

Modelling Legal Documents as Graphs

T. J. M. Bench-Capon , P. E. S. Dunne & G. Stamford

To cite this article: T. J. M. Bench-Capon , P. E. S. Dunne & G. Stamford (1997) Modelling Legal Documents as Graphs, Information and Communications Technology Law, 6:2, 103-120, DOI: [10.1080/13600834.1997.9965761](https://doi.org/10.1080/13600834.1997.9965761)

To link to this article: <https://doi.org/10.1080/13600834.1997.9965761>



Published online: 10 May 2010.



Submit your article to this journal [↗](#)



Article views: 19



View related articles [↗](#)



Citing articles: 1 View citing articles [↗](#)

Modelling Legal Documents as Graphs

T. J. M. BENCH-CAPON & P. E. S. DUNNE

Department of Computer Science, University of Liverpool, Liverpool L69 7ZF, UK

G. STANIFORD

Department of Computer Science, University College Chester, Chester CH1 4BJ, UK

ABSTRACT *Managing documents is at the heart of many computer systems designed to provide support for legal tasks. In this paper, we bring together a number of techniques developed for handling legal documents based on their representation as graphs. We first introduce the use of directed acyclic graphs for the representation of conventional linear documents, and then generalize this to the use of unrestricted graphs for the representation of hypertexts. We describe techniques for controlling the construction, modification and transformation of such documents, and illustrate these techniques with some sample applications.*

Introduction

Documents abound in the law. Almost every task involves the use of one or more documents, and these documents may be of a variety of kinds. So, if we are interested in providing computer support for legal tasks, we need an approach to the handling of documents. This paper describes one approach to handling documents—modelling them as graphs. In the Introduction, we make some observations on the requirements on an effective approach to modelling legal documents, and offer some general remarks as to how our graph theoretic approach can meet them. We then describe in detail how conventional paper-based documents can be represented as directed graphs. We demonstrate how different classes of documents can be represented within this formalism so as to bring out their differing logical structures. The usefulness of this is illustrated by showing how the resulting graph grammars can be used to control the construction and modification of documents, and by showing how they can be used to relate documents of different types, and transform between them. We generalize the notion of graph to accommodate what is becoming the standard electronic representation, hypertext, introducing the notion of a linearization of a hypertext, so as to represent a reading of a hyperdocument, or the production of a paper version of a selection of its material. Techniques for achieving linearizations according to desired criteria are described. We further illustrate the potential of our approach by describing two applications of the techniques: the presentation of documents on the World Wide Web and the preparation of briefs.

One feature of documents in the legal domain is their diversity. In Social Security Law, for example, we can find all of the following:

- legislation—both primary and secondary;
- procedures and guidelines issued to those charged with applying the law;
- handbooks and leaflets directed at those to whom the law applies, published both by the Department and various advisory organizations;
- commentaries written by legal scholars;
- case reports, both in full versions and in digests.

Typically, a legal task will involve using and relating several of these documents: the legislation must be understood in the light of cases which have been decided, commentaries need to be read in conjunction with the legislation on which they are commenting, and so on. An effective treatment of legal documents must therefore be able to relate documents of different types.

A second important feature of legal documents is that they have structure. Different classes of documents will have different structures typical of their class. In some cases—such as legislation—the structure is rather rigidly defined. In others it will be looser, but still present. This structure is important for the proper understanding of the text and it is therefore important that the representation of legal documents be capable of capturing the structures. This is an important defect in conventional Boolean keyword style retrieval systems: because they view the document simply as a succession of words and phrases they lose the notion of structure, and the possibility of relating documents according to their structure.

When a document is represented as a graph, the text is divided into meaningful units, which become the nodes of the graph, and the edges represent relationships between these text units. In a very simple form, applicable to any document, nodes could be paragraphs, and the edges could represent the ‘follows’ relationship. This is not, however, very interesting. The power of the formalism comes when we distinguish between types of nodes, and label the edges to represent different kinds of relationship between the textual units. This is what allows us to capture the structure of documents, by describing the different nodes that a given class of document can contain and the relations that can exist between them in the given class of document. Moreover, given graphs representing different documents we can combine them into a single graph by adding some linking edges, and these will show the interrelationships between the two documents. The above is a brief sketch of how modelling the documents as graphs enables us to fulfil the requirements noted above. In the next section we will describe our formalism in detail.

Representing document structures by directed graphs

Directed graphs are one of the most commonly used mechanisms for formally modelling the logical organization of a structured document. Examples of such models may be found in approaches such as hypertext (Stotts & Furuta, 1988; Brown, 1988; Conklin, 1987) in which documents are represented by arbitrary directed graphs; Koo (1989), Kimura and Shaw (1984), Delisle and Schwartz (1986) and Meyrowitz (1986) used directed acyclic graph structures, and Thomas *et al.* (1985), Christodoulakis *et al.* (1986) and Bertino *et al.* (1988), restricted to tree structures.

We employ the term *document graph* to refer to a graph-theoretic representation of a document. In such models a document is viewed as a collection of (textual) objects: a node of the graph corresponds to a particular object and edges in the graph describe logical relations between objects, e.g. that a particular section must precede another section. Nodes may also be labelled to describe the function of a textual object, e.g. that it is a section title or a definition, etc. The use of node labelling permits sections of the document to be compressed, so that, for example, a node which is labelled as a *table* can subsequently be expanded into a subgraph describing the table in terms of its constituent entries. Among the benefits of this technique is that it allows for the logical connections between sections of a document to be represented simply and directly. Moreover, the problem of encapsulating those documents having a particular logical structure becomes equivalent to specifying the class of document graphs whose linkage and labelling conventions correspond to this logical organization. This specification will essentially describe a set of constraints on the form of graph which can be used to represent a particular class of document.

