatted strings.

Definition 4.18 (text rendering template)**.** A *text rendering template* is a string containing the symbols $\%e$, $\%a$ and $\%v$, marking the position within the string where the entity name, attribute name and value should be placed. A template is said to be valid if it contains at least the symbols $\%e$ and $\%v$. Given a story schema $(\mathcal{E}, \mathcal{A})$, let $\mathcal{T}$ be a mapping from $\mathcal{A}$ to the set of all valid templates that associates each attribute $A \in \mathcal{A}$ with one or more templates.

In other words a template is a string that represents the desired structure of a sentence, with the positions of actual entity names, attribute names and values marked by special symbols. For example, the string "The %a of %e is %v" may be a valid template for attributea such as `mother` and `father`, and it would render sentences like "The *mother* of *Pentheus* is *Agave*".

Note 4.5. Given a story schema $(\mathcal{E}, \mathcal{A})$, at least one template should be associated with each attribute $A \in \mathcal{A}$, in order to be able to render any fact. So

one may think that the number of templates should be equal to the number of distinct attributes in $\mathcal{A}$, but this is not the case for two reasons. First of all, the same template may be reused for several attributes. In fact entire classes of attributes – e.g. `sons`, `daughters`, `brothers`, `sisters`, etc. – may share the same template, because their nature allows to present their values in the same way. The STORY system actually allows the application developer to define templates at attribute cluster level. Attributes are grouped into a hierarchy of clusters based on semantic properties that we can infer from WordNet. Secondly, we may allow the $\mathcal{T}$ mapping to be incomplete. The rendering function introduced in Definition 4.19 would use a default template – e.g. "The %a of %e is %v" – to render any attribute A such that $\mathcal{T}(A) = \emptyset$.

Note 4.6. The symbol $\%a$ is not required for a template to be valid. The reason is that the constant part of the template may already include the semantic of the attribute. For example, the attribute `birthdate` may have an associated template like "%e was born on %v".

Note 4.7. The $\mathcal{T}$ mapping may associate more than a single template with each attribute. This choice derives from the consideration that, different persons, or even the same person at different times, may use different sentence arrangements to describe the same fact. Having different templates to render the same fact may help to make the generated narrative more similar to what would be produced by a human being.

Definition 4.19 (rendering function)**.** Suppose $(\mathcal{E}, \mathcal{A})$ is a story schema and $\mathcal{I}$ is an instance. A rendering function $\mathcal{R}$ is a mapping from $\mathsf{EAV}_{\mathcal{I}}$ to the set of strings. For each $\langle e, A, v \rangle \in \mathsf{EAV}_{\mathcal{I}}$ the rendering function picks a template t from $\mathcal{T}(A)$ and replaces the markers $\%e$, $\%a$ and $\%v$ in t with e, A and v respectively.

Note 4.8. Note that when an attribute A is single-valued the rendering function merely replace $\%v$ with v. When A is set-valued (Example 4.2) $\%v$ is expanded as "$\%v\ [\{, \%v\}\ \textit{and}\ \%v]$", where each $\%v$ represents an element of the set value. When A is a time-varying attribute (Example 4.3) $\%v$ is expanded as "$\%tvt\ [\{, \%tvt\}\ \textit{and}\ \%tvt]$", where each

$\%tvt$ represents a (v, L, U) triple in the time-value. $\%tvt$ is expanded as "$\%v$ [$from$ $\%l$] [to $\%u$]", where $\%l$ and $\%u$ mark the position of L and U respectively.

Example 4.2. Suppose that $\mathcal{T}(\texttt{sons}) = \{$"$\%v$ were $\%e$'s $\%a$"$\}$ and consider the EAV triple $\langle$"Zeus",`sons`,$\{$"Apollo","Ares","Hermes"$\}\rangle$. The function $\mathcal{R}$ picks the only available template for the attribute `sons` and renders the triple as "*Apollo, Ares and Hermes* were *Zeus*'s *sons*".

Example 4.3. Suppose that $\mathcal{T}(\texttt{occupation}) = \{$"$\%e$ has been $\%v$"$\}$ and consider the EAV triple $\langle$"Pope Paul III",`occupation`,{("Pope",1534,1549), ("Cardinal",1493,1533)}$\rangle$. The function $\mathcal{R}$ picks the only available template for the attribute `occupation` and renders the triple as "*Pope Paul III* has been *Cardinal from 1493 to 1533 and Pope from 1534 to 1549*".

Chapter 5

Information Extraction from Text Sources

5.1 Attribute Extraction

Section 4.3.1 has formally defined a *full instance* and how it can be obtained using the source access tables, but avoided discussion of the actual ways to extract attribute values from heterogenous data sources. In this chapter we describe in detail how to extract attribute values from text sources (Section 5.1.1), and how to identify the entities of interest to a given domain (Section 5.1.2). Furthermore, we briefly describe how to extract attribute values from relational and XML sources (Section 5.1.3).

5.1.1 Attribute Extraction from Text Sources

The Text Attribute Extraction (TAE) algorithm to extract attribute values from text sources takes as input a domain name (e.g. "Greek Mythology") and a set of sources (selected web sites, news feeds, or the entire web to be searched using a search engine). It assumes the existence of various subroutines and it is based on the concept of *extraction rule*.

Given a sentence s the constituent tree CT_s of s is a tree representing the syntactic structure of s, whose nodes are labeled NP (Noun Phrase), VP (Verb Phrase), and so on. A standard way of representing such structures has been proposed by the Penn Treebank Project [40]. Figure 5.1.a shows the constituent tree of the sentence "Rome is the capital of Italy". The

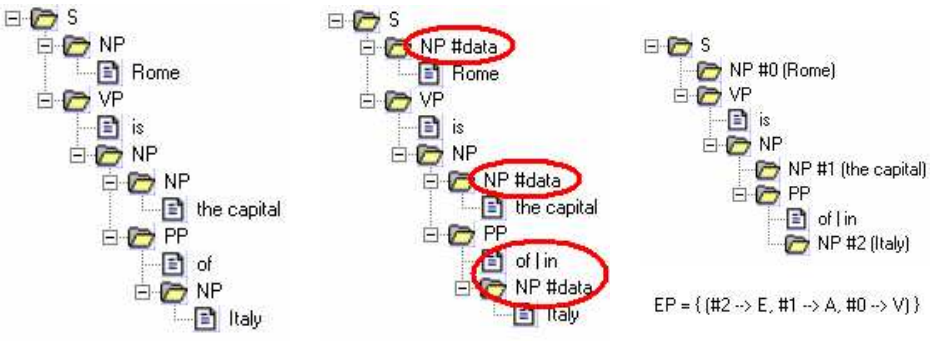
STORY system uses a parser based on the Link Grammar [60] to analyze the syntactic structure of a sentence and generate its constituent tree. We can now define the concept of *extraction rule*.

Definition 5.1 (extraction rule)**.** An extraction rule R is a pair (CT_R, EP), where CT_R is a constituent tree with some leaf nodes marked as data nodes and EP is a set of extraction patterns, an extraction pattern being a function that maps data nodes to the elements of an EAV triple. Let $\mathcal{R}$ denote the set of all extraction rules.

Figure 5.1.c shows an example of extraction rule. It is clear from the picture that the constituent tree of an extraction rule is the constituent tree of a prototype sentence, where subtrees corresponding to portions of the sentence to be considered as data have been replaced by a single node marked as a data node, while an extraction pattern specifies which data nodes should be considered the entity, the attribute and the value respectively. Extraction rules can thus be learned from examples. Actually the process to create extraction rules operates as follows: (i) the user types in a prototype sentence and the system parses it producing its constituent tree[1] (Figure 5.1.a); (ii) the user marks the nodes of the trees that represents data (the whole subtrees rooted at these nodes are considered as a piece of information) and adds alternatives for constant nodes, such as prepositions and sentence connectors (Figure 5.1.b); (iii) if the markup at previous step is valid the user can then add the extraction patterns specifying the role of each data node (Figure 5.1.c).

