Decoding Algorithms for Complex Natural Language Tasks

by

Pawan Deshpande

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 2007
[February 2007]

© Massachusetts Institute of Technology 2007. All rights reserved.

Author .... [illegible]
Department of Electrical Engineering and Computer Science
January 30, 2007

Certified by .... [illegible]
Regina Barzilay
Assistant Professor
Thesis Supervisor

Accepted by .... [illegible]
Arthur C. Smith
Chairman, Department Committee on Graduate Students

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
OCT 0 3 2007
LIBRARIES

ARCHIVES

Decoding Algorithms for Complex Natural Language Tasks

by

Pawan Deshpande

Submitted to the Department of Electrical Engineering and Computer Science
on January 30, 2007, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Engineering

Abstract

This thesis focuses on developing decoding techniques for complex Natural Language Processing (NLP) tasks. The goal of decoding is to find an optimal or near optimal solution given a model that defines the goodness of a candidate. The task is challenging because in a typical problem the search space is large, and the dependencies between elements of the solution are complex.

The goal of this work is two-fold. First, we are interested in developing decoding techniques with strong theoretical guarantees. We develop a decoding model based on the Integer Linear Programming paradigm which is guaranteed to compute the optimal solution and is capable of accounting for a wide range of global constraints. As an alternative, we also present a novel randomized algorithm which can guarantee an arbitrarily high probability of finding the optimal solution. We apply these methods to the task of constructing temporal graphs and to the task of title generation. Second, we are interested in carefully investigating the relations between learning and decoding. We build on the Perceptron framework to integrate the learning and decoding procedures into a single unified process. We use the resulting model to automatically generate tables-of-contents, structures with deep hierarchies and rich contextual dependencies. In all three natural language tasks, our experimental results demonstrate that theoretically grounded and stronger decoding strategies perform better than existing methods. As a final contribution, we have made the source code for these algorithms publicly available for the NLP research community.

Thesis Supervisor: Regina Barzilay
Title: Assistant Professor

Acknowledgments

First and foremost, I would like to thank my advisor Regina Barzilay. In one short year, I was able to accomplish much more than I ever had imagined. It was only through her ever-ready willingness to help and indispensable guidance that this was possible.

Aside from my advisor, I must also thank my collaborators who worked with me on various papers along the way that led to this thesis: S.R.K. Branavan, Philip Bramsen, Yoong Keok Lee and David Karger. I will not forget the many long nights spent in the lab with Branavan, Philip and Yoong Keok.

I also would like to express my gratitude to others who helped in many other ways from proof-reading to providing much-needed feedback. In this regard, I would like to thank Eli Barzilay, Erdong Chen, Harr Chen, Michael Collins, Marcia Davidson, my brother Ashwin Deshpande, Zoran Dzunic, Jacob Eisenstein, Bo-June (Paul) Hsu, Terry Koo, Igor Malioutov, Christina Sauper, Yuan Shen, Ben Snyder, Peter Szolovits and Luke Zettlemoyer.

Finally, I would like to thank my parents and my roommate, Yang Zhang, for the copious sustenance.

Bibliographic Notes

Portions of this thesis are based on the following papers:

"Finding Temporal Order in Discharge Summaries" by Philip Bramsen, Pawan Deshpande, Yoong Keok Lee and Regina Barzilay. The paper appeared in the *Proceedings of the 2006 American Medical Informatics Association (AMIA) Annual Symposium.*

"Inducing Temporal Graphs" by Philip Bramsen, Pawan Deshpande, Yoong Keok Lee and Regina Barzilay. The paper appeared in the *Proceedings of Empirical Methods in Natural Language Processing (EMNLP) 2006.*

"Randomized Decoding for Selection-and-Ordering Problems" by Pawan Deshpande, Regina Barzilay and David R. Karger. The paper will appear in the *Proceedings of the 2007 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL).*

"Generating a Table-of-Contents: A Hierarchical Discriminative Approach" by S.R.K. Branavan, Pawan Deshpande and Regina Barzilay. The paper was submitted to the *45th Annual Meeting of the Association for Computational Linguistics (ACL).*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Decoding in NLP

In Natural Language Processing (NLP), researchers study systems that take natural language as input and/or produce natural language as an output. In order for these systems to perform effectively, we need to exploit both syntactic and semantic knowledge. In the past, NLP systems relied on human experts to manually encode this information. In recent years, increased computing power, advances in machine learning, and the availability of large corpora have allowed the research community to adopt a statistical approach for many language processing tasks. Since then, the success of statistical methods have made them the predominant technique in nearly all areas of NLP from syntactic parsing to natural language generation.

A typical model consists of a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that maps a set of possible inputs $\mathcal{X}$ to a set of possible outputs $\mathcal{Y}$. With supervised models, we seek to learn f from a set of annotated pairs $(x_1, y_1) \ldots (x_n, y_n)$ where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. We then use the learned model to predict an output for a previously unseen input. For example, in a document classification system, we are provided with a set of document-class pairs to learn a model. The model then predicts the class y of an input document x.

In many NLP tasks, the set of outputs $\mathcal{Y}$ consists of a large, possibly infinite, number of highly structured objects such as parse trees or document summaries. In these cases, the mapping function f is not learned directly. Instead, we learn a function $g : \mathcal{X} \times \mathcal{Y} \rightarrow [0,1]$ that returns a score or probability of an input-output hypothesis pair. We then use g to find the best output $\hat{y}$ in $\mathcal{Y}$ for an input x:

$$\hat{y} = \arg\max_{y \in \mathcal{Y}} g(x, y)$$

To date, a major focus of research has been on learning the function g in an effective and robust manner. However, little progress has been made on improving *decoding*, the process of finding the best output $\hat{y}$ given the learned scoring function.

The complexity of decoding depends on the size of the output space. If the output space is small, then decoding is simple. An important class of problems with this property are classification problems. For instance, in the task of word sense disambiguation, a word and its context are provided and we need to determine the sense in which the word is used. For example, if we are given the word *"bass"* in the sentence *"I caught a bass in the river."*, we must determine if the word refers to a type of fish or a low frequency tone. In this case, because the size of the output space is two, we only need to compare the probabilities of each word sense and return the one with the higher probability.

Unfortunately, many NLP tasks cannot be cast as classification problems. When the output structure is a sequence, as in part-of-speech tagging, or a syntax tree, as in parsing, a simplistic decoding technique may not suffice. In these cases, we may want to account for complex inter-dependencies between elements of the output solution. For instance, in part-of-speech tagging, we may want to select the part-of-speech tag of a word, based on the parts-of-speech of the words surrounding it. In the sentence *"He can see."*, a model that captures these dependencies would correctly determine *"can"* to be a modal verb rather than a noun because it is preceded by the pronoun, *"He"*, and followed by the verb, *"see"*. However, capturing these dependencies can greatly increase the size of the output space. If we consider the parts-of-speech of all other words when selecting the part-of-speech for each word, we would need to consider n^m possible tag sequences for a sentence of n words with m possible parts-of-speech.

Of course, we could simplify the decoding process by assuming that each element of the output solution is selected in isolation. Under this assumption, the part-of-speech tagger would ignore the dependencies between words altogether. Though such a decomposition would make decoding constant in the size of the input, it could have potentially drastic effects on the performance of the system. For example, the tagging model would no longer leverage the presence of the parts-of-speech of adjacent words and may now incorrectly determine *"can"* to be a noun.

In summary, the art of decoding lies in balancing the accuracy and the complexity of the dependencies considered. In the next section, we give examples of common decoding strategies to problems where the the output space may be exponential in the size of the input.

1.2 Common Decoding Strategies

How is decoding performed when the output space is exponential in the size of the input space? For many tasks, where we can assume local dependencies between elements of the output, we can use efficient solutions, such as dynamic programming, to find the exact solution. For example, in probabilistic context-free grammar (PCFG) parsing, we can model the local dependency between a child node and its parent, while disregarding global dependencies across the rest of the parse tree. Under this independence assumption, we can use the Cocke-Younger-Kasami (CYK) dynamic programming algorithm to efficiently find the best parse tree for a given input sentence [14].

For other tasks, where global dependencies cannot be decomposed into a set of local dependencies, it is not possible to find the exact solution in an efficient manner. In such cases, a common decoding strategy is to return an approximate solution, sacrificing accuracy for efficiency. A typical solution in this case is to employ a heuristic search method, such as a beam or greedy search, that considers only a subset of the total output space [45]. These methods incrementally build the output solution while maintaining a list of the top partial solutions during the process. For instance, beam search is the primary means of decoding in machine translation [34]. Because these methods prune out large portions of the output space, it is possible that the optimal solution may never be considered.

Another approximate decoding approach is to use sampling-based techniques such as simulated annealing or the Metropolis-Hastings algorithm. Rather than building a solution incrementally, these sampling techniques are initialized with a complete solution and iteratively refine the solution until there can be no further improvement or the desired number of iterations has been exceeded. These techniques are guaranteed to return a locally optimal solution, but not the globally optimal one. In many applications, returning a locally optimal solution is sufficient. In recent years, these methods have been successfully applied to a handful of tasks such as named entity recognition [27] and machine translation [24]. While these approximation methods may perform well in practice, there are no theoretical guarantees on their likelihood of finding the exact solution.

Furthermore, when an inexact inference method is employed, most learning algorithms make no assumptions about the nature of the decoder – the model is learned independent of the decoder. However, during the search for a solution, the actual decoder is in fact approximate and inexact. This failure to account for the actual decoder during training results in suboptimal performance of the entire system.

1.3 Scope of the Thesis and Contributions

In this thesis we explore decoding methods in the context of two NLP areas: natural language generation and discourse processing. Natural language generation deals with translating a semantic representation to a natural language text. For example, we may want to automatically convert a database of financial records into a financial report. Discourse processing involves modelling relations between sentences in a text, covering aspects such as topical, temporal and rhetorical progression. For example, a discourse processing system could be developed to understand the chronology of events described in a text. Both areas need to capture complex dependencies and are still challenging for statistical techniques.

This thesis is a first attempt at developing decoding techniques for these two areas of NLP. The goal of this work is two-fold. First, we are interested in developing decoding techniques with strong theoretical guarantees. In some cases, we can employ Integer Linear Programming (ILP) which is guaranteed to compute the optimal solution and is capable of accounting for a wide range of global constraints [28, 44]. Although ILP is NP-hard, some ILP formulations can be solved in a reasonable amount of time since many powerful heuristics have been developed for this optimization technique. In other cases, where a problem can not be solved quickly with ILP, we can use randomized algorithms that guarantee an arbitrarily high probability of finding the optimal solution. The second objective of this thesis is to carefully investigate the relations between learning and decoding. Ideally, the learning algorithm is aware of the shortcuts used by decoder. However, in traditional approaches, the learning model does not have knowledge of the decoding procedure, divorcing the actual decoder from the learning process. We might therefore improve performance by integrating the learning and decoding processes into a single unified process.