Documents are, however, often modified, both when they are being written and when they are subsequently revised. This gives rise to the possibility that a graph, initially meeting the constraints describing the form of a document in a specific class, will cease to satisfy these after several modifications have been made. Koo (1989) proposed the concept of *graph modification rules* as a means of handling this problem. These rules, which are formally production rules of a graph grammar, are employed to control modifications to a document graph in order that it should reflect changes made to the underlying document: thus, rules may encapsulate how to modify the graph in the event of sections being added to or deleted from the document or rules may indicate how new logical links in the document structure are to be reflected in the document graph form. Such rules can provide a formal basis for a tool to support the construction and modification of documents.

The concepts introduced by Koo were developed in Bench-Capon and Dunne (1989), using as a starting point a formal definition of a *document graph* adapted from Koo (1989).

Definition 1. A *document graph* is a directed acyclic graph, $G(V, E)$. The nodes in V denote *objects* in a document and the edges in E depict logical connections between objects. Objects in V may have an associated *label*.

One important feature of such graphs is that there are only a fixed number of finite paths from source nodes to terminal points permitted by the graph structure. Each of these paths will represent a sensible reading of the final document, to be reflected in the computer representation used for amending it. The ability to capture several readings within the same abstract structure is important because legal documents are not often read straight through from beginning to end, but are rather read selectively to extract specific desired information.

In Bench-Capon and Dunne (1989), we addressed three principal issues:

- (1) *Document specification*: given a particular class of structured documents, define the associated class of document graphs.
- (2) *Modification systems*: given a class of document graphs, define graph modification rules appropriate to the class.

- (3) *Consistency*: define graph modification systems to ensure that these do not transform a 'valid' document graph to one that does not satisfy the class specification.

The consistency problem had been raised, in a very restricted sense, by Koo (1989), wherein a graph modification system was considered consistent if it preserved acyclicity. Bench-Capon and Dunne (1989) introduced the following ideas, in order to address these questions.

Definition 2. A document specification consists of a pair $DS = (C, Init)$. Here C is a finite set of constraints

$$C = \{C_1, C_2, \dots, C_k\}$$

where each C_i is a (computable) predicate on document graphs. *Init* is a set of initial document graphs. Given a document specification DS and a document graph G , G is said to *meet* the specification DS if and only if $G \in Init$ or $C_i(G)$ is true for each constraint C_i .

Definition 3. A graph modification system (or GMS) is a finite set

$$S = \{R_1, R_2, \dots, R_m\}$$

of graph modification rules. Each graph modification rule, R , is a triple $\langle P, G_l, G_r \rangle$ where P is a predicate on document graphs and G_l, G_r are document graphs. A rule $R = \langle P, G_l, G_r \rangle$ acts on a given document graph G as follows: if $P(G)$ is true and G contains G_l as a subdocument graph then G_l in G is replaced by the document graph G_r . In general, applying a rule R to a document graph G results in a new graph H . We say that G *yields* H (denoted $G \rightarrow H$) in this case. Similarly, if H results from repeated applications of rules to G we say that G *derives* H (denoted $G \rightarrow^* H$).

Definition 4. Let $DS = (C, Init)$ be a document specification. *Good* (DS) is the set of all document graphs which meet DS . Let S be a GMS. The *derivation set* of S is the set of graphs, $\Delta(S)$, defined by

$$\Delta(S) = \{H : \exists G \in Init \text{ such that } G \rightarrow^* H\}.$$

S is *consistent with respect to* DS if and only if $\Delta(S) \subseteq Good(DS)$, i.e. every document graph derived using S meets the specification DS .

In general, given a GMS acting on the initial graph of a specification there may be infinitely many new document graphs which can be derived by repeatedly applying the modification rules. In order for the GMS to be 'correct' each document graph which is derived using it should meet the specification. The *inconsistency problem* for graph modification systems is the following. Given $DS = (C, Init)$ a document specification and $S = \{R_1, \dots, R_m\}$ a GMS operating on the initial graphs of DS , is there a document graph, G , for which $G \in \Delta(S)$ and $G \notin Good(DS)$?

Theorem. (Bench-Capon and Dunne, 1989) The inconsistency problem is undecidable.

Thus, there is no effective algorithmic method of solving the inconsistency problem that will be correct for all possible inputs.

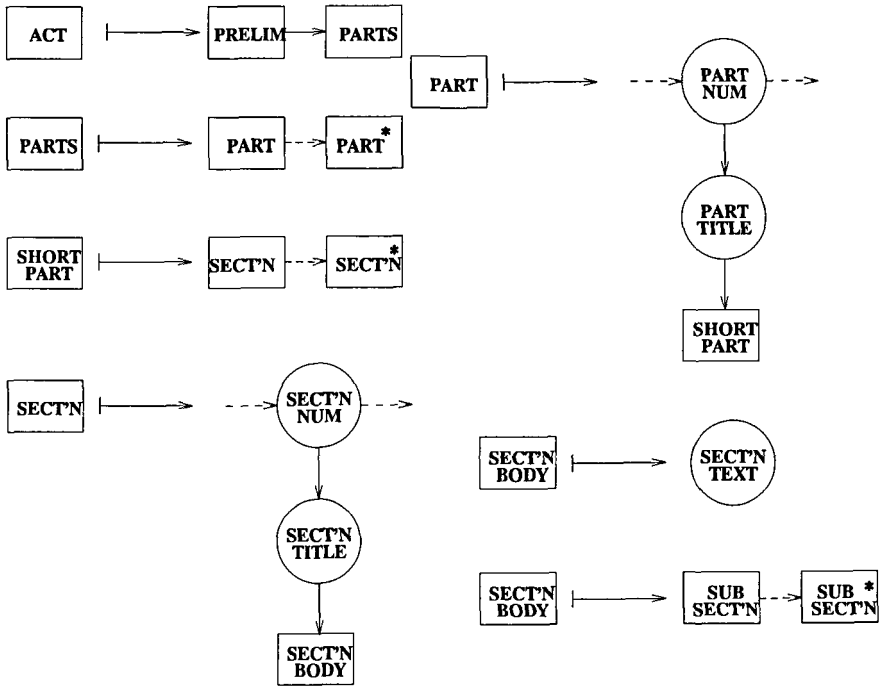


Figure 1. A partial graph modification system for an act.