Note 5.1. Other classes of extraction patterns are available in the system. They use data structures other than EAV triples. For example the EAO model is useful to extract $\langle\, Entity, Action, Object \,\rangle$ triples from sentences such as "Maenads killed Pentheus", while the EQ model is useful to extract $\langle\, Entity, Quality \,\rangle$ pairs from sentences such as "Maenads were cruel". Tuples extracted using these models should be then properly mapped to the EAV model. Mapping rules have been defined to this aim. For example an

[1] Depending on the sentence, more than a single constituent tree may be returned, due to different possible interpretations of the sentences. In that case only a single tree is picked to build the rule.



a b c

Figure 5.1: Extraction rules

EQ pair $\langle e, q \rangle$ is mapped to an EAV triple built as $\langle e, \texttt{quality}, q \rangle$ – e.g. the EQ pair $\langle$ "Maenads", "cruel" $\rangle$ would thus be converted to the EAV triple $\langle$ "Maenads", `quality`, "cruel" $\rangle$.

Definition 5.2 (rule matching)**.** A sentence s matches an extraction rule R, denoted $R \models s$, iff $(\forall\, node\, N \in CT_R\ N \in CT_s) \wedge (\forall\, edge\, E \in CT_R\ E \in CT_s)$.

In other words a sentence s matches a rule R if the constituent tree CT_R of the rule is a subgraph of the constituent tree CT_s of the sentence. Subtrees of CT_s rooted at nodes corresponding to data nodes of CT_R are considered as the pieces of information to be extracted.

Algorithm TAE($Domain$,$Sources$)
  $Domain$ is the domain of knowledge
  $Sources$ is a set of document sources
begin
  $\mathcal{D} := GetDocuments(Domain, Sources)$
  $\mathcal{E} := \emptyset$
  $\mathcal{T} := \emptyset$
  for each document $D \in \mathcal{D}$ **do**
    $T := Tokenize(D)$
    $T := DisambiguatePartOfSpeech(T)$
    $T := RecognizeCompoundForms(T)$
    $\mathcal{E} := \mathcal{E} \cup RecognizeNamedEntities(T)$
    $T := ResolvePronouns(T)$
    $\mathcal{T} := \mathcal{T} \cup \{T\}$
  end for
  for each document $T \in \mathcal{T}$ **do**
    for each sentence $s \in T$

```
            for each extraction rule R ∈ 𝓡 do
                if R ⊨ s then
                    for each extraction pattern ep ∈ R.EP do
                        ⟨ e, A, v ⟩ = ep(CT_s)
                        if exists E ∈ 𝓔 such that E.Name = e then
                            InsertIntoDatabase(⟨ e, A, v ⟩)
                        end if
                    end for
                end if
            end for
        end for
    end for
end.
```

The TAE algorithm listed above assumes the existence of various subroutines. We describe each of these below.

1. *GetDocuments*. This function retrieves all domain-specific documents from the specified data sources. Actually the Extraction Engine is implemented as a background process that continuously crawls data sources looking for new information. *GetDocuments* thus retrieves new documents (documents that have never been visited before) or documents that have been modified since the last visit. We implemented *GetDocuments* using the *keyword spices* approach [51] to recognize if a document is of interest to a given domain. Each domain is in fact associated with a set of keywords.

2. $Tokenize$. Each relevant document $D \in \mathcal{D}$ is then tokenized, i.e. fragmented into units corresponding to single words or punctuation marks, and each token is tagged with its corresponding part of speech. Thus, a token can be defined as a pair $(Word, PartOfSpeech)$. We use WordNet [47] for part of speech tagging. Sentence boundaries are also identified during tokenization.

3. $DisambiguatePartsOfSpeech$. A *part of speech disambiguation* algorithm is applied to resolve situations in which a word has been tagged with more than a single part of speech. We apply simple heuristic rules to address this issue, but more sophisticated algorithms have been proposed in the literature [54].

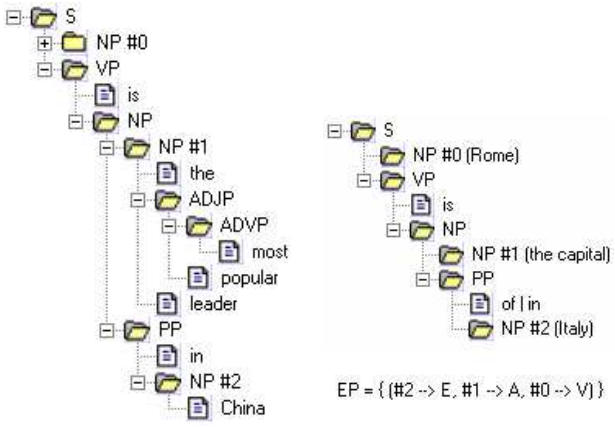
4. *RecognizeCompoundForms*. Next, we use WordNet to *identify compound forms*, such as verbs followed by prepositions or adverbs (eg. "take off", "get out"), and merge their tokens together.

5. *RecognizeNamedEntities*. A *named entity recognition* algorithm [6] is applied in order to identify and classify named entities (people, organizations, places, etc.) that appear in $\mathcal{D}$. This allows us to find entities of interest within the domain (e.g. all the Greek Mythology characters, all the people and organizations involved in nuclear research activities, etc.) and extract data about these entities in advance. A *named entity* can be defined as a tuple $(Name, Class)$, where $Class$ can be (i) person's name (`PN`), (ii) geographic location (`LN`), (iii) organization (`ON`), (iv) date/time (`DT`), (v) unclassified (`NC`), (vi) not an entity (`NaE`). In our implementation, we developed our own named entity recognition algorithm – this is described in Section 5.1.2. The set $\mathcal{E}$ consists of all recognized named entities (repetitions are removed).

6. *ResolvePronouns*. Finally, we *resolve pronouns* by discovering which entities previously named in a document the pronouns refer to. Many sophisticated algorithms [12] have been proposed for this task.

At this stage of the TAE algorithm, we get unambiguous versions of the source documents and a set $\mathcal{E}$ of recognized entities that are deemed to be of interest for the selected domain. We now extract data by applying a set $\mathcal{R}$ of extraction rules that allow us to deduce EAV triples from sentences. The algorithm iterates over all sentences from all documents. If a sentence matches a rule according to Definition 5.2 then an EAV triple is derived from the sentence for each extraction pattern in the rule. If the triple refers to one of the entities of interest to the specific domain – those in the set $\mathcal{E}$ – it is stored in the database.

Example 5.1. Consider the sentence "Hu Jintao is the most popular leader in China". Figure 5.2.a shows the constituent tree of the sentence, while Figure 5.2.b shows the extraction rule against which we are trying to match the sentence. It is clear from the picture that the sentence matches the



a b

Figure 5.2: Data extraction: (a) analyzed sentence; (b) matching rule

rule. Based on the extraction pattern of the rule we thus deduce the EAV triple ⟨ "China", `the most popular leader`, "Hu Jintao" ⟩. Further processing consisting of head noun identification [40, 60] allows to deduce also the simplified triple ⟨ "China", `leader`, "Hu Jintao" ⟩.

5.1.2 Named Entity Recognition

We have developed a named entity recognition algorithm which recognizes and classifies *named entities* in a document. A significant amount of work has been done on this topic. Some authors propose knowledge-based approaches [6] while others favor the use of statistical models such as Hidden Markov Models [24].

In this work, we propose the Tokenized-HMM (T-HMM) algorithm which uses two phases: in the first phase it uses HMMs to identify and classify all tokens that are part of a named entity. In the second phase, named entities are classified based on the classification of their tokens. Tokens associated with a single named entity may have different classifications – we resolve any conflicting classifications using three alternative approaches that we present in the following.

Simply stated, an HMM has a finite set of states, each of which has an associated probability distribution. Transitions among the states are

governed by a set of probabilities called transition probabilities. Each state generates an output based on the associated probability distribution. The observer can only see the outcome, not the state.