In **chapter 2**, we investigate decoding in the ILP framework. We consider the problem of constructing a directed acyclic graph that encodes temporal relations found in a text. The unit of our analysis is a temporal segment, a fragment of text that maintains temporal coherence. The strength of our approach lies in its ability to simultaneously optimize pairwise ordering preferences between temporal segments as well as global constraints on the graph topology. We show that the ILP model outperforms two greedy baseline methods.

Chapter 2 is organized as follows. First, we introduce the problem and highlight the importance of the application. Next, we discuss previous approaches for temporal ordering. We formalize our representation of temporal flow and discuss our method for temporal segmentation. We then describe two greedy decoding methods and one ILP method for graph induction. After that, we discuss our corpus and evaluation methods. To conclude the chapter, we present our results and discuss the performance of our system.

In **chapter 3**, we propose the use of a randomized decoding algorithm for selection-and-ordering problems. The task of selecting and ordering information appears in multiple contexts in text generation and summarization. For instance, methods for title generation construct a headline by selecting and ordering words from the input text. In this chapter, we investigate decoding methods that simultaneously optimize selection and ordering preferences. We formalize decoding as a task of finding an acyclic path in a directed weighted graph. Since the problem is NP-hard, finding an exact solution is challenging. We describe a novel decoding method based on a randomized color-coding algorithm [2]. We prove bounds on the number of color-coding iterations necessary to guarantee any desired likelihood of finding the correct solution. Our experiments show that the randomized decoder is an appealing alternative to a range of decoding algorithms for selection-and-ordering problems, including beam search and ILP.

The organization of chapter 3 is as follows. We first discuss the importance of decoding in selection-and-ordering problems. Then, we provide an overview of existing work on decoding in text generation and other relevant tasks. Following that, we formalize decoding in a graph-theoretic framework and introduce our representation. Next, we discuss the connections between our decoding problem and classical graph-traversal algorithms. The randomized algorithm and an ILP formulation are presented next. Finally, we describe our evaluation setup and data. We conclude the chapter by presenting and discussing our results.

In **chapter 4**, we explore how incorporating a heuristic decoder into the learning process can affect system performance for the task of table-of-contents generation. We implement a model that accounts for decoding during the learning process and apply it to automatically generate a table-of-contents. To generate a coherent table-of-contents, we need to capture both global dependencies across different titles in the table and local constraints within sections. Our algorithm effectively handles these complex dependencies by factoring the model into local and global components, and incrementally constructing the model's output. The results of automatic evaluation and manual assessment confirm the benefits of this design: our system is consistently ranked higher than non-hierarchical baselines.

At the start of chapter 4, we explain the benefits of an automated means of generating a table-of-contents and provide an overview of the challenges involved. We then provide an overview of relevant work on text summarization. Then, we formalize the problem of table-of-contents generation and introduce our approach for incremental learning and decoding. Next, we describe our experimental framework and data. At the end the chapter, we present our results and provide directions for future work.

In **chapter 5**, we conclude the thesis by discussing the key points and discuss avenues for future research.

As a final contribution, we have made the source code for the decoding algorithms discussed in the thesis publicly available for the NLP research community.[1]

[1]The source code for the methods described in chapter 3 is available at `http://people.csail.mit.edu/pawand/randomized/`. The code for the model described in chapter 4 is accessible at `http://people.csail.mit.edu/pawand/toc/`.

Chapter 2

Inducing Temporal Graphs

2.1 Introduction

Understanding the temporal flow of discourse is a significant aspect of text comprehension. Consequently, temporal analysis has been a focus of linguistic research for quite some time. Temporal interpretation encompasses levels ranging from the syntactic to the lexico-semantic [43, 39] and includes the characterization of temporal discourse in terms of rhetorical structure and pragmatic relations [23, 51, 41, 36].

Besides its linguistic significance, temporal analysis has important practical implications. In multidocument summarization, knowledge about the temporal order of events can enhance both the content selection and the summary generation processes [7]. In question answering, temporal analysis is needed to determine when a particular event occurs and how events relate to each other. Some of these needs can be addressed by emerging technologies for temporal analysis [53, 38, 35, 9].

We characterize the temporal flow of discourse in terms of *temporal segments* and their ordering. We define a temporal segment to be a fragment of text that does not exhibit abrupt changes in temporal focus [52]. A segment may contain more than one event or state, but the key requirement is that its elements maintain temporal coherence. For instance, a medical case summary may contain segments describing a patient's admission, his previous hospital visit, and the onset of his original symptoms. Each of these segments corresponds to a different time frame, and is clearly delineated as such in a text.

Our ultimate goal is to automatically construct a graph that encodes ordering between temporal segments. The key premise is that in a coherent document, temporal progression is reflected in a wide range of linguistic features and contextual dependencies. In some cases, clues to segment ordering are embedded in the segments themselves. For instance, given a pair of adjacent segments, the temporal adverb *next day* in the second segment is a strong predictor of a precedence relation. In other cases, we can predict the right order between a pair of segments by analyzing their relation to other segments in the text. The interaction between pairwise ordering decisions can easily be formalized in terms of constraints on the graph topology. An obvious example of such a constraint is prohibiting cycles in the ordering graph. We show how these complementary sources of information can be incorporated in a model using global inference.

We evaluate our temporal ordering algorithm on a corpus of medical case summaries. Temporal analysis in this domain is challenging in several respects: a typical summary exhibits no significant tense or aspect variations and contains few absolute time markers. We demonstrate that humans can reliably mark temporal segments and determine segment ordering in this domain. Our learning method achieves 83% F-measure in temporal segmentation and 84% accuracy in inferring temporal relations between two segments.

Our contributions are twofold:

Temporal Segmentation We propose a fully automatic, linguistically rich model for temporal segmentation. Most work on temporal analysis is done on a finer granularity than proposed here. Our results show that the coarse granularity of our representation facilitates temporal analysis and is especially suitable for domains with sparse temporal anchors.

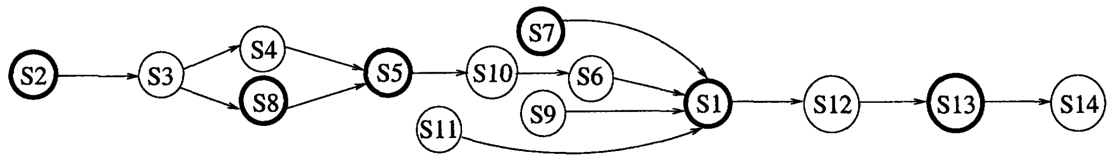
Segment Ordering We introduce a new method for learning temporal ordering. In contrast to existing methods that focus on pairwise ordering, we explore strategies for global temporal inference. The strength of the proposed model lies in its ability to simultaneously optimize pairwise ordering preferences and global constraints on graph topology. While the algorithm has been applied at the segment level, it can be used with other temporal annotation schemes.

2.2 Related Work

Temporal ordering has been extensively studied in computational linguistics [41, 52, 31, 36, 37]. Prior research has investigated a variety of language mechanisms and knowledge sources that guide interpretation of temporal ordering, including tense, aspect, temporal adverbials, rhetorical relations and pragmatic constraints. In recent years, the availability of annotated corpora, such as TimeBank [42], has triggered the use of machine-learning methods for temporal analysis [38, 35, 9]. Typical tasks include identification of temporal anchors, linking events to times, and temporal ordering of events.

Since this chapter addresses temporal ordering, we focus our discussion on this task. Existing ordering approaches vary both in terms of the ordering unit — it can be a clause, a sentence or an event — and in terms of the set of ordering relations considered by the algorithm. Despite these differences, most existing methods have the same basic design: each pair of ordering units (i.e., clauses) is abstracted into a feature vector and a supervised classifier is employed to learn the mapping between feature vectors and their labels. Features used in classification include aspect, modality, event class, and lexical representation. It is important to note that the classification for each pair is performed independently and is not guaranteed to yield a globally consistent order.

In contrast, our focus is on globally optimal temporal inference. While the importance of global constraints has been previously validated in symbolic systems for temporal analysis [26, 55], existing corpus-based approaches operate at the local level. These improvements achieved by a global model motivate its use as an alternative to existing pairwise methods.



S1	A 32-year-old woman was admitted to the hospital because of left subcostal pain...
S2	The patient had been well until four years earlier,
S5	Three months before admission an evaluation elsewhere included an ultrasonographic examination, a computed tomographic (CT) scan of the abdomen...
S7	She had a history of eczema and of asthma...
S8	She had lost 18 kg in weight during the preceding 18 months.
S13	On examination the patient was slim and appeared well. An abdominal examination revealed a soft systolic bruit... and a neurologic examination was normal...

Figure 2-1: An example of the transitive reduction of a TDAG for a case summary. A sample of segments corresponding to the nodes marked in bold is shown in the table.

2.3 TDAG: A representation of temporal flow

We view text as a linear sequence of temporal segments. Temporal focus is retained within a segment, but radically changes between segments. The length of a segment can range from a single clause to a sequence of adjacent sentences. Figure 2-1 shows a sample of temporal segments from a medical case summary. Consider as an example the segment S13 of this text. This segment describes an examination of a patient, encompassing several events and states (i.e., an abdominal and neurological examination). All of them belong to the same time frame, and temporal order between these events is not explicitly outlined in the text.

We represent the ordering of events as a temporal directed acyclic graph (TDAG). An example of the transitive reduction[1] of a TDAG is shown in Figure 2-1. Edges in a TDAG capture temporal precedence relations between segments. Because the graph encodes an order, cycles are prohibited. We do not require the graph to be fully connected — if the precedence relation between two nodes is not specified in the text, the corresponding nodes will not be connected. For instance, consider the segments

[1]The transitive reduction of a graph is the smallest graph with the same transitive closure.

S5 and S7 from Figure 2-1, which describe the patient's previous tests and the history of eczema. Any order between the two events is consistent with our interpretation of the text, therefore we cannot determine the precedence relation between the segments S5 and S7.

In contrast to many existing temporal representations [1, 42], TDAG is a coarse annotation scheme: it does not capture interval overlap and distinguishes only a subset of commonly used ordering relations. Our choice of this representation, however, is not arbitrary. The selected relations are shown to be useful in text processing applications [55] and can be reliably recognized by humans. Moreover, the distribution of event ordering links under a more refined annotation scheme, such as TimeML, shows that our subset of relations covers a majority of annotated links [42].

2.4 Method for Temporal Segmentation

Our first goal is to automatically predict shifts in temporal focus that are indicative of segment boundaries. Linguistic studies show that speakers and writers employ a wide range of language devices to signal change in temporal discourse [8]. For instance, the presence of the temporal anchor *last year* indicates the lack of temporal continuity between the current and the previous sentence. However, many of these predictors are heavily context-dependent and, thus, cannot be considered independently. Instead of manually crafting complex rules controlling feature interaction, we opt to learn them from data.