This result, however, is largely of interest as a theoretical *caveat*: for the classes of document graph encountered in applications, one may construct provably-consistent modification systems, a number of examples being given by Bench-Capon and Dunne (1989).

This provides a formal basis for the specification of classes of documents with a common structure, rules for the modification of an instance of that class so as to preserve its membership of that class and a means of telling whether these rules are consistent. In the next section, we will show how we can use these notions in two example applications.

Example document specifications

Document specifications are usually predicated in a formal notation (see Stanford, 1994) using a set of source classes and a set of target classes. For reasons of clarity, however, we will use a more familiar notation in this paper. We can also express the structure using an extended Backus Naur Form (BNF) grammar. In the Appendix we present such a specification of a UK Statute. In this section, we present an example partial document modification system for an Act of Parliament which uses this grammar. This grammar is expressed diagrammatically in Fig. 1.

In Fig. 1 we show seven graphical predicates of the form $X| \rightarrow Y$ where, as above, we say that X yields Y . We see that **ACT** yields **PARTS** and both the source and the target nodes are represented as rectangular boxes to indicate expandable subgraphs. **PARTS** yields a **PART** followed by zero or more **PARTs**,

indicated by the dotted arrow and the asterisks in the second **PART** node. **PART** yields a chain consisting of the two unexpandable (ground) nodes **PART NUM** and **PART TITLE** (indicated by circles) plus an expandable subgraph **SHORT PART**. The expansion of **SHORT PART** is developed further in the remaining predicates. Notice that **PART NUM** has two dotted arrows connected to it which should be read to mean that 'if it is possible to replace the dotted arrows by solid arrows during an expansion then this must be done'. This indicates in the example that it is legal to add a **PART NUM** on the left, on the right, or to insert between other **PART NUM**s. There are two possible expansions of **SECT'N BODY** which indicates that either one or the other or both types of expansion are legally allowed within one document.

We now have the kernel of a rich and powerful set of predicate types with which to build complex directed acyclic graphs that, in conjunction with a relabelling function over **PART NUM**s may be used to create, maintain and access Acts of Parliament expressed electronically as directed acyclic graphs. Because it is represented as a graph, the system to do this can draw on all the well-understood graph theoretic algorithms for traversal and modification. For a detailed description of such a system, the reader is referred to Staniford (1994).

Transforming between document structures—the rapporteur system

In this section, we further illustrate the flexibility of graphs for representing documents by considering a particular application which involves extracting a document with a specific structure from a document with a different structure. The particular application is the production of a note of a discussion, but the principles involved apply equally to other types of summarizing activity.

A report of a discussion is a simple example of the more general document models described earlier. A *report graph*, $G_r(V_r, E_r)$, is a directed acyclic graph. The vertices in V_r denote *objects* in the report and the edges in E_r depict logical connections between the objects. Each object has an associated *object type* which consists of two parts: a *data type*, which specifies the domain of possible data values for the object (e.g. word, phrase, sentence, etc.); and an *attribute type*, which indicates the domain of possible properties that the object may possess (e.g. font, size, etc.). Objects may also be labelled. A *report specification* consists of a pair $RS = (C, Init)$, where C is a finite set of *constraints*,

$$C = \{C_1, C_2 \dots, C_k\}$$

where each C_i is a computable predicate on report graphs. *Init* is a set of *initial* report graphs. Given a report specification RS and a report graph G_r , G_r is said to meet the specification RS if and only if $G_r \in Init$ or $C_i(G_r)$ is **true** for each constraint C_i .

Report graphs are abstract representations of report structure: the form that is manipulated during the generation of the report from the raw dialogue graph. Our objective is to have this abstract representation closely matching the dialogue participant's conceptual representation of an accurately summarized discussion.

One common way in which people collaborate is through dialectical discussion. Dialectical discussion is a form of cooperation, between authors, whose object is the establishment of high degrees of confidence in the truth of some more or less doubtful propositions. It is abstract reasoning upon the basis of

propositions through categorization, definition, drawing out of implications and exposure of contradictions. Dialectical discussion involves the analysis and synthesis of fundamental terms in controversial questions: one person tries to establish a point while his colleague, either because of genuine scepticism or because he is playing devil's advocate for the purposes of the discussion, attempts to rebut the point. Such a discussion will help to structure the argument, clarify the position, and anticipate objections which require either additional exposition or refutation, or else which require the original position to be modified or withdrawn. A verbatim account of the discussion is, however, not the most useful form in which to record the discussion. Instead, it needs to be summarized so as to capture the essence of the argument that was developed. In our system, when such dialectical discussion occurs, it is recorded by an autonomous agent based program (Staniford, 1994) acting as a *rapporteur* whose responsibility is to synthesize what may be a rambling discussion into a coherent document setting out the thrust of the debate. *Rapporteur* is thus designed to support two or more colleagues collaborating through dialectic by producing a report of their discussion.

The discussion itself is in the form of a dialogue game, managed by the system, which progressively constructs a graph structure representing a dialectical argument. The notion of such games is discussed by Bench-Capon *et al.* (1992a). The particular game used by *rapporteur* is as follows. One participant must adopt the role of proposer, making an initial assertion and then taking turns to provide arguments in support of that assertion. The other participant adopts an opposition role in which challenges and objections to the proposer's assertion and supporting premises are put forward. *Rapporteur* allows counter objections and makes provision for both sides to modify earlier arguments. Either side can win the argument; in the case of the opposition being successful, the original assertion must either be negated or withdrawn.