In our case the set of possible states coincides with the set of possible named entities classes mentioned earlier, i.e. the set {`PN`, `LN`, `ON`, `DT`, `NC`, `NaE`}. As the document is read, the HMM receives tokens from the document which may cause state changes to occur. Our algorithm considers not only the current token, but also the features of the previous and the next tokens: this has greatly improved the accuracy of recognition, since it takes into account some contextual information. Let us assume the following notation:

- N is the number of states;
- $V = \{v_1, ..., v_M\}$ is a finite set of M observation symbols;
- S_t is the state of the system at time t;
- $\pi = \{\pi_1, ..., \pi_N\}$ is the initial state vector with $\pi_i = P(i_1 = i)\ \forall i \in \{1, ..., N\}$; in other words π_i is the probability that the system is in the state i at time 1, i.e. when the first token is observed;
- $A = \{a_{ij}\}$ is the the state transition probabilities matrix, with $a_{ij} = P(i_{t+1} = j | i_t = i)$;
- $B = \{b_j(k)\}$ is the probability distribution of observation symbols, with $b_j(k) = P(v_k \text{ in } t | i_t = j)$;
- O_t is the observed symbol at time t, that in our case consists of the observable features of the t-th token and, eventually, of the surrounding tokens.

A Hidden Markov Model is a triple $\lambda(A, B, \pi)$. We wish to identify the most likely state sequence corresponding to the observed features:

Problem 5.1. Given an Hidden Markov Model $\lambda(A, B, \pi)$, find a state sequence $S = S_1, S_2, ..., S_T$ such that the joint probability $P(O, S|\lambda)$ of S and the observation sequence $O = O_1, O_2, ..., O_T$ is maximized w.r.t. the model.

Algorithm T-HMM(T)
T is a tokenized document
begin
$\mathcal{E} := \emptyset$
$O := ExtractFeatures(T)$
$S := Viterbi(O, \lambda(A, B, \pi))$
for each j, m such that $S[j-1] = \texttt{NaE} \wedge S[j, \ldots, j+m] \neq \texttt{NaE} \wedge S[j+m+1] = \texttt{NaE}$ **do**
$Name := Merge(T[j].Word, \ldots, T[j+m].Word)$
$Class := Select(S[j], \ldots, S[j+m])$
$\mathcal{E} := \mathcal{E} \cup \{(Name, Class)\}$
$T := Replace(T, j, j+m, (Name, Class))$
end for
return $\mathcal{E}$
end.

Phase I: Above, you can see the T-HMM algorithm for recognizing named entities. This algorithm takes as input a tokenized document T and builds the sequence of observation symbols using a function that extracts features from the tokens. The current implementation of the feature extraction function returns an array of 18 boolean features which include, for example `CityNameSuffix` (e.g. "Hyattsville" has a city name suffix). We have also considered two more solutions in which the features of the previous token (bigram configuration) and of both previous and next tokens (trigram configuration) are taken into account. The Viterbi algorithm [50] is a well-known approach to solving Problem 5.1. It finds the state sequence S that maximizes the joint probability of the observation sequence O and state sequence given the model. In order to learn the parameter A, B and π of the model, the system has been trained on text documents randomly selected from the *Brown Corpus* [18].

Phase II: Each sequence of tokens whose corresponding state is not `NaE` (i.e. not an entity) undergoes further processing to be identified. The entity's name is clearly the concatenation of the words in each token, while its class is determined based on the classes of the component tokens. If tokens have been assigned to different classes (let C be the set of these classes) we need to select one of them. To this aim we have proposed three solutions.

- *Probabilistic classification*: selects the class in C that is the most likely

	R	$P_{probabilistic}$	P_{voting}	$P_{evolving}$
monogram	55,80 %	63,30 %	63,30 %	63,30 %
bigram	65,75 %	68,57 %	68,48 %	68,86 %
trigram	71,88 %	66,36 %	66,45 %	66,36 %

Table 5.1: Recall and precision performance of the Named Entity Recognition Algorithm

to produce the observation sequence corresponding to the sequence of tokens;

- *Voting classification*: selects the class that has been assigned to most of the tokens;
- *Evolving classification*: selects the class assigned to the last token, assuming that the precision of the recognition increases over time.

We now present an example of how the proposed algorithm works.

Example 5.2. Consider a document D with a single sentence "West Palm Beach's mayor is T. J. Smith". After the first step of the algorithm we obtain the following classification: ⟨ ("West",`ON`), ("Palm",`ON`), ("Beach",`LN`), ("'s",`NaE`), ("mayor",`NaE`), ("is",`NaE`), ("T.",`PN`), ("J.",`PN`), ("Smith", `PN`) ⟩.

In the second phase "T. J. Smith" is correctly classified as a person name using all the three approaches, and "West Palm Beach" is correctly classified as a location name with both the probabilistic and evolving approach, while is wrongly classified as an organization name with the voting approach.

Table 5.1 shows the results in terms of recall (the percentage of identified named entities) and precision (the percentage of correctly classified entities) that we have obtained for each configuration and for each class selection strategy. The results are comparable with the ones described in the literature. Since we are mainly interested in recall, the trigram configuration seems to be the most promising.

5.1.3 Attribute Extraction from Relational and XML Sources

First, let us consider a relational table $T = \{c_1, \ldots, c_m, \ldots, c_n\}$ where $c_1, \ldots, c_n$ are columns and $c_1, \ldots, c_m$ are also keys. Then for each two

columns c_i, c_j such that $1 \leq i \leq m$ and $1 \leq j \leq n$ we add the following entry to the source access table:

$$\langle\, c_j,\ T : c_i,\ f_{c_j,T:c_i}(e) = \pi_{c_j}\sigma_{c_i=e}T \,\rangle.$$

In other words, given a table T as the source, we obtain a value for an attribute c_j of an entity e by looking for all table rows that can be referred by e.

It is also possible to extract attribute values from XML sources. Consider an XML node

$$N = \langle\, name, value, \{c_1, \ldots, c_n\} \,\rangle$$

where $c_1, \ldots, c_n$ are children nodes. Assuming that N is a root node in an XML document, and nodes may act both as entities and attributes, one can write the following algorithm to return a given attribute of an entity.

Algorithm $GetXMLAttr(N,e,A)$
  N is the root XML node
  e is the entity
  A is the attribute
begin
  $Result := \emptyset$
  if $N.value = e$ or $N.name = e$ **then**
    for each child c of N such that $c.name = A$ **do**
      $Result := Result \cup \{c.value\}$
    end for
  else
    for each child c of N **do**
      $Result := Result \cup GetXMLAttr(c, e, A)$
    end for
  end if
  return $Result$
end.

The $GetXMLAttr()$ recursively finds all occurrences of an entity in the XML tree, collects all values of the requested attribute, and returns the collected set of values. Notice that the algorithm tries to match e to both *node value* and *node name*. We can now enumerate all the attribute names occurring in the XML tree as $A_1, \ldots, A_m$, and for each A_i, add a source access table entry $\langle\, A_i,\ N,\ GetXMLAttr(N, e, A_i) \,\rangle$.

Part III

Experiments and Conclusions

Chapter 6

Video Summaries

6.1 Implementation

To evaluate the efficiency and effectiveness of PriCA, we have implemented a prototype system in JAVA on top of Oracle 8i and MS Access DBMS backends. The prototype is a complete video management framework that allows to both index and summarize videos, and consists of the PriCA algorithm implementation, as well as an implementation of CPRgen, CPRdyn, and SEA algorithms described in Chapter 3, and a user interface for specifying the desired summary content. In addition, the system is capable of automatically segmenting video into shots and detecting soccer-related events, for annotation purposes.

In a typical example of system's usage, the user selects a video he/she would like to process and the system checks whether this video is already indexed or not. In the latter case, the system will offer to index the video. Once the video is indexed, the user can modify the indexing, query the database to find specific blocks, or summarize the video.

Figure 6.1 shows the indexing/querying interface to a previously indexed video. In this example, the user asks the system to retrieve all the blocks in which the action *goal* occurs. The resulting set of blocks is listed at the left side of the interface and can be viewed through the video player in the top left corner of the interface.