We model temporal segmentation as a binary classification task. Given a set of candidate boundaries (e.g., sentence boundaries), our task is to select a subset of the boundaries that delineate temporal segment transitions. To implement this approach, we first identify a set of potential boundaries. Our analysis of the manually-annotated corpus reveals that boundaries can occur not only between sentences, but also within a sentence, at the boundary of syntactic clauses. We automatically segment sentences into clauses using a robust statistical parser [12]. Next, we encode each boundary as a vector of features. Given a set of annotated examples, we train a classifier[2] to predict boundaries based on the following feature set:

Lexical Features Temporal expressions, such as *tomorrow* and *earlier*, are among the strongest markers of temporal discontinuity [41, 8]. In addition to a well-studied set of domain-independent temporal markers, there are a variety of domain-specific temporal markers. For instance, the phrase *initial hospital visit* functions as a time anchor in the medical domain.

To automatically extract these expressions, we provide a classifier with n-grams from each of the candidate sentences preceding and following the candidate segment boundary.

Topical Continuity Temporal segmentation is closely related to topical segmentation [11]. Transitions from one topic to another may indicate changes in temporal flow and, therefore, identifying such transitions is relevant for temporal segmentation.

We quantify the strength of a topic change by computing a cosine similarity between sentences bordering the proposed segmentation. This measure is commonly used in topic segmentation [29] under the assumption that change in lexical distribution corresponds to topical change.

Positional Features Some parts of the document are more likely to exhibit temporal change than others. This property is related to patterns in discourse organization of a document as a whole. For instance, a medical case summary first discusses various developments in the medical history of a patient and then focuses on his current conditions. As a result, the first part of the summary contains many short temporal segments. We encode positional features by recording the relative position of a sentence in a document.

Syntactic Features Because our segment boundaries are considered at the clausal level, rather than at the sentence level, the syntax surrounding a hypothesized boundary may be indicative of temporal shifts. This feature takes into account the position of a word with respect to the boundary. For each word within three words of the hypothesized boundary, we record its part-of-speech tag along with its distance from the

[2]BoosTexter package [46].

boundary. For example, NNP_{+1} encodes the presence of a proper noun immediately following the proposed boundary.

2.5 Learning to Order Segments

Our next goal is to automatically construct a graph that encodes ordering relations between temporal segments. One possible approach is to cast graph construction as a standard binary classification task: predict an ordering for each pair of distinct segments based on their attributes alone. If a pair contains a temporal marker, like *later*, then accurate prediction is feasible. In fact, this method is commonly used in event ordering [38, 35, 9]. However, many segment pairs lack temporal markers and other explicit cues for ordering. Determining their relation out of context can be difficult, even for humans. Moreover, by treating each segment pair in isolation, we cannot guarantee that all the pairwise assignments are consistent with each other and yield a valid TDAG.

Rather than ordering each pair separately, our ordering model relies on global inference. Given the pairwise ordering predictions of a local classifier[3], our model finds a globally optimal assignment. In essence, the algorithm constructs a graph that is maximally consistent with the individual ordering preferences of each segment pair and at the same time satisfies graph-level constraints on the TDAG topology.

In Section 2.5.2, we present three global inference strategies that vary in their computational and linguistic complexity. But first we present our underlying local ordering model.

2.5.1 Learning Pairwise Ordering

Given a pair of segments (i, j), our goal is to assign it to one of three classes: forward, backward, and null (not connected). We generate the training data by using all pairs of segments (i, j) that belong to the same document, such that i appears before j in the text.

[3]The perceptron classifier.

The features we consider for the pairwise ordering task are similar to ones used in previous research on event ordering [38, 35, 9]. Below we briefly summarize these features.

Lexical Features This class of features captures temporal markers and other phrases indicative of the order between two segments. Representative examples in this category include domain-independent cues like *years earlier* and domain-specific markers like *during next visit.* To automatically identify these phrases, we provide a classifier with two sets of n-grams extracted from the first and the second segments. The classifier then learns phrases with high predictive power.

Temporal Anchor Comparison Temporal anchors are one of the strongest cues for the ordering of events in text. For instance, medical case summaries use phrases like *two days before admission* and *one day before admission* to express relative order between events. If the two segments contain temporal anchors, we can determine their ordering by comparing the relation between the two anchors. We identified a set of temporal anchors commonly used in the medical domain and devised a small set of regular expressions for their comparison.[4] The corresponding feature has three values that encode preceding, following and incompatible relations.

Segment Adjacency Feature Multiple studies have shown that two subsequent sentences are likely to follow a chronological progression [8]. To encode this information, we include a binary feature that captures the adjacency relation between two segments.

2.5.2 Global Inference Strategies for Segment Ordering

Given the scores (or probabilities) of all pairwise edges produced by a local classifier, our task is to construct a TDAG. In this section, we describe three inference strategies that aim to find a consistent ordering between *all* segment pairs. These strategies vary significantly in terms of linguistic motivation and computational complexity. Examples of automatically constructed TDAGs derived from different inference strategies

[4]We could not use standard tools for extraction and analysis of temporal anchors as they were developed on the newspaper corpora, and are not suitable for analysis of medical text [53].

are shown in Figures A-1-A-4 of the appendix.

Greedy Inference in Natural Reading Order (NRO)

The simplest way to construct a consistent TDAG is by adding segments in the order of their appearance in a text. Intuitively speaking, this technique processes segments in the same order as a reader of the text. The motivation underlying this approach is that the reader incrementally builds temporal interpretation of a text; when a new piece of information is introduced, the reader knows how to relate it to already processed text.

This technique starts with an empty graph and incrementally adds nodes in order of their appearance in the text. When a new node is added, we greedily select the edge with the highest score that connects the new node to the existing graph, without violating the consistency of the TDAG. Next, we expand the graph with its transitive closure. We continue greedily adding edges and applying transitive closure until the new node is connected to all other nodes already in the TDAG. The process continues until all the nodes have been added to the graph.

Greedy Best-first Inference (BF)

Our second inference strategy is also greedy. It aims to optimize the score of the graph. The score of the graph is computed by summing the scores of its edges. While this greedy strategy is not guaranteed to find the optimal solution, it finds a reasonable approximation [13].

This method begins by sorting the edges by their score. Starting with an empty graph, we add one edge at a time, without violating the consistency constraints. As in the previous strategy, at each step we expand the graph with its transitive closure. We continue this process until all the edges have been considered.

Exact Inference with Integer Linear Programming (ILP)

We can cast the task of constructing a globally optimal TDAG as an optimization problem. In contrast to the previous approaches, the method is not greedy. It computes the optimal solution within the Integer Linear Programming (ILP) framework.

For a document with N segments, each pair of segments (i, j) can be related in the graph in one of three ways: forward, backward, and null (not connected). Let $s_{i\rightarrow j}$, $s_{i\leftarrow j}$, and $s_{i\nleftrightarrow j}$ be the scores assigned by a local classifier to each of the three relations respectively. Let $I_{i\rightarrow j}$, $I_{i\leftarrow j}$, and $I_{i\nleftrightarrow j}$ be indicator variables that are set to 1 if the corresponding relation is active, or 0 otherwise.

The objective is then to optimize the score of a TDAG by maximizing the sum of the scores of all edges in the graph:

$$\max \sum_{i=1}^{N} \sum_{j=i+1}^{N} s_{i\rightarrow j} I_{i\rightarrow j} + s_{i\leftarrow j} I_{i\leftarrow j} + s_{i\nleftrightarrow j} I_{i\nleftrightarrow j} \tag{2.1}$$

subject to:

$$I_{i\rightarrow j}, I_{i\leftarrow j}, I_{i\nleftrightarrow j} \in \{0,1\} \quad \forall\, i, j = 1, \ldots N, i < j \tag{2.2}$$

$$I_{i\rightarrow j} + I_{i\leftarrow j} + I_{i\nleftrightarrow j} = 1 \quad \forall\, i, j = 1, \ldots N, i < j \tag{2.3}$$

We augment this basic formulation with two more sets of constraints to enforce validity of the constructed TDAG.

Transitivity Constraints The key requirement on the edge assignment is the transitivity of the resulting graph. Transitivity also guarantees that the graph does not have cycles. We enforce transitivity by introducing the following constraint for every triple (i, j, k):

$$I_{i\rightarrow j} + I_{j\rightarrow k} - 1 \leq I_{i\rightarrow k} \tag{2.4}$$

If both indicator variables on the left side of the inequality are set to 1, then the indicator variable on the right side must be equal to 1. Otherwise, the indicator variable on the right can take any value.

Connectivity Constraints The connectivity constraint states that each node i is connected to at least one other node and thereby enforces connectivity of the generated TDAG. We introduce these constraints because manually-constructed TDAGs do not have any disconnected nodes. This observation is consistent with the intuition that the reader is capable to order a segment with respect to other segments in the TDAG.

$$(\sum_{j=1}^{i-1} I_{i\leftrightarrow j} + \sum_{j=i+1}^{N} I_{j\leftrightarrow i}) < N - 1 \tag{2.5}$$

The above constraint rules out edge assignments in which node i has null edges to the rest of the nodes.

Solving ILP Solving an integer linear program is NP-hard [17]. Fortunately, there exist several strategies for solving ILPs. We employ an efficient Mixed Integer Programming solver *lp_solve*[5] which implements the Branch-and-Bound algorithm. It takes less than five seconds to decode each document on a 2.8 GHz Intel Xeon machine.

2.6 Evaluation Setup

We first describe the corpora used in our experiments and the results of human agreement on the segmentation and the ordering tasks. Then, we introduce the evaluation measures that we use to assess the performance of our model.

2.6.1 Corpus Characteristics

We applied our method for temporal ordering to a corpus of medical case summaries. The medical domain has been a popular testbed for methods for automatic temporal analyzers [16, 55]. The appeal is partly due to rich temporal structure of these documents and the practical need to parse this structure for meaningful processing of medical data.

We compiled a corpus of medical case summaries from the online edition of The New England Journal of Medicine.[6] The summaries are written by physicians of

[5] `http://groups.yahoo.com/group/lp_solve`
[6] `http://content.nejm.org`

Massachusetts General Hospital. A typical summary describes an admission status, previous diseases related to the current conditions and their treatments, family history, and the current course of treatment. For privacy protection, names and dates are removed from the summaries before publication.

The average length of a summary is 47 sentences. The summaries are written in the past tense, and a typical summary does not include instances of the past perfect. The summaries do not follow a chronological order. The ordering of information in this domain is guided by stylistic conventions (i.e., symptoms are presented before treatment) and the relevance of information to the current conditions (i.e., previous onset of the same disease is summarized before the description of other diseases).