Both sides take turn and turn about in presenting their respective cases, although one member of a side may take several consecutive turns for that side in order to present a particular line of thought. Game play takes place in a structured way which reflects the different roles that the two sides bring to the dialogue. The proposers are required to present an assertion; and are allowed to modify that assertion, provide supporting premises and modify those premises, refute objections from the opposition and require the opposition to continue objections and challenges. In their turn the opposition is allowed to challenge the assertion or premises, object to premises and modify those objections and requires the proposers to continue the assertion, premises and refutations. Either side may accept defeat: the opposition by accepting it has no valid challenge or objection, the proposers by accepting they have no valid refutation. *Rapporteur* oversees the game and will only allow legal moves to be made. The general dialogue graph model that is realized by *rapporteur* is presented in Fig. 2. Sequences of legal moves, representing particular argument graphs, can readily be worked out from this graph.

Rapporteur has no notion of the semantics of any argument: it provides a way of imposing a general syntactic structure on a dialogue represented by a graph. This structure is sufficiently flexible to allow the participants to conduct their dialogue using deduction, induction or indeed abduction as the mode of reasoning in their arguments.

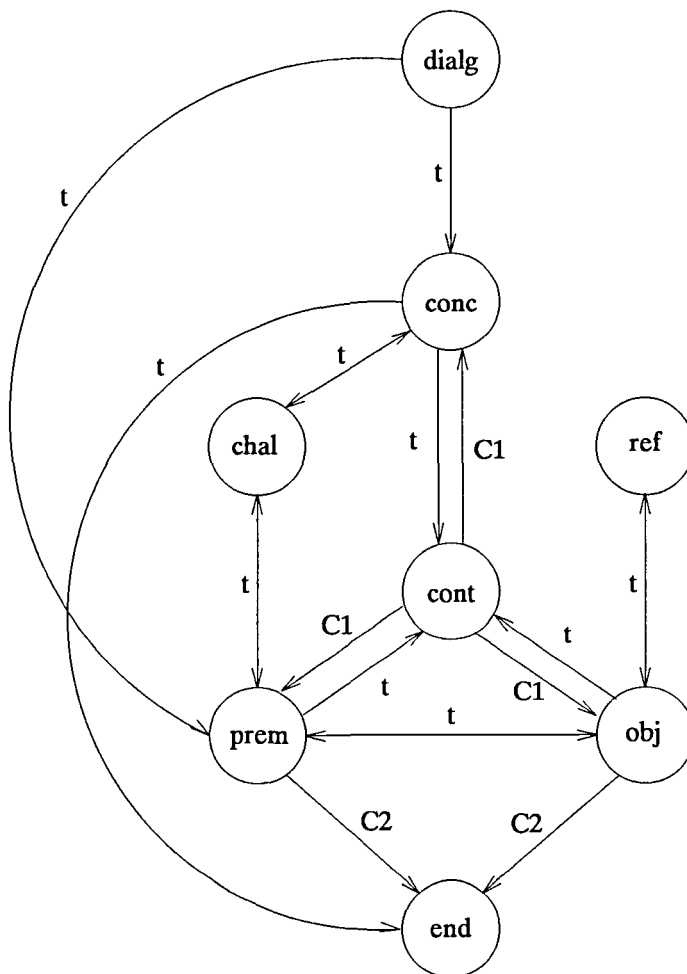


Figure 2. Rapporteur's dialogue control graph. The nodes are labelled with short forms of the moves in the game: dialogue, claim, challenge, premise, continue, objection, reference and end. *t* indicates an edge can always be followed: C1 is the condition that the last move was continue, and C2 the condition that player whose turn it is cannot make a move and so must concede the game.

We thus have two graphs, one—a directed cyclic graph—representing the realized dialogue space, and one—a directed acyclic graph—representing the model of a report of a dialectic discussion; both graphs containing single sources and sinks. The main task facing *rappporteur* is to transform the former into the latter. This will, for example, enable the digressions common in dialectic, such as when a person puts forward a definition which is found by a challenge to be inadequate, and which is consequently modified, to be elided so that the report will show only the final form of the definition, and the debate which led to the modification will be included in the justification of that definition.

To this end, during the course of a dialogue, *rappporteur* explicitly builds a set of nodes V_d , while implicitly following a set of edges E_d of the dialogue graph

$G_d(V_d, E_d)$. To achieve a mapping between the two graphs the agent adopts the strategy of partitioning V_d such that a set of candidate nodes, SP_d say, includes all nodes in V_d that lie on the shortest path through E_d , and then discards nodes that lie in $V_d \not\supseteq SP_d$. Depending on which side won the game, the agent partitions SP_d to produce V_r as follows:

- (a) Proposer wins—then V_r includes all nodes in SP_d that are premises and in addition the assertion node. $SP_d - V_r$ is discarded.
- (b) Opposition wins—then V_r includes all nodes in SP_d that are accepted premises, all nodes in SP_d that are successful objections and in addition a node containing the negation of the assertion node. $SP_d - V_r$ is discarded.

To complete the report graph $G_r(V_r, E_r)$ *rapporteur* produces E_r in accordance with the constraints present in the report specification.

We now give a highly simplified example of a dialectical discussion involving an *ad hominem* attack: based upon Walton (1985) in which he cites an example first discussed by Groarke (1982).

A_1 : Your government is subjecting dissidents to abuses that contravene the United Nations charter on human rights.

A_2 : What do you mean by abuses of human rights?

A_1 : Torture, for example.

A_2 : How can you say that? Your government is guilty of equally bad abuses of human rights.

A_1 : What do you mean by 'equally bad abuses'?

A_2 : Well you routinely torture all manner of prisoners: political and criminal.

A_1 : That I cannot deny.

This dialogue is summarized by *rapporteur* as:

The government of A_1 abuses human rights in a manner that contravenes the United Nations charter on human rights by using torture.