Figure 6.2 shows the summarization interface. Using controls in this interface, the user specifies the desired summary features. The system allows

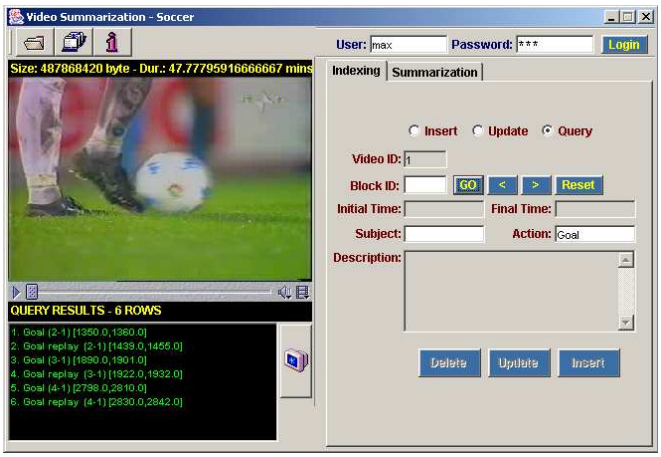


Figure 6.1: Indexing and query interface

the user to select any of the four algorithms, and interface controls change according to the selected algorithm.

Figure 6.2 shows an example of summary content specification. The blocks in the summary created by PriCA are listed on the left while Figure 6.3 shows a representative frame for each of the five resulting blocks.

Figure 6.2: Summarization interface

Note that in order to summarize a video, we can use data structures such as those in AVIS [1] to determine what activities and objects occur in frames. This will clearly speed up the algorithms within the PriCA framework.

Figure 6.3: Summarization result

6.2 Experimental Setting

In this section, we describe a set of experiments conducted to evaluate the performance of the PriCA system.

A key issue in automated summary construction is the evaluation of the quality of the summary with respect to the original data. There seems to be general consensus on the non-existence of some universally accepted solution – this is mainly due to the many different approaches to the video summarization problem.

A number of alternative approaches are thus available. Considering user based evaluation methods, a group of users is asked to provide an evaluation of the summaries or to accomplish certain tasks (i.e. answering questions) with or without the knowledge of the summary, thus measuring the effect of the summary on their performance. Alternatively, for summaries created using a mathematical criterion, the corresponding value can be used directly as a measure of quality. However, all these evaluation techniques present

several drawbacks; user-based ones are difficult and expensive to set-up and their bias is non trivial to control, whereas mathematically based ones are difficult to interpret and compare to human judgement.

We first compare the PriCA algorithm to the CPRgen, CPRdyn, and SEA algorithms proposed in [16] and described in Chapter 3, both in terms of time spent to compute a summary and quality of resulting summaries, then try to compare our results to the results of other authors. As the only effective way of evaluating quality of summaries is via human subjects, we enrolled a group of 200 people, mainly students from the University of Naples.

Our data set consisted of about 50 soccer videos, totaling about 80 hours. The videos were segmented into blocks and annotated, as described in section 3.3.2. The resulting blocks had an average length of about 10 seconds, with a relatively low variance.

6.3 Qualitative Evaluation

To assess the quality of the results produced by the four algorithms being compared, we asked the group of reviewers to rate the resulting summaries on a 1 to 5 scale. The experiment was repeated three times, with desired summary lengths of 2, 4, and 6 minutes respectively, for all videos. The results, shown in Figure 6.4, indicate that summaries produced by the PriCA algorithm have been rated best in 48%, 46%, and 45% of all cases respectively. These percentages are significantly better than those for the other three algorithms.

The comparison between the above mentioned algorithms has been made easier, since all the four algorithms are based on some common assumptions – segmentation into shots, summarization based on high-level descriptions, summaries as sequences of shots. However, things quickly become complex when we try to compare the results of totally different algorithms on different data sets in different domains.

We now discuss some evaluation results obtained by other researchers and compare them to our result, but this comparison has to be taken with a bit of salt for the above reasons.

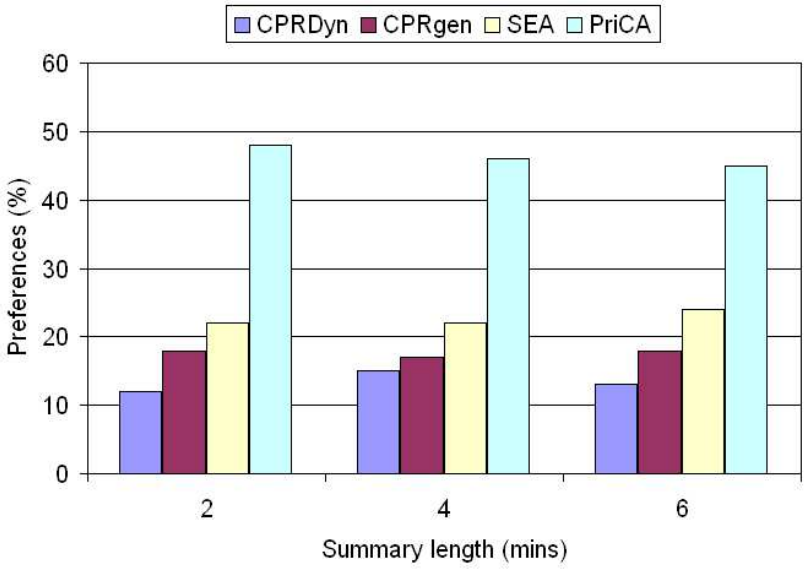


Figure 6.4: Summary quality ratings

In [58], Shao et al. propose an approach to automatically summarize musical videos, based on an analysis of both video and audio tracks. They evaluated the quality of the summaries through a subjective user study and compared the results with those obtained by analyzing either audio track only and video track only. The subject enrolled in the experiments rated *conciseness* and *coherence* of the summaries on a 1 to 5 scale. The *conciseness* parameters does not have a corresponding one in our framework, since users explicitly specify the desired length of the summaries in our framework. The coherence parameter is similar, though not equivalent, to our quality parameter. In conclusion, the average value of the coherence obtained in [58] is 4.57, while the value of the judged quality of summaries produced by PriCA is 4.46. Note that these two parameters are not immediately comparable. Shao et al. do not discuss execution times, but we can reasonably state that our algorithm is surely faster than theirs, because they have to analyze both audio and video tracks. It is important to note that this work only focuses on music videos, whereas our work is applicable to a any video. Just to consider another example, [38] reports an average value of 93.7 – on a 1 to 100 scale, it would be 4.47 on a 1 to 5 scale – for a *meaningfulness*

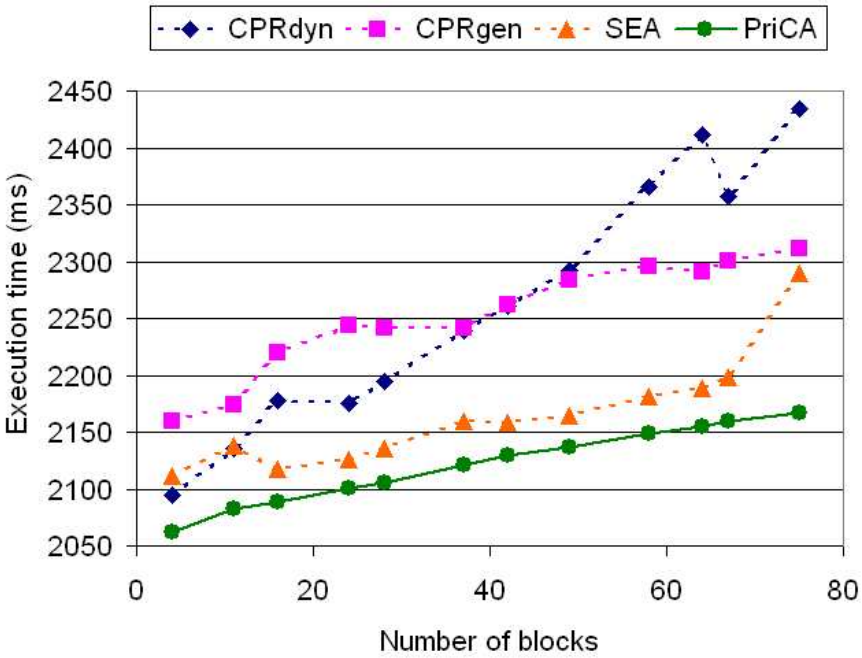


Figure 6.5: Summary creation times

parameter, that is evaluated through a user study too.

6.4 Execution Times

To assess performance, we fixed the desired length of the summary to 60 seconds. We then varied the number of candidate blocks in the 4-75 range, by choosing an increasing number of events and subjects of interest.