2.6.2 Annotating Temporal Segmentation

Our approach for temporal segmentation requires annotated data for supervised training. We first conducted a pilot study to assess the human agreement on the task. We employed two annotators to manually segment a portion of our corpus. The annotators were provided with two-page instructions that defined the notion of a temporal segment and included examples of segmented texts. Each annotator segmented eight summaries which on average contained 49 sentences. Because annotators were instructed to consider segmentation boundaries at the level of a clause, there were 877 potential boundaries. The first annotator created 168 boundaries, while the second — 224 boundaries. We computed a Kappa coefficient of 0.71 indicating a high inter-annotator agreement and thereby confirming our hypothesis about the reliability of temporal segmentation.

Once we established high inter-annotator agreement on the pilot study, one annotator segmented the remaining 52 documents in the corpus.[7] Among 3,297 potential boundaries, 1,178 (35.7%) were identified by the annotator as segment boundaries. The average segment length is three sentences, and a typical document contains around 20 segments.

[7]It took approximately 20 minutes to segment a case summary.

2.6.3 Annotating Temporal Ordering

To assess the inter-annotator agreement, we asked two human annotators to construct TDAGs from five manually segmented summaries. These summaries consist of 97 segments, and their transitive closure contain a total of 1,331 edges. We computed the agreement between human judges by comparing the transitive closure of the TDAGs. The annotators achieved a surprisingly high agreement with a Kappa value of 0.98.

After verifying human agreement on this task, one of the annotators constructed TDAGs for another 25 summaries.[8] The transitive reduction of a graph contains on average 20.9 nodes and 20.5 edges. The corpus consists of 72% forward, 12% backward and 16% null segment edges inclusive of edges induced by transitive closure. At the clause level, the distribution is even more skewed — forward edges account for 74% edges, equal for 18%, backward for 3% and null for 5%.

2.6.4 Evaluation Measures

We evaluate temporal segmentation by considering the ratio of correctly predicted boundaries. We quantify the performance using F-measure, a commonly used binary classification metric. We opt not to use the P_k measure, a standard topical segmentation measure, because the temporal segments are short and we are only interested in the identification of the exact boundaries.

Our second evaluation task is concerned with ordering manually annotated segments. In these experiments, we compare an automatically generated TDAG against the annotated reference graph. In essence, we compare edge assignment in the transitive closure of two TDAGs, where each edge can be classified into one of the three types: forward, backward, or null.

Our final evaluation is performed at the clausal level. In this case, each edge can be classified into one of the four classes: forward, backward, equal, or null. Note that the clause-level analysis allows us to compare TDAGs based on the automatically

[8]It took approximately one hour to build a TDAG for each segmented document.

derived segmentation.

2.7 Results

We evaluate temporal segmentation using leave-one-out cross-validation on our corpus of 60 summaries. The segmentation algorithm achieves a performance of 83% F-measure, with a recall of 78% and a precision of 89%.

To evaluate segment ordering, we employ leave-one-out cross-validation on 30 annotated TDAGs that overall contain 13,088 edges in their transitive closure. In addition to the three global inference algorithms, we include a majority baseline that classifies all edges as forward, yielding a chronological order.

Our results for ordering the manually annotated temporal segments are shown in Table 2.1. All inference methods outperform the baseline, and their performance is consistent with the complexity of the inference mechanism. As expected, the ILP strategy, which supports exact global inference, achieves the best performance — 84.3%.

An additional point of comparison is the accuracy of the pairwise classification, prior to the application of global inference. The accuracy of the local ordering is 81.6%, which is lower than that of ILP. The superior performance of ILP demonstrates that accurate global inference can further refine local predictions. Surprisingly, the local classifier yields a higher accuracy than the two other inference strategies. Note, however, the local ordering procedure is not guaranteed to produce a consistent TDAG, and thus the local classifier cannot be used on its own to produce a valid TDAG.

Table 2.2 shows the ordering results at the clausal level. The four-way classification is computed using both manually and automatically generated segments. Pairs of clauses that belong to the same segment stand in the equal relation, otherwise they have the same ordering relation as the segments to which they belong.

On the clausal level, the difference between the performance of ILP and BF is blurred. When evaluated on manually-constructed segments, ILP outperforms BF by

Algorithm	Accuracy
Integer Linear Programming (ILP)	**84.3**
Best First (BF)	78.3
Natural Reading Order (NRO)	74.3
Baseline	72.2

Table 2.1: Accuracy for 3-way ordering classification over manually-constructed segments.

less than 1%. This unexpected result can be explained by the skewed distribution of edge types — the two hardest edge types to classify (see Table 2.3), backward and null, account only for 7.4% of all edges at the clause level.

When evaluated on automatically segmented text, ILP performs slightly worse than BF. We hypothesize that this result can be explained by the difference between training and testing conditions for the pairwise classifier: the classifier is trained on manually-computed segments and is tested on automatically-computed ones, which negatively affects the accuracy on the test set. While all the strategies are negatively influenced by this discrepancy, ILP is particularly vulnerable as it relies on the score values for inference. In contrast, BF only considers the rank between the scores, which may be less affected by noise.

We advocate a two-stage approach for temporal analysis: we first identify segments and then order them. A simpler alternative is to directly perform a four-way classification at the clausal level using the union of features employed in our two-stage process. The accuracy of this approach, however, is low — it achieves only 74%, most likely due to the sparsity of clause-level representation for four-way classification. This result demonstrates the benefits of a coarse representation and a two-stage approach for temporal analysis.

2.8 Conclusions

We introduce a new method for temporal ordering. The unit of our analysis is a temporal segment, a fragment of text that maintains temporal coherence. After in-

Algorithm	Manual Seg.	Automatic Seg.
ILP	**91.9**	84.8
BF	91.0	**85.0**
NRO	87.8	81.0
Baseline	73.6	73.6

Table 2.2: Results for 4-way ordering classification over clauses, computed over manually and automatically generated segments.

Algorithm	Forward	Backward	Null
ILP	**92.5**	**45.6**	**76.0**
BF	91.4	42.2	74.7
NRO	87.7	43.6	66.4

Table 2.3: Per class accuracy for clause classification over manually computed segments.

vestigating several inference strategies, we concluded that integer linear programming and best first greedy approach are valuable alternatives for TDAG construction.

In the future, we will explore a richer set of constraints on the topology on the ordering graph. We will build on the existing formal framework [26] for the verification of ordering consistency. We are also interested in expanding our framework for global inference to other temporal annotation schemes. Given a richer set of temporal relations, the benefits from global inference can be even more significant.

Chapter 3

Randomized Decoding for Selection-and-Ordering Problems

3.1 Introduction

The task of selecting and ordering information appears in multiple contexts in text generation and summarization. For instance, a typical multidocument summarization system creates a summary by selecting a subset of input sentences and ordering them into a coherent text. Selection and ordering at the word level is commonly employed in lexical realization. For instance, in the task of title generation, the headline is constructed by selecting and ordering words from the input text.

Decoding is an essential component of the selection-and-ordering process. Given selection and ordering preferences, the task is to find a sequence of elements that maximally satisfies these preferences. One possible approach for finding such a solution is to decompose it into two tasks: first, select a set of words based on individual selection preferences, and then order the selected units into a well-formed sequence. Although the modularity of this approach is appealing, the decisions made in the selection step cannot be retracted. Therefore, we cannot guarantee that selected units can be ordered in a meaningful way, and we may end up with a suboptimal output.

In this chapter, we investigate decoding methods that simultaneously optimize selection and ordering preferences. We formalize decoding as finding a path in a directed weighted graph.[1] The vertices in the graph represent units with associated selection scores, and the edges represent pairwise ordering preferences. The desired solution is the highest-weighted acyclic path of a prespecified length. The requirement for acyclicity is essential because in a typical selection-and-ordering problem, a well-formed output does not include any repeated units. For instance, a summary of multiple documents should not contain any repeated sentences.

Since the problem is NP-hard, finding an exact solution is challenging. We introduce a novel randomized decoding algorithm based on the idea of color-coding [2]. Although the algorithm is not guaranteed to find the optimal solution on any single run, by increasing the number of runs the algorithm can guarantee an arbitrarily high probability of success. We provide a theoretical analysis that establishes the connection between the required number of runs and the likelihood of finding the correct solution.

Next, we show how to find an exact solution using an integer linear programming (ILP) formulation. Although ILP is NP-hard, this method is guaranteed to compute the optimal solution. This allows us to experimentally investigate the trade-off between the accuracy and the efficiency of decoding algorithms considered in the chapter.

We evaluate the accuracy of the decoding algorithms on the task of title generation. The decoding algorithms introduced in the chapter are compared against beam search, a heuristic search algorithm commonly used for selection-and-ordering and other natural language processing tasks. Our experiments show that the randomized decoder is an appealing alternative to both beam search and ILP when applied to selection-and-ordering problems.

[1]We assume that the scoring function is local; that is, it is computed by combining pairwise scores. In fact, the majority of models that are used to guide ordering (i.e., bigrams) are local scoring functions.

3.2 Problem Formulation

In this section, we formally define the decoding task for selection-and-ordering problems. First, we introduce our graph representation and show an example of its construction for multidocument summarization. (An additional example of graph construction for title generation is given in Section 3.6.) Then, we discuss the complexity of this task and its connection to classical NP-hard problems.

3.2.1 Graph Representation

We represent the set of selection units as the set of vertices V in a weighted directed graph G. The set of edges E represents pairwise ordering scores between all pairs of vertices in V. We also add a special source vertex s and sink vertex t. For each vertex v in V, we add an edge from s to v and an edge from v to t. We then define the set of all vertices as $V^* = V \cup \{s,t\}$, and the set of all edges as $E^* = E \cup \{(s,v)\ \forall\ v \in V\} \cup \{(v,t)\ \forall\ v \in V\}$.

To simplify the representation, we remove all vertex weights in our graph structure and instead shift the weight for each vertex onto its incoming edges. For each pair of distinct vertices $(v,u) \in V$, we set the weight of edge $e_{v,u}$ to be the sum of the logarithms of the selection score for u and the pairwise ordering score of (v,u).

We also enhance our graph representation by grouping sets of vertices into *equivalence classes*. We introduce these classes to control for redundancy as required in many selection-and-ordering problems.[2] For instance, in title generation, an equivalence class may consist of morphological variants of the same stem (i.e., *examine* and *examination*). Because a typical title is unlikely to contain more than one word with the same stem, we can only select a single representative from each class.

Our task is now to find the highest weighted acyclic path starting at s and ending at t with k vertices in between, such that no two vertices belong to the same

[2]An alternative approach for redundancy control would be to represent all the members of an equivalence class as a single vertex in the graph. However, such an approach does not allow us to select the best representative from the class. For instance, one element in the equivalence class may have a highly weighted incoming edge, while another may have a highly weighted outgoing edge.

equivalence class.