The government of A_2 is guilty of equally bad abuses of human rights by routinely torturing all manner of prisoners: political and criminal.

Therefore:

The governments of A_1 and A_2 are subjecting dissidents to abuses that contravene the United Nations charter on human rights.

The examples presented in this section have not been exhaustive, but we hope that they enable the reader to envisage how directed acyclic document graphs may be specified, used and managed, and to give a flavour of their potential for application. In the next section we consider a significant generalization to the work so far presented, as we go beyond directed acyclic graphs to richer structures.

Generalization to hypertext

Although the directed acyclic graph model, discussed in the previous section, provides considerable flexibility and has a number of advantages compared with more general structures, it has not been widely adopted in practice. Instead, the more intricate connectivity of hypertext representation has become a popular methodology for access to textual data. With this approach, arbitrary directed

graphs define the underlying structure and both nodes and links may be labelled, allowing for a richer set of relationships between the elements of the document. The labelled graph structure, modelling a hypertext, forms a semantic net.

The approach has a number of advantages. Importantly, it allows the integrated storage of a set of documents. For example, a set of case reports can be related to legislation by supplying appropriate links between the cases and the parts of the legislation on which they were decided. Similarly, commentaries can be integrated by linking their elements to the legislation and cases they discuss. In this way the whole body of documents can be seen as a single complex document.

Now, by following different paths through the graph, a browser of a hypertext can gain different perspectives on the material comprising the hyperdocument. The original documents can be extracted as sub-graphs, but following the links relating different documents we can effectively construct a new document incorporating related parts of several of the original documents.

Given this rich integrated structure we will often wish to read, or to print, a selection of nodes from the structure. For example, we might wish to extract all the legislation and cases relevant to a topic such as 'good cause for late claim' together with any commentary upon it. To do so the document nodes must be ordered into a sensible linear traversal. The production of such traversals is termed the 'linearization problem' and is, of course, related to the problem of navigating a hypertext; in essence any reader navigating a hypertext is constructing a linearization on the fly. There are, however, navigational problems associated with complex hypertexts that many workers have reported (see, for example, Conklin, 1987; Simpson, 1989, 1990). Typically such problems arise from the intricacy of the linkage structure that may be present in a hyperdocument, and result in the inclusion of irrelevant nodes, or the omission of relevant nodes. In addition, what constitutes an acceptable linearization may depend heavily on the intended audience for the final linear text. Thus, some readers may want only a *précis* of the hypertext content, whereas others may wish to see an almost complete exposition of the textual material. In the following section, we will give a formal exposition of the representation of a hypertext as a graph, and of techniques for achieving particular linearizations of this hypertext.

Hypertext as graphs

Bench-Capon *et al.* (1992b, 1993) describe approaches to the development of hypertext linearization algorithms that are capable of dealing with problems arising from the need to linearize hypertext. The approach insists that a specification of the target document structure—the desired linearization—be given as input together with the hypertext to be linearized. The intention of this specification is to prescribe the relationships between textual nodes in the hypertext to be included in the actual linear version. The view that linearization should proceed with respect to a given target document structure thus divides this task into two separate activities: firstly finding an embedding of the hypertext information on to a document graph with a specified structure; and secondly, producing a linearization of the embedded form. If the target document graph is sufficiently richly structured, then the task of producing a linearization of this will be relatively straightforward, cf. the examples in

Bench-Capon and Dunne (1989). Requiring the target document structure to be a directed acyclic graph allows the richness of a hypertext linkage to be retained, while simplifying specific traversal problems to the problem of extracting valid paths from an acyclic scheme, and ensures that the resulting linearization constitutes a coherent and sensible document.

To these ends, Bench-Capon *et al.* (1992b, 1993) introduce a specification formalism (*extended regular expressions*) which may be used to define a (family of) directed *acyclic* graphs. From this, it is not difficult to exhibit a correspondence between paths from source to sink nodes in these directed graphs and the strings in the set generated by the corresponding extended regular expression. Finally, by viewing a hypertext linearization as a constrained total ordering of a subset of hypertext nodes, we can define a linearization as conforming to a specification if the sequence of node and edge labels used within it give rise to a string within the defined extended regular set of strings.

Definition 5. A hypertext, H , over the character set (or *alphabet*) Σ is defined by a quintuple:

$$H \equiv (V; E; \lambda_V; \lambda_E; \chi)$$

where $V = \{1, 2, \dots, n\}$ is a finite set of graph nodes; $E \subseteq V \times V$ is a finite set of graph edges; $\lambda_V: V \rightarrow \Sigma^*$ is a *node-labelling* function; $\lambda_E: E \rightarrow \Sigma^*$ is an *edge-labelling* function; and $\chi: V \rightarrow \Sigma^*$ is the mapping describing the textual content stored at each node.

For a mapping λ_R from some set R on to Σ^* , we define the set $Names(\lambda_R)$ by

$$Names(\lambda_R) =_{def} \{\alpha: \exists x \in R \text{ such that } \lambda_R(x) = \alpha\}.$$

Hence, $Names(\lambda_V)$ is the set of node labels used and $Names(\lambda_E)$ the set of edge labels used. We assume throughout that there is a null (empty) label in both sets.

Definition 6. An *extended regular expression* over an alphabet of terminal symbols, Σ and *connection set*, Λ , is any expression built in accordance with the following:

- (1) $\forall \sigma \in \Sigma$: σ is an extended regular expression.
- (2) If S and T are extended regular expressions, then so are:
 - 2.1. $S \oplus T$ (*alternative*)
 - 2.2. $S \rightarrow T$ (*connection*)
 - 2.3. $S \overset{\lambda}{\rightarrow} T, \forall \lambda \in \Lambda$ (*labelled connection*)
 - 2.4. (S) (*bracketing*)
 - 2.5. S^* (*repetition*).
- (3) All that are extended regular expressions arise by reason of (1) and (2) alone.