The processing times were computed for each algorithm by averaging the results of 10 executions for each video. Figure 6.5 shows times taken by different algorithms. From this figure, we can conclude that the PriCA algorithm outperforms the other three algorithms. This is true even without using the optimization for $Peaks()$ mentioned earlier.

Chapter 7

Story System Evaluation

7.1 Introduction

We have implemented a prototype STORY system and applied it to several scenarios, among which the creation of stories about Greek Mythology characters for the archaeological site of Pompeii, where many of those characters are depicted in paintings and sculptures. The system consists of two main components:

- an offline component, the *Extraction Engine*, continuously crawls the data sources specified by the application developer – selected web sites or the entire web, news feeds, etc. – in order to extract new information, in the form of EAV triples. Extracted triples can be stored both in XML/RDF format or into a relational database;
- an online component, the *Story Creation Engine*, creates stories upon user's request, using the information collected by the *Extraction Engine*.

The architecture of the system is thus similar to the architecture of any search engine, where a background process continuously crawls the web and indexes new discovered pages, while a real time process answers user queries by accessing the index rather than the actual web pages. The advantage of this architecture is that queries can be answered very fast. The obvious drawback is that the answers are limited to what is known to the system, i.e. the information that has been indexed. Likewise, we will show that

the STORY system can generate stories very fast, but the facts that can be included into a story are limited to those that have already been discovered.

This chapter describes the experiments we run in order to validate our approach and verify that high quality stories can be efficiently constructed and delivered to the users, over both wired and wireless networks to multiple heterogeneous devices. We evaluated both the quality of stories produced by our algorithms (Section 7.2) and the time taken to construct them (Section 7.3).

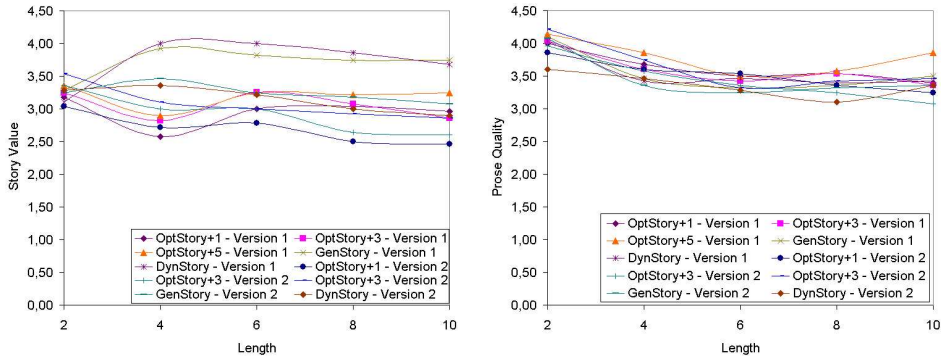
7.2 Story Quality

7.2.1 Experimental Setting

In Chapter 3 we pointed out that a key issue in automated summary construction is the evaluation of the quality of the summary with respect to the original data, but there seems to be general consensus on the non-existence of some universally accepted solution. When no metric can be defined to measure the quality of the produced summaries, user based evaluation methods are the most feasible solution. The same considerations apply to automatic story creation.

We thus evaluated the quality of the stories produced by our system through human ratings. 61 humans, mainly students from the University of Naples, were enrolled in the experiments. They were subdivided into two groups: (i) 10 experts and (ii) 51 non-experts. They were asked to review stories about the following Greek Mythology characters: Pentheus, Cadmus, Apollo, Agave, Semele, and Dionysus. Before starting the experiments, the expert group was asked to read some documentation about the selected subjects, while non-experts were not given any a priori knowledge about the subjects of the stories.

Each reviewer was asked to rate the story value (in terms of included facts) as well as the quality of the narrative prose on a scale from 1 (worst) to 5 (best). We evaluated five algorithms, namely OptStory$^+$ with branching factor 1,3 and 5 respectively, GenStory and DynStory. To the aim of the experimentation we adopted a story evaluation function *eval* based on



a b

Figure 7.1: Non-expert reviewers: (a) Story Value and (b) Prose Quality

the CPR criteria. Reviewers were presented with stories of different lengths, computed with five different algorithms, varying continuity, priority and repetition parameters. In addition, we rendered stories in two ways: (1) using sentences from textual sources whenever available and (2) using templates only. We were interested in determining which algorithm and which rendering strategy performs best. A total of different 200 stories were created for each subject, making the evaluation process highly time-consuming and limiting the number of story subjects (lest the reviewers become bored and inaccurate).

7.2.2 Non-Expert Reviewers

Figure 7.1.a shows the value of the story as judged by human subjects as a function of story length for five different algorithms. Figure 7.1.b shows the quality of the prose in the story as perceived by the reviewers. Version 1 refers to the case where sentences from the sources are used, while version 2 refers to the case where templates are used.

Which algorithm produces the best story value?

- DynStory algorithm yields the highest story value, while algorithm GenStory is the second best.
- Both DynStory(1) and GenStory(1) algorithms are consistently better than algorithms DynStory(2) and GenStory(2). This is due to the fact

that including original sentences into the story adds more information than rendering the same facts through templates.

- The prose quality (as assessed by human readers) is almost the same for all algorithms and decreases slightly as the story length increases. This is to be expected, given that all algorithms use the same mechanism to render actual stories.

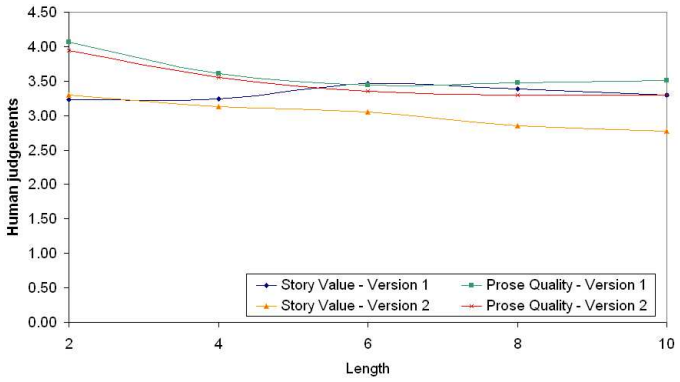


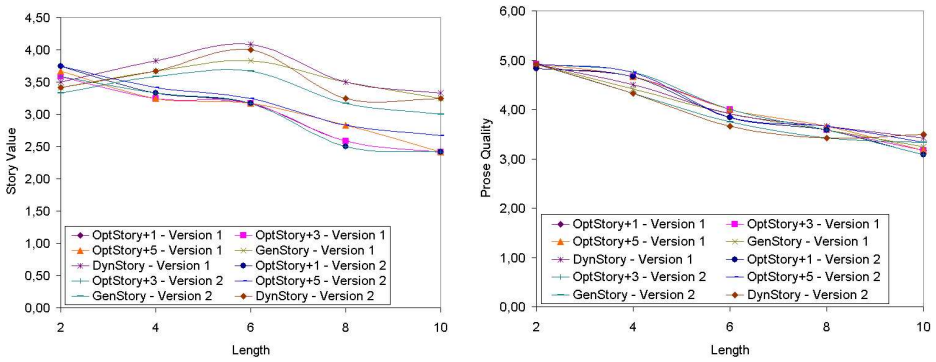
Figure 7.2: Non-expert reviewers: average Story Value and Prose Quality

How does using source sentences compare with using templates? Let us now consider the graph in Figure 7.2. The curves have been obtained by averaging, for each rendering technique, the results of all five algorithms. The y-axis in this graph shows the quality of the story as judged by human subjects on a 1 to 5 scale. The graph indicates that using source sentences significantly improves the story value as perceived by a human, but only slightly improves the prose quality over the template rendering method.

7.2.3 Expert Reviewers

We now present the results of experiments with expert reviewers and point out the difference from the non-expert group's results.

Which algorithm produces the best story value? Figures 7.3.a and 7.3.b were obtained in the same way as Figures 7.1.a and 7.1.b, but for the expert group of reviewers. These figures allow to make the following observations:



a b

Figure 7.3: Expert reviewers: (a) Story Value and (b) Prose Quality

- DynStory and GenStory algorithms ensure the best story value, same as in the case of non-experts. However, in this case their source-based and temlplate-based versions are much closer than in the previous case. This may be due to the fact that expert reviewers recognize that the rendered facts are the same in both versions.
- The prose quality is still the same for all algorithms, but it rapidly decreases with rising story length.