3.2.2 Example: Decoding for Multidocument summarization

In multidocument summarization, the vertices in the decoding graph represent sentences from input documents. The vertices may be organized into equivalence classes that correspond to clusters of sentences conveying similar information.[3] The edges in the graph represent the combination of the selection and the ordering scores. The selection scores encode the likelihood of a sentence to be extracted, while pairwise ordering scores capture coherence-based precedence likelihood. The goal of the decoder is to find the sequence of k non-redundant sentences that optimize both the selection and the ordering scores. Finding an acyclic path with the highest weight will achieve this goal.

3.2.3 Relation to Classical Problems

Our path-finding problem may seem to be similar to the tractable shortest paths problem. However, the requirement that the path be long makes it more similar to the the Traveling Salesman Problem (TSP). More precisely, our problem is an instance of the *prize collecting traveling salesman problem*, in which the salesman is required to visit k vertices at best cost [5, 4].

Since our problem is NP-hard, we might be pessimistic about finding an exact solution. But our problem has an important feature: the length k of the path we want to find is small relative to the number of vertices n. This feature distinguishes our task from other decoding problems, such as decoding in machine translation [28], that are modeled using a standard TSP formulation. In general, the connection between n and k opens up a new range of solutions. For example, if we wanted to find the best length-2 path, we could simply try all subsets of 2 vertices in the graph, in all 2 possible orders. This is a set of only $O(n^2)$ possibilities, so we can check all to identify the best in polynomial time.

[3]Such clusters are computed automatically by analyzing lexical similarity of sentences from different documents.

This approach is very limited, however: in general, its runtime of $O(n^k)$ for paths of length k makes it prohibitive for all but the smallest values of k. We cannot really hope to avoid the exponential dependence on k, because doing so would give us a fast solution to an NP-hard problem, but there is hope of making the dependence "less exponential." This is captured by the definition of *fixed parameter tractability* [22]. A problem is fixed parameter tractable if we can make the exponential dependence on the parameter k *independent* of the polynomial dependence on the problem size n. This is the case for our problem: as we will describe below, an algorithm of Alon et al. can be used to achieve a running time of roughly $O(2^k n^2)$. In other words, the path length k only exponentiates a small constant, instead of the problem size n, while the dependence on n is in fact quadratic.

3.3 Related Work

Decoding for selection-and-ordering problems is commonly implemented using beam search [6, 18, 33]. Being heuristic in nature, this algorithm is not guaranteed to find an optimal solution. However, its simplicity and time efficiency make it a decoding algorithm of choice for a wide range of NLP applications. In applications where beam decoding does not yield sufficient accuracy, researchers employ an alternative heuristic search, A* [32, 28]. While in some cases A* is quite effective, in other cases its running time and memory requirements may equal that of an exhaustive search. Time- and memory-bounded modifications of A* (i.e., IDA-A*) do not suffer from this limitation, but they are not guaranteed to find the exact solution. Nor do they provide bounds on the likelihood of finding the exact solution. Newly introduced methods based on local search can effectively examine large areas of a search space [24], but they still suffer from the same limitations.

As an alternative to heuristic search algorithms, researchers also employ exact methods from combinatorial optimization, in particular integer linear programming [28, 44]. While existing ILP solvers find the exact solution eventually, the running time may be too slow for practical applications.

Our randomized decoder represents an important departure from previous approaches to decoding selection-and-ordering problems. The theoretically established bounds on the performance of this algorithm enable us to explicitly control the tradeoff between the quality and the efficiency of the decoding process. This property of our decoder sets it apart from existing heuristic algorithms that cannot guarantee an arbitrarily high probability of success.

3.4 Randomized Decoding with Color-Coding

One might hope to solve decoding with a dynamic program (like that for shortest paths) that grows an optimal path one vertex at a time. The problem is that this dynamic program may grow to include a vertex already on the path, creating a cycle. One way to prevent this is to remember the vertices used on each partial path, but this creates a dynamic program with too many states to compute efficiently.

Instead, we apply a *color coding* technique of Alon et al. [2]. The basic step of the algorithm consists of randomly coloring the graph vertices with a set of colors of size r, and using dynamic programming to find the optimum length-k path *without repeated colors.* (Later, we describe how to determine the number of colors r.) Forbidding repeated colors excludes cycles as required, but remembering only colors on the path requires less state than remembering precisely which vertices are on the path. Since we color randomly, any single iteration is not guaranteed to find the optimal path; in a given coloring, two vertices along the optimal path may be assigned the same color, in which case the optimal path will never be selected. Therefore, the whole process is repeated multiple times, increasing the likelihood of finding an optimal path.

Our algorithm is a variant of the original color-coding algorithm [2], which was developed to detect the existence of paths of length k in an unweighted graph. We modify the original algorithm to find the highest weighted path and also to handle equivalence classes of vertices. In addition, we provide a method for determining the optimal number of colors to use for finding the highest weighted path of length k.

We first describe the dynamic programming algorithm. Next, we provide a probabilistic bound on the likelihood of finding the optimal solution, and present a method for determining the optimal number of colors for a given value of k.

Dynamic Programming Recall that we began with a weighted directed graph G to which we added artificial *start* and *end* vertices s and t. We now posit a *coloring* of that graph that assigns a color c_v to each vertex v aside from s and t. Our dynamic program returns the maximum score path of length $k+2$ (including the artificial vertices s and t) from s to t with no repeated colors.

Our dynamic program grows *colorful paths*—paths with at most one vertex of each color. For a given colorful path, we define the *spectrum* of a path to be the set of colors (each used exactly once) of nodes on the *interior* of the path—we exclude the starting vertex (which will always be s) and the ending vertex. To implement the dynamic program, we maintain a table $q[v, S]$ indexed by a path-ending vertex v and a spectrum S. For vertex v and spectrum S, entry $q[v, S]$ contains the value of the maximum-score colorful path that starts at s, terminates at v, and has spectrum S in its interior.

We initialize the table with length-one paths: $q[v, \emptyset]$ represents the path from s to v, whose spectrum is the empty set since there are no interior vertices. Its value is set to the score of edge (s, v). We then iterate the dynamic program k times in order to build paths of length $k+1$ starting at s. We observe that the optimum colorful path of length ℓ and spectrum S from s to v must consist of an optimum path from s to u (which will already have been found by the dynamic program) concatenated to the edge (u, v). When we concatenate (u, v), vertex u becomes an interior vertex of the path, and so its color must not be in the preexisting path's spectrum, but joins the spectrum of the path we build. It follows that

$$q[v, S] = \max_{(u,v)\in G, c_u \in S, c_v \notin S} q[u, S - \{c_u\}] + w(u, v)$$

After k iterations, for each vertex v we will have a list of optimal paths from s to v of length $k+1$ with all possible spectra. The optimum length-$k+2$ colorful path from s to t must follow the optimum length-$k+1$ path of *some* spectrum to *some* penultimate vertex v and then proceed to vertex t; we find it by iterating over all such possible spectra and all vertices v to determine $argmax_{v,S}q[v, S] + w(v, t)$.

Amplification The algorithm of Alon et al., and the variant we describe, are somewhat peculiar in that the probability of finding the optimal solution in one coloring iteration is quite small. But this can easily be dealt with using a standard technique called *amplification* [40]. Suppose that the algorithm succeeds with small probability p, but that we would like it to succeed with probability $1 - \delta$ where δ is very small. We run the algorithm $t = (1/p) \ln 1/\delta$ times. The probability that the algorithm fails every single run is then $(1 - p)^t \leq e^{-pt} = \delta$. But if the algorithm succeeds on even one run, then we will find the optimum answer (by taking the best of the answers we see).

No matter how many times we run the algorithm, we cannot absolutely guarantee an optimal answer. However, the chance of failure can easily be driven to negligible levels—achieving, say, a one-in-a-billion chance of failure requires only $20/p$ iterations by the previous analysis.

Determining the number of colors Suppose that we use r random colors and want to achieve a given failure probability δ. The probability that the optimal path has no repeated colors is:

$$1 \cdot \frac{r-1}{r} \cdot \frac{r-2}{r} \cdots \frac{r-(k-1)}{r}.$$

By the amplification analysis, the number of trials needed to drive the failure probability to the desired level will be inversely proportional to this quantity. At the same time, the dynamic programming table at each vertex will have size 2^r (indexing on a bit vector of colors used per path), and the runtime of each trial will be proportional to this. Thus, the running time for the necessary number of trials T_r will be proportional to

$$1 \cdot \frac{r}{r-1} \cdot \frac{r}{r-2} \cdots \frac{r}{r-(k-1)} \cdot 2^r$$

What $r \geq k$ should we choose to minimize this quantity? To answer, let us consider the ratio:

$$\begin{aligned}\frac{T_{r+1}}{T_r} &= \left(\frac{r+1}{r}\right)^k \cdot \frac{r-(k-1)}{r+1} \cdot 2 \\ &= 2(1+1/r)^k(1-k/(r+1))\end{aligned}$$

If this ratio is less than 1, then using $r+1$ colors will be faster than using r; otherwise it will be slower. When r is very close to k, the above equation is tiny, indicating that one should increase r. When $r \gg k$, the above equation is huge, indicating one should decrease r. Somewhere in between, the ratio passes through 1, indicating the optimum point where neither increasing nor decreasing r will help. If we write $\alpha = k/r$, and consider large k, then $\frac{T_{r+1}}{T_r}$ converges to $2e^{\alpha}(1-\alpha)$. Solving numerically to find where this is equal to 1, we find $\alpha \approx .76804$, which yields a running time proportional to approximately $(3/2)^k$.

In practice, rather than using an (approximate) formula for the optimum r, one should simply plug all values of r in the range $[k, 2k]$ into the running-time formula in order to determine the best; doing so takes negligible time.

3.5 Decoding with Integer Linear Programming

In this section, we show how to formulate the selection-and-ordering problem in the ILP framework. We represent each edge (i, j) from vertex i to vertex j with an indicator variable $I_{i,j}$ that is set to 1 if the edge is selected for the optimal path and 0 otherwise. In addition, the associated weight of the edge is represented by a constant $w_{i,j}$.

The objective is then to maximize the following sum:

$$\max_I \sum_{i \in V} \sum_{j \in V} w_{i,j} I_{i,j} \tag{3.1}$$

This sum combines the weights of edges selected to be on the optimal path.

To ensure that the selected edges form a valid acyclic path starting at s and ending at t, we introduce the following constraints:

Source-Sink Constraints Exactly one edge originating at source s is selected:

$$\sum_{j \in V} I_{s,j} = 1 \tag{3.2}$$

Exactly one edge ending at sink t is selected:

$$\sum_{i \in V} I_{i,t} = 1 \tag{3.3}$$

Length Constraint Exactly $k + 1$ edges are selected:

$$\sum_{i \in V} \sum_{j \in V} I_{i,j} = k + 1 \tag{3.4}$$

The $k + 1$ selected edges connect $k + 2$ vertices including s and t.

Balance Constraints Every vertex $v \in V$ has in-degree equal to its out-degree:

$$\sum_{i \in V} I_{i,v} = \sum_{i \in V} I_{v,j} \quad \forall\, v \in V^* \tag{3.5}$$

Note that with this constraint, a vertex can have at most one outgoing and one incoming edge.