$ERE(\Sigma, \Lambda)$ denotes the set of all extended regular expressions with alphabet Σ and connection set Λ .

The extended regular expressions that may be produced using this definition can be mapped on to labelled tree structures; this affords a relatively low level pattern matching facility for subgraphs of a hypertext. This enables Definition 7 to be used in a mapping from $ERE(\Sigma, \Lambda)$ on to suitable subgraphs of $H \equiv (V; E; \lambda_V; \lambda_E; \chi)$.

Definition 7. A *target graph specification* is an extended regular expression with alphabet $Names(\lambda_V)$ and connection set $Names(\lambda)$, i.e. an element of $ERE(Names(\lambda_V); Names(\lambda_E))$.

A linearization of a hypertext, $H(V, E)$, may be viewed as constructing a total ordering (chain) on some subset of the hypertext nodes, the ordering being required to be consistent with the link structure of the hypertext, i.e. if a node v precedes a node w then there is a directed path from v to w in $H(V, E)$. In a directed acyclic graph there are a finite number of source and sink nodes. When traversing a path from some source to a sink, the sequence of node and edge labels occurring on this path can be regarded as a finite string over the alphabet $Names(\lambda_V) \cup Names(\lambda_E)$. This string always consists of alternating node and edge labels.

In terms of implementing the procedure the linearization process uses the target graph specification to extract a maximal portion of the hypertext defining a directed acyclic graph in conformance with the specified structure. This graph could then either be linearized automatically (just by selecting any source-to-sink path) or passed to the end-user who could select a desired linearization by traversing a source-sink path. We have then a mechanism by which users may predefine desired patterns of linearization structure and then either use those patterns to generate automatically reports or use them as an aid in refining material to be searched.

Experimental studies involving this formalism, however, reveal some weaknesses that would arise in practical situations. These may be summarized as follows.

- (1) Specific orderings of hypertext nodes can only be produced if the linkage structure of the network connects them. Thus, linear orderings that are implicit in the network linkage cannot be specified. For example, there will be a link from cases to the justice who presided, but the list of all cases in which a particular justice presided is only implicit in the network.
- (2) Hypertext systems may be constructed as a hierarchy of node classes. Target graph specifications operate only on the lowest node level and thus cannot combine sub-levels of different hierarchies together easily. For example, a case may have a link to a list of cases cited in the decision, but to extract a list of all cases in which a particular case had been cited would involve unpacking this structure.

Again, it is not possible to cater for certain natural linear orderings of the hypertext, since the hierarchical representation is not expanded by the regular expression formalism.

The two problems illustrated above mean that in order to extract particular orderings the hypertext representation itself would have to be amended by the inclusion of extra, rather unnatural, links if target graph specifications were to be used. *Linearization schema* were introduced by Bench-Capon *et al.* (1993) as a means of overcoming these problems. The central element is the concept of *constructor definitions*. Constructors define sub-networks in terms of hypertext node and edge labels but provide the facility for these to be combined in a manner that need not mirror the exact linkage structure of the hypertext.

Definition 8. A constructor on $H(V, E, \lambda_V, \lambda_E, \chi)$ is a specification of the form

```

constructor < constructor-name > is
  signature < Source-name, Sink-name > by { < edge-label > }
  operator < Source-name > *  $\overset{\lambda}{\downarrow}$  < Sink -name > |
  operator < Source-name > )  $\overset{\lambda}{\downarrow}$  < Sink -name > *
end

```

where *Source-name* and *Sink-name* are elements of $Names(\lambda_V)$ or previously defined constructor-names; *edge-label* is a member of $Names(\lambda_E)$.

A *linearization schema* for H is a sequence of constructors

$$C = \langle C_1, \dots, C_m \rangle$$

together with an expression

$$A \in ERE (C \cup Names(\lambda_V); names(\lambda_E)).$$

Constructor definitions, when executed, return a set of directed acyclic graphs 'implicitly consistent' with the structure of the hypertext being linearized. The above definition gives a syntactic form for schemata. The mapping from these on to directed acyclic graphs is rather more complicated and is described in Bench-Capon *et al.* (1993).

The mapping from constructors to graphs in the alternative operator definition is performed in a similar manner.

Presenting a statute in a WWW browser

In this section, we will illustrate the previous material by considering some different ways of presenting a document on a computer terminal. Suppose we have a statute represented in the above manner, its grammar given by the specification in the Appendix. It would be wrong to assume that the best presentation on a terminal would necessarily be close to the presentation as a paper document: the act of reading from a physical document is very different from that of reading an electronic document, and the layout conventions that support the ready perusal of a physical document may well not be appropriate in an electronic document.

Finding a section

As an example consider *section titles*. These are set out in the left margin, in very small font, possibly running over several lines, alongside their corresponding section. The font is small, both to distinguish them from the text of the sections and to avoid them drawing too much attention to themselves once their purpose of leading the reader to the desired section has been served. Scanning down the margin reading these titles is an effective way of locating the desired section relatively rapidly. This function would not, however, be as well served if reproduced in the electronic display.

- Small text spread across several narrow lines is very difficult to read.
- Paging an electronic document is less time-efficient than rifling through a physical document.

- The conventions associated with other electronic documents strongly prejudice us in favour of the 'click-unfold' method of obtaining information.