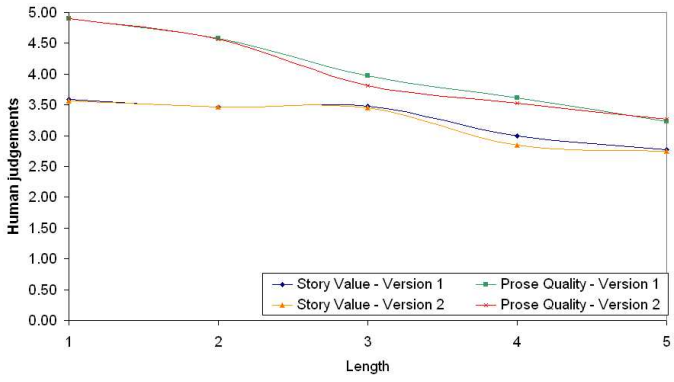


Figure 7.4: Expert reviewers: average Story Value and Prose Quality

How does using source sentences compare with using templates? Figure 7.4 shows the quality of a story as judged by human subjects on a 1 to 5 scale. Once again the curves have been obtained by averaging, for

each rendering technique, the results of all five algorithms. It shows that for expert reviewers, using source sentences does not improve the story value, while prose quality is slightly higher for templated-rendered stories.

How do experts compare with non-experts? Let us now compare results from expert and non-expert reviewers, as shown in Figure 7.5 (the y-axis still shows human subject judgements of the stories produced).

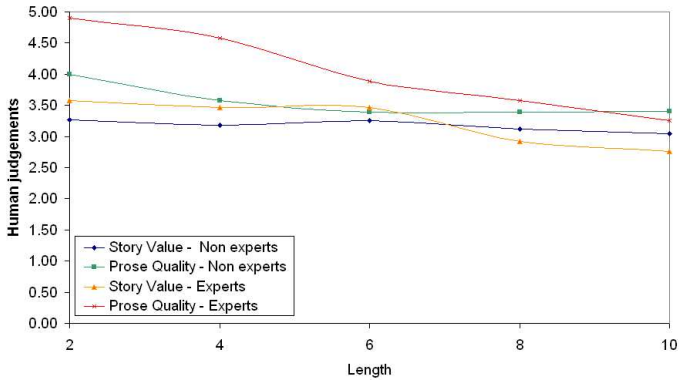


Figure 7.5: Experts vs. non-experts Comparison

- Experts rate short stories with higher value than non-experts.
- As stories become longer, experts rate their value lower, while ratings from non-experts remain almost the same.
- For all but very long stories, experts rate prose quality higher than non-experts. This may be due to the fact that they are more aware of the machine-generated story nature.

We used the *t-test* to analyze the statistical significance of our experiments. We first considered the non-experts and compared their judgements about both the value of stories and the quality of the prose. We obtained *tValues* between 0.4 to 0.5. Those numbers indicate no statistically significant difference in the sample means. We then applied the same analysis to the judgements of expert reviewers, obtaining similar results, and thus concluding that human judgments can be considered significant within groups of similar people. We then compared expert and non-expert judgements,

obtaining *tValues* greater than 1. The difference in the means became significant in some particular cases, such as judgements about the prose for low values of the story length (*tValues* greater than 3), thus confirming the interpretation of Figure 7.5.

7.3 Execution Times

Besides the qualitative evaluation we also evaluated the times taken to extract information from data sources and to create stories. In particular we compared the time taken to create a story of a given length k using the information already in the database and time needed to retrieve the same k facts from data sources.

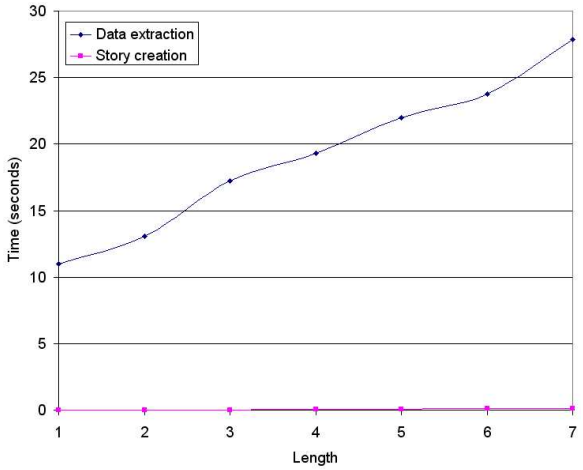


Figure 7.6: Comparison between execution times

Figure 7.6 reports the results of this comparison and clearly shows that the data extraction time is two orders of magnitude greater than the story creation time (tens of milliseconds vs seconds).

Note 7.1. The graph in Figure 7.6 shows for example that the time taken to extract 4 facts is 19 seconds, so one main think that it takes approximately 5 seconds to extract a single fact. These numbers are so large because we computed the time taken to access the sources and find the values of a given set of attributes, ignoring any other information in the same sources.

The actual behavior of the Extraction Engine is to extract all the available information – not just a single fact – every time it accesses a data source, thus the ratio of the processing time to the number of extracted triples drops down significantly.

Chapter 8

Discussion and Conclusions

8.1 Conclusions

In this work techniques and algorithms to extract and summarize information from large data repositories have been presented. Proposed techniques have been applied to different kinds of data. In particular two major scenarios have been considered throughout the thesis: digital video collections and collection of text documents (e.g. the world wide web). The choice of these two application domains has been motivated by the consideration that digital video data represent the most voluminous type of data in the field of multimedia databases, while the world wide web represents nowadays a huge and global information repository, counting billions of documents. Management of these classes of data thus arises several challenging research issues.

Starting from the consideration that both raw video data and natural language text are unstructured data, we have pointed out the need to define techniques for extracting structured data out of them. This would enable an information retrieval system to effectively answer queries that ask for specific information rather than for documents containing specific information. We have also pointed out that the ever increasing amount of available data causes information retrieval systems to produce very large answers to any user query, thus requiring a new capabilities for any modern information retrieval system, namely the capability of automatically summarizing large data sets in order to produce compact overviews of them.

In the video databases context, we have shown that the knowledge extraction phase requires the segmentation of videoclips into meaningful units (shots and scenes) and the identification of events occurring and objects appearing in each unit, while, the summarization task requires the selection of a subset of those units, such that certain constraints (e.g. the maximum allowed length for the summary) and properties (e.g. continuity and no repetitions) are satisfied.

In the context of text documents we have proposed a technique to extract structured information from natural language text and use such information to build succinct stories about people, places, events, etc., such that certain constraints and properties are satisfied.

The major contributions presented in this thesis have been

- the Priority Curve Algorithm (PriCA) for video summarization;
- the Text Attribute Extraction (TAE) Algorithm for extracting structured information from natural language text;
- a Named Entity Recognition algorithm (T-HMM) for recognizing in a set of text documents the entities of interest to a given knowledge domain;
- three heuristic algorithms (OptStory$^+$, GenStory and DynStory) for generating stories out of the information collected by the TAE algorithm.

The Priority Curve Algorithm (PriCA) retains the core ideas on which the CPR model is based but uses a completely different approach to the problem of finding good summaries fast. It completely eliminates the objective function upon which the previous algorithms were based, but captures the same intuitions in a compelling way. Experiments have shown that PriCA outperforms the three algorithms presented in [17] both in terms of user judged quality and in the terms of processing time.

The Text Attribute Extraction (TAE) Algorithm allows to extract structured information from natural language text. It is based on the concept of *extraction rule* that allows to match the syntactic structure of a sentence against a set of prototypes.

The Named Entity Recognition algorithm (T-HMM) allows to recognize in a set of text documents the entities of interest to a given knowledge domain. It is based on Hidden Markov Models and uses an innovative two-phase processing to recognize and classify named entities. Performance of the proposed algorithm have proved to be comparable with those of other algorithms presented in the literature.

The three heuristic story creation algorithms, namely OptStory$^+$, GenStory and DynStory, allows to build suboptimal stories using the information collected by the TAE algorithm. Our approach to story creation is different from both the text summarization approaches and the automatic storytelling approaches described in the literature. Experiments have shown that the STORY system can generate stories very fast (tens of milliseconds) and quality of produced stories is rated positively from the users, both in terms of value of the facts included in the story and in the terms of readability of the generated text.