Redundancy Constraints To control for redundancy, we require that at most one representative from each equivalence class is selected. Let Z be a set of vertices that belong to the same equivalence class. For every equivalence class Z, we force the total out-degree of all vertices in Z to be at most 1.

$$\sum_{i \in Z} \sum_{j \in V} I_{i,j} \leq 1 \quad \forall\, Z \subseteq V \tag{3.6}$$

Acyclicity Constraints The constraints introduced above do not fully prohibit the presence of cycles in the selected subgraph. Figure 3-1 shows an example of a selected subgraph that contains a cycle while satisfying all the above constraints.

We force acyclicity with an additional set of variables. The variables $f_{i,j}$ are

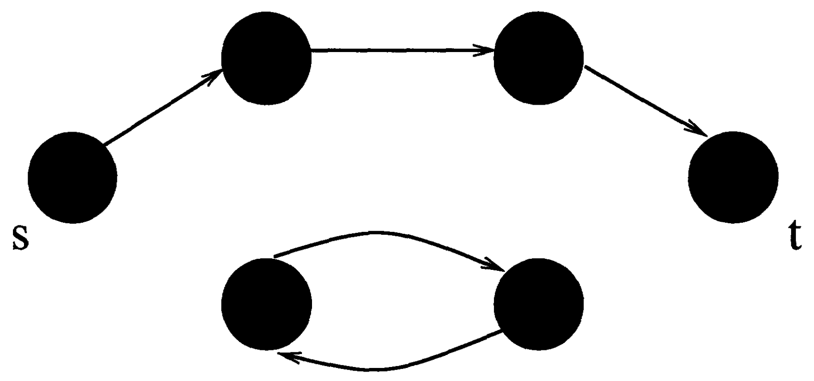


Figure 3-1: A subgraph that contains a cycle, while satisfying constraints 3.2 through 3.5.

intended to number the edges on the path from 1 to $k+1$, with the first edge getting number $f_{i,j} = k+1$, and the last getting number $f_{i,j} = 1$. All other edges will get $f_{i,j} = 0$. To enforce this, we start by ensuring that only the edges selected for the path ($I_{i,j} = 1$) get nonzero f-values:

$$0 \leq f_{i,j} \leq (k+1)\ I_{i,j} \quad \forall\ i,j \in V \tag{3.7}$$

When $I_{i,j} = 0$, this constraint forces $f_{i,j} = 0$. When $I_{i,j} = 1$, this allows $0 \leq f_{i,j} \leq k+1$. Now we introduce additional variables and constraints. We constrain demand variables d_v by:

$$d_v = \sum_{i \in V} I_{i,v} \quad \forall\ v \in V^* - \{s\} \tag{3.8}$$

The right hand side sums the number of selected edges entering v, and will therefore be either 0 or 1. Next we add variables a_v and b_v constrained by the equations:

$$a_v = \sum_{i \in V} f_{i,v} \tag{3.9}$$
$$b_v = \sum_{i \in V} f_{v,i} \tag{3.10}$$

Note that a_v sums over f values on all edges entering v. However, by the previous constraints those f-values can only be nonzero on the (at most one) selected edge entering v. So, a_v is simply the f-value on the selected edge entering v, if one exists, and 0 otherwise. Similarly, b_v is the f-value on the (at most one) selected edge leaving v.

Finally, we add the constraints

$$\begin{aligned} a_v - b_v &= d_v \quad v \neq s & (3.11) \\ b_s &= k+1 & (3.12) \\ a_t &= 1 & (3.13) \end{aligned}$$

These last constraints let us argue, by induction, that a path of length exactly $k+1$ must run from s to t, as follows. The previous constraints forced exactly one edge leaving s, to some vertex v, to be selected. The constraint $b_s = k+1$ means that the f-value on this edge must be $k+1$. The balance constraint on v means some edge must be selected leaving v. The constraint $a_v - b_v = d_v$ means this edge must have f-value k. The argument continues the same way, building up a path. The balance constraints mean that the path must terminate at t, and the constraint that $a_t = 1$ forces that termination to happen after exactly $k+1$ edges.[4]

For those familiar with max-flow, our program can be understood as follows. The variables I force a flow, of value 1, from s to t. The variables f represent a flow with supply $k+1$ at s and demand d_v at v, being forced to obey "capacity constraints" that let the flow travel only along edges with $I = 1$.

3.6 Experimental Set-Up

Task We applied our decoding algorithm to the task of title generation. This task has been extensively studied over the last six years [6, 33, 50]. Title generation is a classic selection-and-ordering problem: during title realization, an algorithm has to take into account both the likelihood of words appearing in the title and their ordering preferences. In the previous approaches, beam search has been used for

[4]The network flow constraints allow us to remove the previously placed length constraint.

decoding. Therefore, it is natural to explore more sophisticated decoding techniques like the ones described in this chapter.

Our method for estimation of selection-and-ordering preferences is based on the technique described in [6]. We compute the likelihood of a word in the document appearing in the title using a maximum entropy classifier. Every stem is represented by commonly used positional and distributional features, such as location of the first sentence that contains the stem and its TF*IDF. We estimate the ordering preferences using a bigram language model with Good-Turing smoothing.

In previous systems, the title length is either provided to a decoder as a parameter, or heuristics are used to determine it. Since exploration of these heuristics is not the focus of this thesis, we provide the decoder with the actual title length (as measured by the number of content words).

Graph Construction We construct a decoding graph in the following fashion. Every unique content word comprises a vertex in the graph. All the morphological variants of a stem belong to the same equivalence class. An edge (v, u) in the graph encodes the selection preference of u and the likelihood of the transition from v to u.

Note that the graph does not contain any auxiliary words in its vertices. We handle the insertion of auxiliary words by inserting additional edges. For every auxiliary word x, we add one edge representing the transition from v to u via x, and the selection preference of u. The auxiliary word set consists of 24 prepositions and articles extracted from the corpus.

Corpus Our corpus consists of 547 sections of a commonly used undergraduate algorithms textbook. The average section contains 609.2 words. A title, on average, contains 3.7 words, among which 3.0 are content words; the shortest and longest titles have 1 and 13 words respectively. Our training set consists of the first 382 sections, the remaining 165 sections are used for testing. The bigram language model is estimated from the body text of all sections in the corpus, consisting of 461,351 tokens.

To assess the importance of the acyclicity constraint, we compute the number of titles that have repeated content words. The empirical findings support our assumption: 97.9% of the titles do not contain repeated words.

Decoding Algorithms We consider three decoding algorithms: our color-coding algorithm, ILP, and beam search.[5] The beam search algorithm can only consider vertices which are not already in the path.[6]

To solve the ILP formulations, we employ a Mixed Integer Programming solver *lp_solve* which implements the Branch-and-Bound algorithm. We implemented the rest of the decoders in Python with the Psyco speed-up module. We put substantial effort to optimize the performance of all of the algorithms. The color-coding algorithm is implemented using parallelized computation of coloring iterations.

3.7 Results

Table 3.1 shows the performance of various decoding algorithms considered in the chapter. We first evaluate each algorithm by the running times it requires to find all the optimal solutions on the test set. Since ILP is guaranteed to find the optimal solution, we can use its output as a point of comparison. Table 3.1 lists both the average and the median running times. For some of the decoding algorithms, the difference between the two measurements is striking — 6,536 seconds versus 57.3 seconds for ILP. This gap can be explained by outliers which greatly increase the average running time. For instance, in the worst case, ILP takes an astounding 136 hours to find the optimal solution. Therefore, we base our comparison on the median running time.

The color-coding algorithm requires a median time of 9.7 seconds to find an optimal solution compared to the 57.3 seconds taken by ILP. Furthermore, as Figure 3-2 shows, the algorithm converges quickly: just eleven iterations are required to find an optimal solution in 90% of the titles, and within 35 iterations all of the solutions are found. An alternative method for finding optimal solutions is to employ a beam

[5]The combination of the acyclicity and path length constraints require an exponential number of states for A* since each state has to preserve the history information. This prevents us from applying A* to this problem.

[6]Similarly, we avoid redundancy by disallowing two vertices from the same equivalence class to belong to the same path.

	Average (s)	Median (s)	ROUGE-L	Optimal Solutions (%)
Beam 1	0.6	0.4	0.0234	0.0
Beam 80	28.4	19.3	0.2373	64.8
Beam 1345	368.6	224.4	0.2556	100.0
ILP	6,536.2	57.3	0.2556	100.0
Color-coding	**73.8**	**9.7**	**0.2556**	**100.0**

Table 3.1: Running times in seconds, ROUGE scores, and percentage of optimal solutions found for each of the decoding algorithms.

search with a large beam size. We found that for our test set, the smallest beam size that satisfies this condition is 1345, making it twenty-three times slower than the randomized decoder with respect to the median running time.

Does the decoding accuracy impact the quality of the generated titles? We can always trade speed for accuracy in heuristic search algorithms. As an extreme, consider a beam search with a beam of size 1: while it is very fast with a median running time of less than one second, it is unable to find any of the optimal solutions. The titles generated by this method have substantially lower scores than those produced by the optimal decoder, yielding a 0.2322 point difference in ROUGE[7] scores.[8] Even a larger beam size such as 80 (as used by Banko et al. [6]) does not match the title quality of the optimal decoder.

3.8 Conclusions

In this chapter, we formalized the decoding task for selection-and-ordering as a problem of finding the highest-weighted acyclic path in a directed graph. The presented decoding algorithm employs randomized color-coding, and can closely approximate the ILP performance, without blowing up the running time. The algorithm has been tested on title generation, but the decoder is not specific to this task and can be applied to other generation and summarization applications.

[7]http://www.isi.edu/licensed-sw/see/rouge/

[8]Our accuracy of title generation is comparable with the performance of other systems reported in the literature [50].