All this suggests that the main mode of access to sections should be through a table of contents composed from the section titles which can be expanded by clicking on the desired item to get the expanded text. Thus, a useful linearization would be one which constructed a table of contents. One possible structure for such a target document would be

```
<Contents> := <Short Title> <Contents_line>*
<Contents_line> := <Section_number> <Section_title>
```

For an act of reasonable size, however, this would give rise to a large number of entries, and the reader may well prefer to see something at a higher level of abstraction. The target structure for the most abstract level would be:

```
<Contents_2> := <Short Title> <Contents_2_line>*
<Contents_2_line> := <Part_Line>* <Schedule_line>*
<Part_Line> := <Part number> <Part_title>
<Schedule_line> := <Schedule_number> <Schedule_Title>
```

A third, intermediate level of abstraction could be obtained by using the group titles

```
<Contents_3> := <Short Title> <Part_Content_Line>*
<Schedule_line> :=
<Part_Content_Line> := <Long_Part_Lines> | <Short_Part_line>
<Long_Part_Lines> := <Part_number> <Part_title> <Group_Line>*
<Group_Line> := <Group_Title>
<Short_Part_Line> := <Part_number> <Part_title>
```

Which of these would best suit a reader would depend on the reader's preferences, the reader's knowledge of the act, the size of the act, and so on. It is, however, possible to leave that choice to the reader, since each of the linearizations can be achieved by posing the request to extract the required target document from the source graph, producing a customized table of contents. For example to extract using the second form, we would use

```
constructor single_part_line is
  signature part_number, part title by null
  operator part_number → part_title
end constructor
```

```
constructor part_line_list is
  signature single_part_line by null
  operator (part_line_list)*
end constructor
```

```
constructor single_schedule_line is
  signature schedule_number, schedule_title by null
  operator schedule_number → schedule_title
end constructor
```

```
constructor schedule_line_list is
  signature single_schedule_line by null
```

```

operator (schedule_line_list)*
end constructor

```

```

Short_title) → (part_line_list → schedule_line_list)*

```

The final line gives the target graph structure in terms of subgraph structures specified by the four constructor definitions.

Schedules

Schedules relate to one or more sections of the act. These are indicated by one or more marginal cross references. This is the best that can be done with a physical document, but in an electronic version we would not wish to be continually switching between the display of the section(s) and the display of the schedule. Thus, a better presentation of the schedule would display the text both of the sections and of the schedule as a single virtual document, rather than simply giving the sections as anchors.

Thus, a possible linearization for a schedule would be

```

< Schedule_display > := < Schedule_number > < Schedule_title >
                        < sections > < Schedule_parts >
< sections > := < section > < section > *

```

where the terminals are as defined in the Appendix to this paper, and the sections in < sections > are those denoted by the cross-references in the schedule. This could be extracted by

```

constructor sections is
signature schedule_title, section_body by section_cross_ref
operator schedule_title section_cross_ref → section_body*
end constructor

```

```

schedule_number → sections → Schedule_part → (Schedule_part)*

```

The constructor *sections* is used to collect all parts of a section indicated by a cross-reference within a named schedule.

The above two examples show cases where it is clear that the target document for electronic display should be in a form different from that of the physical document. It is, however, likely that there will be many other examples where particular linearizations will be required by particular readers with particular tasks to perform on particular hardware platforms. It is unnecessary to anticipate all of these in advance, since where the required form can be described as a target document graph, the query to extract it can be constructed. It is the very flexibility of our scheme which gives due weight to the structure of the source document that gives it advantages over systems which would hardwire the display into the mark-up language.

Preparing a brief

In this section, we will describe another application of the techniques, the preparation of a brief. When a brief is prepared information from a variety of sources is organized into a coherent argument for a particular point of view. A

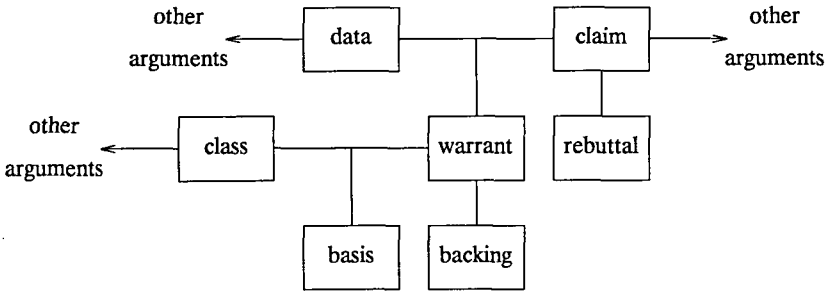


Figure 3. Modified argument schema.

number of suggestions have been proposed for the representation of an argument in schematic form, such as those for Toulmin (1958) and Hosking (1994). We will confine ourselves here to Toulmin’s schema, as extended in Bench-Capon *et al.* (1991), which is shown in Fig. 3.

It is obvious that Toulmin’s representation of an argument forms a particular kind of graph, which a set of distinct node types connect in a particular way. When supporting the preparation of a brief, we use this formalism to mediate between an expert system and the hypertext representing the various legal sources. This is achieved as follows.

Using the techniques described by Bench-Capon *et al.* (1991), the expert system is executed so as to produce its output in the form of a graph representing an argument represented using the Toulmin structure. The rules of the expert system used to justify the answer form the warrants of the graph. The rules themselves are justified by reference to items found in the legal sources, cases, sections of legislation, etc. The backing nodes state the sources which justify the particular rule. The backing nodes therefore give us an entry point into the hypertext of legal sources: the argument graph can be integrated with the hypertext by making the backing nodes of the argument graph the corresponding nodes of the hypertext.

The brief can now be constructed by traversing the argument graph, which will provide structured access to the relevant sources to support the argument derived from the expert system. Typically, the argument graph will contain more detail than is required: in traversing the graph the user will select those nodes required for an effective argument. Finally, using a specification of the form required for a brief, the selected nodes can be organized into the desired format.

This system has been prototyped in a system PLAID: for a fuller description see Bench-Capon and Staniford (1995).

Conclusions

In this paper, we have described some techniques for the management of legal documents using a graph theoretic representation. Key elements of this approach include that we can provide a formal basis for the description of the structures of various typical classes of document and that we can use these

structural descriptions to relate documents within a document base or a hypertext.