8.2 Future Work

Several research issues still need to be further investigated with respect to both video summarization and automatic story creation. We plan to move forward in several new directions.

A key area of research on video summarization that we have mentioned in this thesis, but not described in detail, relates to the problem of actually summarizing video using a mix of audio, text and raw video streams associated with a video file. For example, news reports are often accompanied by text streams as well as audio streams and these streams can be invaluable in eliciting content.

Another major research topic relates to the problem of summarizing video based on context. In our approach, we can summarize video based on priorities – however, a number of methods can be used to express those priorities and often to automatically learn those priorities so that the summaries shown to different users are different.

The first goal in the field of story creation is to quantitatively define

what constitutes a good story. We studied numerous definitions in literature – most of them are behavioral definitions that describe the quality of a story in terms of its impact on its readers (e.g. *"a story is good if I can't stop reading it and when I do stop reading it, I can't stop thinking about it"*). Coming up with quantitative models of such statements is a formidable challenge. Our second goal is to further improve our algorithms to extract *entity attribute value* triples from data sources. A third goal is to assess how best to weight the conditions of continuity, non-repetition and priority of facts within a given story.

Bibliography

[1] S. Adali, K.S. Candan, S.-S. Chen, K. Erol, and V.S.Subrahmanian. The Advanced Video Information System: Data Structures and Query Processing. *ACM Multimedia Systems Journal*, 4(4):172–186, 1996.

[2] R. Agrawal, Jr. R.J. Bayardo, and R. Srikant. Athena: Mining-Based Interactive Management of Text Database. In *Proceedings of the 7th International Conference on Extending Database Technology: Advances in Database Technology*, volume 1777 of *Lecture Notes In Computer Science*, pages 365–379. Springer-Verlag, March 2000.

[3] M. Albanese, A. Chianese, V. Moscato, and L. Sansone. A Formal Model for Video Shot Segmentation and its Application via Animate Vision. *Multimedia Tools and Applications Journal*, 24(3):253–272, December 2004.

[4] N. Ancona, G. Cicirelli, A. Branca, and A.Distante. Goal Detection in Football by Using Support Vector Machines for Classification. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, volume 1, pages 611–616, July 2001.

[5] G. Boccignone, A. Chianese, V. Moscato, and A. Picariello. Foveated Shot Detection for Video Segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(3):365– 377, March 2005.

[6] J. Callan and T. Mitamura. Knowledge-Based Extraction of Named Entities. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, pages 532–537. ACM Press, November 2002.

[7] J.Y. Chen, C.M. Taskiran, A. Albiol, C.A. Bouman, and E.J. Delp. ViBE: a Video Indexing and Browsing Environment. In *Proceedings of the SPIE Conference on Multimedia Storage and Archiving Systems IV*, volume 3846, pages 148–164, August 1999.

[8] A. Chianese, R. Miscioscia, V. Moscato, S. Parlato, and A. Picariello. A Fuzzy Approach to Video Scene Detection and its Application for Soccer Matches. In *Proceedings of the 4th International Conference on Intelligent Systems Design and Application*, August 2004.

[9] R. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, and V. Zue, editors. *Survey of the State of the Art in Human Language Technology*. Studies in Natural Language Processing. Cambridge University Press, March 1998.

[10] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, September 2001.

[11] J.-P. Courtiat, R. Cruz de Oliveira, and L.F. Rust da Costa Carmo. Towards a New Multimedia Synchronization Mechanism and its Formal Specification. In *Proceedings of the Second ACM International Conference on Multimedia*, pages 133–140. ACM Press, 1994.

[12] I.L. de Oliverira and R.S. Wazlawick. A Modular Connectionist Parser for Resolution of Pronominal Anaphoric References in Multiple Sentences. In *Proceedings of the IEEE International Joint Conference on Neural Networks, IEEE World Congress on Computational Intelligence*, volume 2, May 1998.

[13] D. DeMenthon, V. Kobla, and D. Doermann. Video Summarization by Curve Simplification. In *Proceedings of the sixth ACM international conference on Multimedia*, pages 211–218, New York, NY, USA, September 1998. ACM Press.

[14] H.P. Edmundson. New Methods in Automatic Extracting. *Journal of the ACM*, 16(2):264–285, April 1969.

[15] W.E. Farag and H. Abdel-Wahab. *Multimedia Systems and Content-Based Image Retrieval*, chapter Video Content-Based Retrieval Techniques, pages 114–154. 2004.

[16] M. Fayzullin, V. S. Subrahmanian, A. Picariello, and M. L. Sapino. The CPR Model for Summarizing Video. *Multimedia Tools and Applications*, 26(2):153–173, June 2005.

[17] M. Fayzullin, V.S. Subrahmanian, A. Picariello, and M.L. Sapino. The CPR Model for Summarizing Video. In *Proceedings of the 1st ACM In-*

ternational Workshop on Multimedia Databases, pages 2–9. ACM Press, November 2003.

[18] W.N. Francis and H. Kucera. Brown Corpus Manual - A Standard Corpus of Present-Day Edited American English. http://helmer.aksis.uib.no/icame/brown/bcm.html, 1979.

[19] J. Goldstein, M. Kantrowitz, V. Mittal, and J. Carbonell. Summarizing Text Documents: Sentence Selection and Evaluation Metrics. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 121–128. ACM Press, August 1999.

[20] J. Goldstein, V. Mittal, J. Carbonell, and J. Callan. Creating and Evaluating Multi-document Sentence Extract Summaries. In *Proceedings of the Ninth ACM International Conference on Information Knowledge Management*, pages 165–172, November 2000.

[21] Y. Gong and X. Liu. Video Summarization Using Singular Value Decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 174–180, 2000.

[22] Y. Gong and X. Liu. Generic Text Summarization Using Relevance Measure and Latent Semantic Analysis. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 19–25. ACM Press, September 2001.

[23] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Prentice Hall, 2nd edition, January 2002.

[24] Z. GuoDong and S. Jian. Integrating Various Features in Hidden Markov Model Using Constraint Relaxation Algorithm for Recognition of Named Entities Without Gazetteers. In *Proceedings of the International Conference on Natural Language Processing and Knowledge Engineering*, pages 465–470, October 2003.

[25] U. Hahn and I. Mani. The Challenges of Automatic Summarization. *IEEE Computer*, 33(11):29–36, November 2000.

[26] A. Hampapur, T. Weymouth, and R. Jain. Digital Video Segmentation. In *Proceedings of the Second ACM International Conference on Multimedia*, pages 357–364. ACM Press, October 1994.

[27] A. Hanjalic and H.-J. Zhang. Optimal Shot Boundary Detection Based on Robust Statistical Models. In *Proceedings of IEEE International Conference on Multimedia Computing and Systems*, volume 2, pages 710–714, June 1999.

[28] F. Idris and S. Panchanathan. Review of Image and Video Indexing Techniques. *Journal of Visual Communication and Image Representation*, 8(2):146–166, June 1997.

[29] L. Itti and C. Koch. Computational Modeling of Visual Attention. *Nature Reviews Neuroscience*, 2(3):194–203, March 2001.

[30] H.M. Jamil and L.V.S. Lakshmanan. A Declarative Semantics for Behavioral Inheritance and Conflict Resolution. In *Proceedings of the International Logic Programming Symposium*, pages 130–144, 1995.

[31] H. Jing and K.R. McKeown. The decomposition of Human-Written Summary Sentences. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 129–136. ACM Press, August 1999.

[32] S.X. Ju, M.J. Black, S. Minneman, and D. Kimber. Summarization of Videotaped Presentations: Automatic Analysis of Motion and Gesture. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(5):686–696, September 1998.

[33] H. Kang. *Distributed Multimedia Databases: Techniques and Applications*, chapter Video Abstraction Techniques for a Digital Library, pages 120–132. 2002.

[34] F.W. Lancaster. *Information Retrieval Systems: Characteristics, Testing and Evaluation.* John Wiley, New York, 1968.