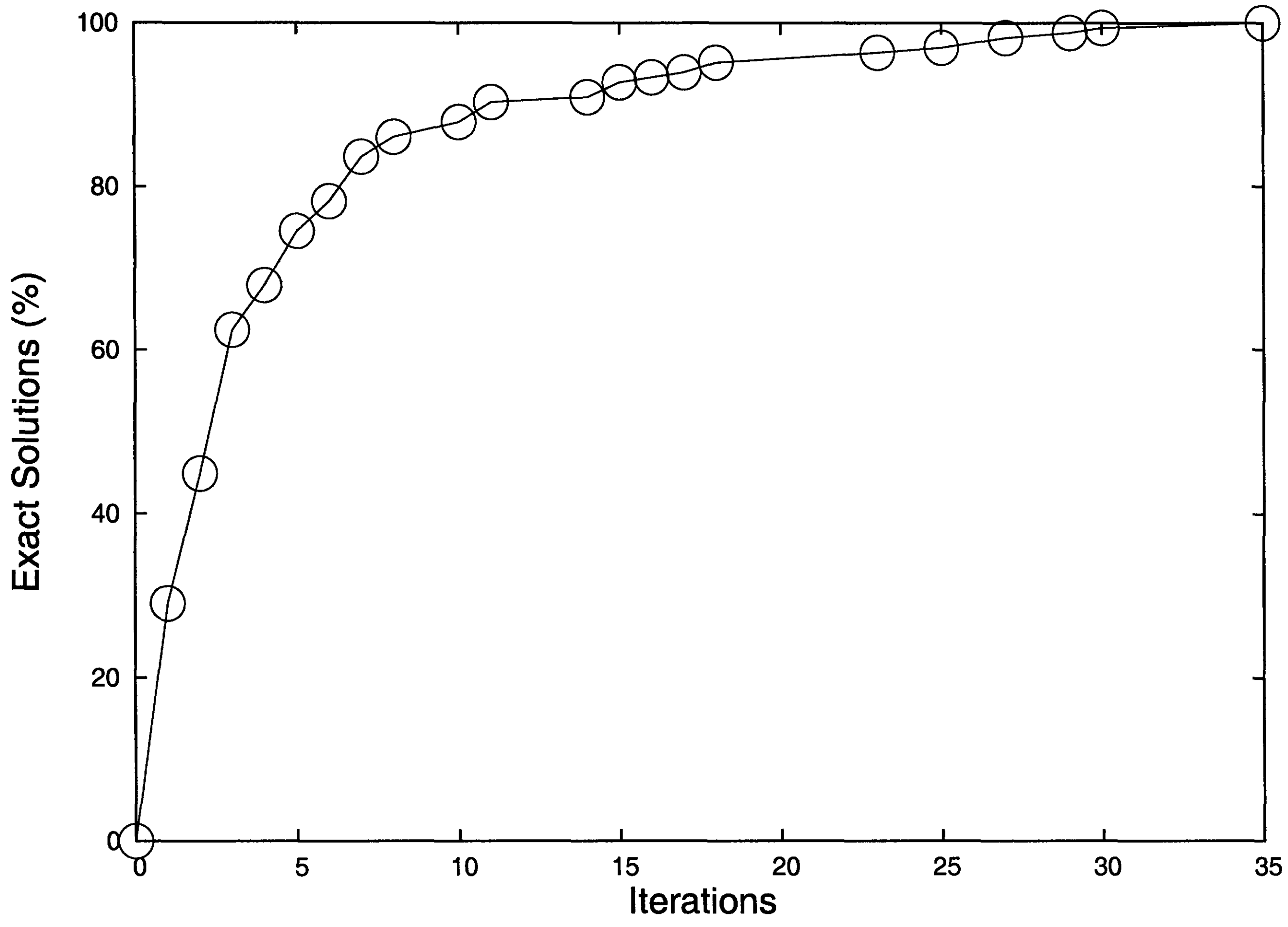


Figure 3-2: The proportion of exact solutions found for each iteration of the color coding algorithm.

Chapter 4

Generating a Table-of-Contents: A Hierarchical Discriminative Approach

4.1 Introduction

Current research in summarization focuses on processing short articles, primarily in the news domain. While in practice the existing summarization methods are not limited to this material, they are not universal: texts in many domains and genres cannot be summarized using these techniques. A particularly significant challenge is the summarization of longer texts, such as books. The requirement for high compression rates and the increased need for the preservation of contextual dependencies between summary sentences places summarization of such texts beyond the scope of current methods.

In this chapter, we investigate the automatic generation of *tables-of-contents*, a type of indicative summary particularly suited for accessing information in long texts. A typical table-of-contents lists topics described in the source text and provides information about their location in the text. The hierarchical organization of information in the table further refines information access by specifying the relations between dif-

- Scientific computing
  - Remarkable recursive algorithm for multiplying matrices
    - Divide and conquer algorithm design
    - Making a recursive algorithm
  - Solving systems of linear equations
    - Computing an LUP decomposition
    - Forward and back substitution
  - Symmetric positive definite matrices and least squares approximation

Figure 4-1: A fragment of a table-of-contents generated by our method.

ferent topics and providing rich contextual information during browsing. Commonly found in books, tables-of-contents can also facilitate access to other types of texts. For instance, this type of summary could serve as an effective navigation tool for understanding a long, unstructured transcript for an academic lecture or a meeting.

Given a text, our goal is to generate a tree wherein a node represents a segment of text and a title that summarizes its content. This process involves two tasks: the hierarchical segmentation of the text, and the generation of informative titles for each segment. The first task can be addressed using existing topic segmentation algorithms, either by directly applying hierarchical methods [54] or by repeatedly applying linear segmentation algorithms [30]. The second task also seems simple: we can just employ existing methods of title generation to each segment, and combine the results into a tree structure.

However, the latter approach cannot guarantee that the generated table-of-contents forms a coherent representation of the entire text. Since titles of different segments are generated in isolation, some of the generated titles may be repetitive. Even non-repetitive titles may not provide sufficient information to discriminate between the content of one segment and another. Therefore, it is essential to generate an entire content tree in a concerted fashion.

We present a hierarchical discriminative approach for table-of-contents generation. Figure 4-1 shows a fragment of a table-of-contents automatically generated by this approach. Our method has two important points of departure from existing techniques. First, we introduce a structured discriminative model for table-of-contents generation that accounts for a wide range of phrase-based and collocational features. The flexibility of this model results in improved summary quality. Second, our model captures both global dependencies across different titles in the tree and local dependencies within sections. We decompose the model into local and global components that handle different classes of dependencies. We further reduce the search space through incremental construction of the model's output by considering only the promising parts of the decision space.

We apply our method to process a 1,180 page algorithms textbook. To assess the contribution of our hierarchical model, we compare our method with state-of-the-art methods that generate each segment title independently. The results of automatic evaluation and manual assessment of title quality show that the output of our system is consistently ranked higher than that of non-hierarchical baselines.

4.2 Related Work

Even though most current research in summarization focuses on newspaper articles, a number of approaches have been developed for processing longer texts. Most of these approaches are tailored to a particular domain, such as medical literature or scientific articles. By making strong assumptions about the input structure and the desired format of the output, these methods achieve a high compression rate while preserving summary coherence. For instance, Teufel and Moens [48] summarize scientific articles by selecting rhetorical elements that are commonly present in scientific abstracts. Elhadad and McKeown [25] generate summaries of medical articles by following a certain structural template in content selection and realization.

Our work, however, is closer to domain-independent methods for summarizing long texts. Typically, these approaches employ topic segmentation to identify a list of topics described in a document, and then produce a summary for each part [10, 3]. In contrast to our method, these approaches perform either sentence or phrase extraction, rather than summary generation. Moreover, extraction for each segment is performed in isolation, and global constraints on the summary are not enforced.

Finally, our work is also related to research on title generation [6, 33, 21]. Since work in this area focuses on generating titles for newspaper articles, the issue of hierarchical generation, which is unique to our task, does not arise in that application. However, this is not the only novel aspect of the proposed approach. Our model learns title generation in a fully discriminative framework, in contrast to the commonly used noisy-channel model. Thus, instead of independently modeling the selection and grammatical constraints, we learn both types of features in a single framework. This joint training regime supports greater flexibility in capturing a range of contextual features and modeling their interaction.

4.3 Problem Formulation

We formalize the problem of table-of-contents generation as a supervised learning task where the goal is to map a tree of text segments S to a tree of titles T. A segment may correspond to a chapter, section or subsection.

Since the focus of our work is on the generation aspect of table-of-contents construction, we assume that the hierarchical segmentation of a text is provided in the input. This division can either be automatically computed using one of the many available text segmentation algorithms [30, 54], or it can be based on demarcations already present in the input (e.g., paragraph markers).

During training, the algorithm is provided with a set of pairs (S^i, T^i) for $i = 1, \ldots, p$, where S^i is the i^{th} tree of text segments, and T^i is the table-of-contents for that tree. During testing, the algorithm generates tables-of-contents for unseen trees of text segments.

We also assume that during testing the desired title length is provided as a parameter to the algorithm.

4.4 Algorithm

To generate a coherent table-of-contents, we need to take into account multiple constraints: the titles should be grammatical, they should adequately represent the content of their segments, and the table-of-contents as a whole should clearly convey the relations between the segments. Taking a discriminative approach for modeling this task would allow us to achieve this goal: we can easily integrate a range of constraints in a flexible manner. Since the number of possible labels (i.e., tables-of-contents) is prohibitively large and the labels themselves exhibit a rich internal structure, we employ a structured discriminative model that can easily handle complex dependencies. Our solution relies on two orthogonal strategies to balance tractability and the richness of the model. First, we factor the model into local and global components. Second, we incrementally construct the output of each component using a search-based discriminative algorithm. Both of these strategies have the effect of intelligently pruning the decision space.

Our model factorization is driven by the different types of dependencies which are captured by the two components. The first model is *local*: for each segment, it generates a list of candidate titles ranked by their individual likelihoods. This model focuses on grammaticality and word selection constraints, but it does not consider relations among different titles in the table-of-contents. These latter dependencies are captured in the *global* model that constructs a table-of-contents by selecting titles for each segment from the available candidates. Even after this factorization, the decision space for each model is large: for the local model, it is exponential in the length of the segment title, and for the global model it is exponential in the size of the tree.

Therefore, we construct the output for each of these models *incrementally* using beam search. The algorithm maintains the most promising partial output structures, which are extended at every iteration. The model incorporates this decoding procedure into the training process, thereby learning model parameters best suited for the specific decoding algorithm. Similar models have been successfully applied in the past to other tasks including parsing [15], chunking [20], and machine translation [19].

4.4.1 Model Structure

The model takes as input a tree of text segments S. Each segment $s \in S$ and its title z are represented as a *local feature vector* $\Phi_{\mathbf{loc}}(s, z)$. Each component of this vector stores a numerical value. This feature vector can track any feature of the segment s together with its title z. [1] For instance, the i^{th} component of this vector may indicate whether the bigram $(z[j]z[j+1])$ occurs in s, where $z[j]$ is the j^{th} word in z:

$$(\Phi_{\mathbf{loc}}(s, z))_i = \begin{cases} 1 & \text{if } (z[j]z[j+1]) \in s \\ 0 & \text{otherwise} \end{cases}$$

In addition, our model captures dependencies among *multiple titles* that appear in the same table-of-contents. We represent a tree of segments S paired with titles T with the *global feature vector* $\Phi_{\mathbf{glob}}(S, T)$. The components here are also numerical features. For example, the i^{th} component of the vector may indicate whether a title is repeated in the table-of-contents T:

$$(\Phi_{\mathbf{glob}}(S, T))_i = \begin{cases} 1 & \text{repeated title} \\ 0 & \text{otherwise} \end{cases}$$

Our model constructs a table-of-contents in two basic steps:

Step One The goal of this step is to generate a list of k candidate titles for each segment $s \in S$. To do so, for each possible title z, the model maps the feature vector $\Phi_{\mathbf{loc}}(s, z)$ to a real number. This mapping can take the form of a linear model,

$$\Phi_{\mathbf{loc}}(s, z) \cdot \alpha_{\text{loc}}$$

where α_{loc} is the local parameter vector.