References

- Bench-Capon, T.J.M. & Dunne, P.E.S. (1989) Some computational properties of a model for electronic documents, *Electronic Publishing*, 2, pp. 231–256.
- Bench-Capon, T.J.M., Lowes, D. & McEnery, A.M. (1991) Using Toulmin's argument schema to explain logic programs, *Knowledge Based Systems*, 4, pp. 177–183.
- Bench-Capon, T.J.M., Dunne, P.E. & Leng, P.H. (1992a) A dialogue game for dialectical interaction with expert systems, *12th Annual Conference on Expert Systems and Their Applications*, Avignon.
- Bench-Capon, T.J.M., Dunne, P.E.S. & Staniford, G. (1992b) Linearising hypertext through target graph specifications, *Proc. Dexa92*, pp. 173–178 (Valencia, Springer).
- Bench-Capon, T.J.M., Dunne, P.E.S. & Staniford, B. (1993) Linearisation Schematic for Hypertext, *Proc. Dexa93*, pp. 697–708 (Springer, LOVCS, vol. 720).
- Bench-Capon, T.J.M. & Staniford, G. (1995) PLAID—proactive legal assistance, *5th International Conference on AI and Law*, University of Maryland, pp. 81–88 (New York, ACM Press).
- Bertino, E. Rabitti, F. & Gibbs, S. (1988) Query processing in a multimedia document system, *ACM Transactions on Office Information Systems*, 6, 1988, pp. 1–41.
- Brown, P.J. (1988) Hypertext: the way forward, in J.C. van Vliet (Ed.), *Document Manipulation and Typography*, pp. 183–191 (Cambridge, Cambridge University Press).
- Christodoulakis, S. Theodoridou, M., Ho, F., Papa, M. & Pathria, A. (1986) Multimedia document presentation, information extraction, and document formation in minos: a model and a system, *ACM Transactions on Office Information Systems*, 4(4), pp. 345–383.
- Conklin, J. (1987) Hypertext: an introduction and survey, *Computer*, 20, pp. 17–41.
- Delisle, N. & Schwartz, M. (1986) Neptune: a hypertext system for CAD applications, *Proc. ACM International Conference on Management of Data*, pp. 132–143.
- Groarke, L. (1982) When Two Wrongs Make a Right, *Informal Logic Newsletter*, 5, pp. 10–13.
- Hosking, P., (1994) Argument Representation and Conceptual Retrieval for Litigation Support, Technical Report CS-TR-94/19, Department of Computer Science, Victoria University of Welling.
- Kimura, G.D. & Shaw, A.C. (1984) The structure of abstract document objects, *Proc. ACM Conference on Office Information Systems*, pp. 161–169.
- Koo, R. (1989) A model for electronic documents, *ACM SIGOIS Bulletin*, 10, pp. 23–33.
- Meyrowitz, N. (1986) Intermedia: the architecture and construction of an object-oriented hypermedia system and applications framework, *Proc. Object-oriented Program Systems, Languages and Applications*, pp. 186–201.
- Simpson, A. (1989) Navigation in hypertext: design issues, *International Online 89 Conference*, London, December.
- Simpson, A. & McKnight, C. (1990) Navigation in hypertext: structural cues and mental maps, in R. McAleese & C. Green (Eds), *Hypertext: State of the Art* (Oxford, Intellect).
- Staniford, G. (1994) Multi-agent systems in support of computer supported cooperative authorship, PhD Thesis, Department of Computer Science, University of Liverpool.
- Stotts, P.D. & Furuta, R. (1988) Adding browsing semantics to the hypertext model, *Proc. ACM Conference on Document Processing Systems*, pp. 43–50.
- Thomas, R.H., Forsdick, H.C., Crowley, T.R., Robertson, G.G., Schaaf, R.W., Tomlinson, R.S. & Travers, V.M. (1985) Diamond: a multimedia message system built upon a distributed architecture, *Computer*, 18, pp. 65–78.
- Toulmin, S. (1958) *The Uses of Argument* (Cambridge, Cambridge University Press).
- Walton, D.N. (1985) Arguers position: a pragmatic study of ad hominem attack, criticism, refutation, and fallacy, *Contributions in Philosophy* (London, Greenwood Press).

Appendix: A grammar for legislation

In this section we present the specification of a UK Statute below using a stylized BNF notation.

< Act >

:=

< Short_Title > < Date > < Chapter > < Long_Title > < Preamble >

```

<Parts> <Schedule>*
<Parts>:= <Part> <Part>*
<Part>:= <Part_number> <Part_title> <Long_Part>|<Short_Part>
<Long_Part>:= <Group> <Group>*
<Short_Part>:= <Section> <Section>*
<Group>:= <Group_title> <Short Part>
<Section>:= <Section_number> <Section_Title> <Section_Body>
<Section_Body>:= <Section_Text>|<Sub_section> <Sub_section>*
<Sub_section>:= <Sub_section_number> <Sub_section_body>
<Sub_section_body>:= <Sub_section_text>{<sub_sub_section>}
<Sub_sub_section>:= <sub_sub_section_letter> <sub_sub_section_body>
<Sub_sub_section_body>:= <Sub_sub_section_text>|
                                <sub_sub_section header> <sub_sub_sub_section>
<sub_sub_sub_section>:= <roman_numeral>
<sub_sub_sub_section_text>

<Schedule>
:=
<Schedule_number> <schedule_title> <section_cross_refs>
<Schedule_parts>
<section_cross_refs>:= <cross_reference> <cross_reference>*
<Schedule_parts>:= <schedule_part> <schedule_part>*.

```

Terms in **bold** font indicate that these structures are not sub-divided further, i.e. are purely textual; such terms are the base node labels occurring in the document graph. Terms in *italic* font indicate link labels between given node types. A term of the form <xxx> denotes a structure that may be expanded into lower level structures or terminal node labels. For example, <Block> := <Title> <Body> indicates that a text node with node label <Title> linked to a text node with node label <Body> defines a structure called a <Block>.