[35] S.-W. Lee, Y.-M. Kim, and S.W. Choi. Fast Scene Change Detection Using Direct Feature Extraction from MPEG Compressed Videos. *IEEE Transactions on Multimedia*, 2(4):240–254, December 2000.

[36] R. Lienhart, S. Pfeiffer, and W. Effelsberg. The MoCA Workbench: Support for Creativity in Movie Content Analysis. In *Proceedings of the Third IEEE International Conference on Multimedia Computing and Systems*, pages 314–321, June 1996.

[37] C.-C. Lo and S.-J. Wang. Video Segmentation Using a Histogram-Based Fuzzy C-Means Clustering Algorithm. In *Proceedings of the 10th IEEE International Conference on Fuzzy Systems*, pages 920–923, December 2001.

[38] S. Lu, M.R. Lyu, and I. King. Semantic video summarization using mutual reinforcement principle and shot arrangement patterns. In *Proceedings of the 11th International Multimedia Modelling Conference*, pages 60–67, January 2005.

[39] Y. Luo, J. Hwang, and T. Wu. *Multimedia Systems and Content-based Image Retrieval*, chapter Object-based Video Analysis and Interpretation, pages 182–199. 2004.

[40] A. Taylor M. Marcus and R. MacIntyre. The Penn Treebank Project. http://www.cis.upenn.edu/ treebank/.

[41] Y.-F. Ma, L. Lu, H.-J. Zhang, and M. Li. A User Attention Model for Video Summarization. In *Proceedings of the Tenth ACM International Conference on Multimedia*, pages 533–542. ACM Press, December 2002.

[42] I. Machado, R. Prada, and A. Paiva. Bringing Drama into a Virtual Stage. In *Proceedings of the Third International Conference on Collaborative Virtual Environments*, pages 111–117. ACM Press, 2000.

[43] I. Mani. *Automatic Summarization*, volume 3 of *Natural Language Processing*. John Benjamins Publishing Company, 2001.

[44] B.S. Manjunath and W.Y. Ma. Texture Features for Browsing and Retrieval of Image Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–842, August 1996.

[45] O. Marques and B. Furht. *Distributed multimedia databases: techniques & applications*, chapter Content-Based Visual Information Retrieval, pages 37–57. 2002.

[46] J. Meng, Y. Juan, and S.-F. Chang. Scene Change Detection in a MPEG Compressed Video Sequence. In *Proceedings of the SPIE Symposium on Digital Video Compression: Algorithms and Technologies*, volume 2419, pages 14–25, February 1995.

[47] George A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, November 1995.

[48] J.L. Mitchell, W.B. Pennebaker, C.E. Fogg, and D.J. Legall, editors. *MPEG Video Compression Standard*. Chapman & Hall, Ltd., 1996.

[49] V. Mittal, M. Kantrowitz, J. Goldstein, and J. Carbonell. Selecting Text Spans for Document Summaries: Heuristics and Metrics. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and The Eleventh Annual Conference on Innovative Applications of Artificial Intelligence*, pages 467–473. AAAI, September 1999.

[50] D. Neuhoff. The Viterbi Algorithm as an Aid in Text Recognition. *IEEE Transactions on Information Theory*, 21(2):222–226, March 1975.

[51] S. Oyama, T. Kokubo, and T. Ishida. Domain-Specific Web Search with Keyword Spices. *IEEE Transactions on Knowledge and Data Engineering*, 16(1):17–27, January 2004.

[52] W. B. Pennebaker and J.L. Mitchell. *JPEG Still Image Data Compression Standard*. Kluwer Academic Publishers, 1992.

[53] D.R. Radev, W. Fan, and Z. Zhang. WebInEssence: A Personalized Web-Based Multi-Document Summarization and Recommendation System. In *Proceedings of the NAACL Workshop on Automatic Summarization*, pages 79–88, June 2001.

[54] P. Rosso, F. Masulli, and D. Buscaldi. Word Sense Disambiguation Combining Conceptual Distance. In *Proceedings of the International Conference on Natural Language Processing and Knowledge Engineering*, pages 120–125, October 2003.

[55] Y. Rui, T. Huang, and S. Mehrotra. Browsing and Retrieving Video Content in a Unified Framework. In *Proceedings of the IEEE Second Workshop on Multimedia Signal Processing*, pages 9–14, December 1998.

[56] H.S. Sawhney and S. Ayer. Compact Representations of Videos Through Dominant and Multiplemotion Estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):814–830, August 1996.

[57] C.E. Shannon and W. Weaver. *The Mathematical Theory of Communication*. University of Illinois Press, Urbana, October 1963.

[58] X. Shao, C. Xu, and M.S. Kankanhalli. Automatically Generating Summaries for Musical Video. In *Proceedings of the International Conference on Image Processing*, volume 2, pages 547–500, September 2003.

[59] M. Shneier and M. Abdel-Mottaleb. Exploiting the JPEG Compression Scheme for Image Retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):849–853, August 1996.

[60] D.D. Sleator and D. Temperley. Parsing English with a Link Grammar. In *Proceedings of thr Third International Workshop on Parsing Technologies*, August 1993.

[61] K. Sparck Jones. *Automatic Keyword Classification for Information Retrieval*. Butterworths, London, 1971.

[62] V.S. Subrahmanian. *Principles of Multimedia Database Systems*. Morgan Kaufmann.

[63] M.J. Swain and D.H. Ballard. Color Indexing. *International Journal of Computer Vision*, 7(1):11–32, November 1991.

[64] N. Szilas. A New Approach to Interactive Drama: from Intelligent Characters to an Intelligent Virtual Narrator. In *Proceedings of the AAAI Spring Symposium on AI and Interactive Entertainment*, pages 72–76, 2001.

[65] M. Theune, S. Faas, A. Nijholt, and D. Heylen. The Virtual Storyteller: Story Creation by Intelligent Agents. In *Proceedings of the Conference on Technologies for Interactive Digital Storytelling and Entertainment*, pages 204–215, 2003.

[66] Y. Tonomura. Video Handling Based on Structured Information for Hypermedia Systems. In *Proceedings of the ACM International Conference on Multimedia Information Systems*, pages 333–344. McGraw-Hill, Inc., 1991.

[67] S. Uchihashi and J. Foote. Summarizing Video using a Shot Importance Measure and a Frame-Packing Algorithm. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 6, pages 3041–3044, March 1999.

[68] M. Umaschi Bers, E. Ackermann, J. Cassell, B. Donegan, J. Gonzalez-Heydrich, D.R. DeMaso, C. Strohecker, S. Lualdi, D. Bromley, and

J. Karlin. Interactive Storytelling Environments: Coping with Cardiac Illness at Boston's Children's Hospital. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 603–610. ACM Press/Addison-Wesley Publishing Co., April 1998.

[69] N. Vasconcelos and A. Lippman. Towards Semantically Meaningful Feature Spaces for the Characterization of Video Content. In *Proceedings of the 1997 International Conference on Image Processing*, volume 1, pages 25–28. IEEE Computer Society, October 1997.

[70] A. Vellaikal W. Zhou and C.C. Jay Kuo. Rule-Based Video Classification System for Basketball Video Indexing. In *Proceedings of the 200 ACM Workshops on Multimedia*, pages 213–216. ACM Press, November 2000.

[71] H.D. Wactlar, T. Kanade, M.A. Smith, and S.M. Stevens. Intelligent Access to Digital Video: Informedia Project. *IEEE Computer*, 29(5):46–52, May 1996.

[72] I. Yahiaoui, B. Merialdo, and B. Huet. Generating Summaries of Multi-Episode Video. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, pages 611–614, August 2001.

[73] B.-L. Yeo and B. Liu. Rapid Scene Analysis on Compressed Video. *IEEE Transactions on Circuits and Systems for Video Technology*, 5(6):533–544, December 1995.

[74] H.J. Zhang, A. Kankanhalli, and S.W. Smoliar. Automatic Partitioning of Full-Motion Video. *Multimedia Systems*, 1(1):10–28, June 1993.

[75] W. Zhou and S. Dao. Combining Hierarchical Classifiers with Video Semantic Indexing Systems. In *Proceedings of the Second IEEE Pacific Rim Conference on Multimedia - Advances in Multimedia Information Processing*, pages 78–85. Springer-Verlag, October 2001.