Since the number of possible titles is exponential, we cannot consider all of them. Instead, we prune the decision space by incrementally constructing promising titles. At each iteration j, the algorithm maintains a beam Q of the top k partially generated titles of length j. During iteration $j+1$, a new set of candidates is grown by

[1] In practice, $\Phi_{\mathbf{loc}}(s, z)$ also tracks features of the neighbhoring segments of s.

appending a word from s to the right of each member of the beam Q. We then sort the entries in Q: $z_1, z_2, \dots$ such that $\Phi_{\mathbf{loc}}(s, z_i) \cdot \alpha_{\text{loc}} \geq \Phi_{\mathbf{loc}}(s, z_{i+1}) \cdot \alpha_{\text{loc}}, \forall i$. Only the top k candidates are retained, forming the beam for the next iteration. This process continues until a title of the desired length is generated. Finally, the list of k candidates is returned.

Step Two Given a set of candidate titles $z_1, z_2, \dots, z_k$ for each segment $s \in S$, our goal is to construct a table-of-contents T by selecting the most appropriate title from each segment's candidate list. To do so, our model computes a score for the pair (S, T) based on the global feature vector $\Phi_{\mathbf{glob}}(S, T)$:

$$\Phi_{\mathbf{glob}}(S, T) \cdot \alpha_{\text{glob}}$$

where α_{glob} is the global parameter vector.

As with the local model (step one), the number of possible tables-of-contents is too large to be considered exhaustively. Therefore, we incrementally construct a table-of-contents by traversing the tree of segments in a pre-order walk (i.e., the order in which segments appear in the text). In this case, the beam contains partially generated tables-of-contents, which are expanded by one segment title at a time.

4.4.2 Training the Model

Training for Step One We now describe how the local parameter vector α_{loc} is estimated from training examples. We are given a set of training examples (s^i, y^i) for $i = 1, \dots, l$, where s^i is the i^{th} text segment, and y^i is the title of this segment.

This linear model is learned using a variant of the incremental perceptron algorithm [15, 20]. This on-line algorithm traverses the training set multiple times, updating the parameter vector α_{loc} after each training example. The algorithm encourages a setting of the parameter vector α_{loc} that assigns the highest score to the feature vector associated with the correct title.

The pseudocode of the algorithm is shown in Figure 4-2. Given a segment text s and the corresponding title y, the training algorithm maintains a beam Q containing the top k partial titles of length j. The beam is updated on each iteration using the functions GROW and PRUNE. For every word in segment s and for every title in Q, GROW creates a new title by appending this word to the title. PRUNE retains only the top ranked candidates based on the scoring function $\Phi_{\mathbf{loc}}(s, z) \cdot \alpha_{\mathrm{loc}}$. If $y[1 \dots j]$ (i.e., the prefix of y of length j) is not in the modified beam Q, then α_{loc} is updated[2] as shown in line 4 of the pseudocode in Figure 4-2. In addition, Q is replaced with a beam containing only $y[1 \dots j]$ (line 5). This process is performed $|y|$ times. We repeat this process for all training examples over a specified number of training iterations.[3]

```
$s$ – segment text.
$y$ – segment title.
$y[1 \dots j]$ – prefix of $y$ of length $j$.
$Q$ – beam containing partial titles.

1. for $j = 1 \dots |y|$
2.     $Q = \mathrm{PRUNE}(\mathrm{GROW}(s, Q))$
3.     if $y[1 \dots j] \notin Q$
4.         $\alpha_{\mathrm{loc}} = \alpha_{\mathrm{loc}} + \Phi_{\mathbf{loc}}(s, y[1 \dots j])$
               $- \sum_{z \in Q} \frac{\Phi_{\mathbf{loc}}(s,z)}{|Q|}$
5.         $Q = \{y[1 \dots j]\}$
```

Figure 4-2: The training algorithm for the local model.

Training for Step Two To train the global parameter vector α_{glob}, we are given training examples (S^i, T^i) for $i = 1, \dots, p$, where S^i is the i^{th} tree of text segments, and T^i is the table-of-contents for that tree. However, we cannot directly use these tables-of-contents for training our global model. Since this model selects one of the candidate titles $z_1^i, \dots, z_k^i$ returned by the local model, the true title of the segment may not be among these candidates. Therefore, to determine a new target title for the segment, we need to identify the title in the set of candidates that is closest to the true title.

[2]If the word in the j^{th} position of y does not occur in s, then the parameter update is not performed.

[3]For decoding, α_{loc} is averaged over the training iterations as in Collins and Roark [15].

We employ the L_1 distance measure to compare the content word overlap between two titles.[4] For each input (S, T), and each segment $s \in S$, we identify the segment title closest in the L_1 measure to the true title y:

$$z^* = \arg\min_i L_1(z_i, y)$$

Once all the training targets in the corpus have been identified through this procedure, the global linear model $\Phi_{\mathbf{glob}}(S, T) \cdot \alpha_{\mathbf{glob}}$ is learned using the same perceptron algorithm as in step one. Rather than maintaining the beam of partially generated titles, the beam Q holds partially generated tables-of-contents. Also, the loop in line 1 of Figure 4-2 iterates over segment titles rather than words.

4.5 Features

Local Features Our local model aims to generate titles which adequately represent the meaning of the segment and are grammatical. Selection and contextual preferences are encoded in the local features. The features that capture selection constraints are specified at the word level, and contextual features are expressed at the word sequence level.

The selection features capture the position of the word, its TF*IDF, and part-of-speech information. In addition, they also record whether the word occurs in the body of neighboring segments. We also generate conjunctive features by combining features of different types.

The contextual features record the trigram language model scores, one for words and one for part-of-speech tags. The trigram scores are averaged over the title. Both language models[5] are trained using the SRILM toolkit. Another type of contextual feature models the collocational properties of noun phrases in the title. This feature aims to eliminate from titles generic phrases, such as *"the following section"*.[6] To

[4]This measure is close to ROUGE-1 which in addition considers the overlap in auxiliary words.

[5]Witten-Bell discounting is used for the part-of-speech language model, and Chen and Goodman's modified Kneser-Ney discounting is used for the lexical language model.

[6]Unfortunately, we could not use more sophisticated syntactic features due to the low accuracy

Segment has the same title as its sibling
Segment has the same title as its parent
Two adjacent segment titles have the same head
Two adjacent segment titles start with the same word
Rank given to the title by the local model

Table 4.1: Examples of global features.

	Training	Testing
# Titles	436	104
# Trees	31	8
Tree Depth	4	4
# Words	206,319	62,070
Avg. Title Length	3.74	3.57
Avg. Branching	3.38	2.95
Avg. Title Duplicates	12	9

Table 4.2: Statistics on the corpus used in the experiments.

achieve this effect, for each noun phrase in the title, we measure the ratio of their frequency in the segment to their frequency in the corpus.

Global Features Our global model describes the interaction between different titles in the tree (See Table 4.1). These interactions are encoded in three types of global features. The first type of global feature indicates whether titles in the tree are redundant at various levels of the tree structure. The second type of feature encourages parallel constructions within the same tree. For instance, titles in the same segment may all be verbalized as noun phrases with the same head (e.g., *"Bubble sort algorithm"*, *"Merge sort algorithm"*). We capture this property by comparing words that appear in certain positions in adjacent sibling titles. Finally, our global model also uses the rank of the title provided by the local model. This feature enables the global model to account for the preferences of the local model in the title selection process.

4.6 Evaluation Setup

Data We apply our method to an undergraduate algorithms textbook. We divide its table-of-contents into training and testing portions by splitting it into a set of independent subtrees. Given a table-of-contents of depth n with a root branching factor of r, we generate r subtrees, with a depth of at most $n - 1$. We randomly select 80% of these trees for training, and the rest are used for testing. For detailed statistics on the training and testing data see Table 4.2.

Admittedly, this method for generating training and testing data omits some dependencies at the level of the table-of-contents as a whole. However, the subtrees used in our experiments still exhibit a sufficiently deep hierarchical structure, rich with contextual dependencies.

Baselines As an alternative to our hierarchical method, we consider three baselines that build a table-of-contents by generating a title for each segment individually, without taking into account the tree structure. The first method generates a title for a segment by selecting the noun phrase from that segment with the highest TF*IDF.[7] This simple method is commonly used to generate keywords for browsing applications in information retrieval, and has been shown to be effective for summarizing technical content [49].

The second baseline is based on the noisy-channel generative model proposed by Banko et al. [6]. Similar to our local model, this method captures both selection and grammatical constraints. However, these constraints are modeled separately, and then combined in a generative framework.

We use our local model as the third baseline. Like the second baseline, this model omits global dependencies, and only focuses on features that capture relations within individual segments. This model is close to traditional methods for title generation as it does not have access to hierarchical information. Note that we could not compare against other methods for title generation because they are designed exclusively for

of statistical parsers on our corpus.

[7]This is the only baseline that does not use a length parameter in the selection process.

newspaper articles [21, 50].

The last two baselines and our algorithm are provided with the title length as a parameter. In our experiments, the algorithms use the reference title length.

Experimental Design: Comparison with reference tables-of-contents Reference based evaluation is most commonly used to assess the quality of machine-generated headlines [50]. We compare our system's output with the table-of-contents from the textbook using ROUGE metrics. We employ a publicly available software package,[8] with all the parameters set to default values.

Experimental Design: Human assessment In this experiment, human judges evaluated the generated tables-of-contents. For each test segment, the judges were presented with its text, and a list of alternative titles consisting of the reference and the automatically generated titles. The system identities were hidden from the judges, and the titles were presented in random order. The judges rated the titles from one to five based on how well they represent the content of the segment. We compute the score of a system by averaging its ratings over all the test instances.

Four people participated in this experiment. All the participants were graduate students in computer science who had taken the algorithms class in the past and were reasonably familiar with the material.

Parameter Settings For the local model, we train and test with a beam size of 50, while for the global model we use a beam size of 250. We used 50 and 170 iterations for training the local and the global models, respectively.

4.7 Results

Figure 4-3 shows fragments of the tables-of-contents generated by our method and the three baselines along with the reference counterpart. These extracts illustrate three general phenomena that we observed in the test corpus. First, the titles produced by

[8]http://www.isi.edu/licensed-sw/see/rouge/

Reference:

- NP Completeness
  - Polynomial time
    - Abstract problems
    - Encodings
    - A formal language framework
  - NP completeness proofs
    - Formula satisfiability
    - 3 CNF satisfiability
  - NP complete problems

Keyword Extraction:

- Polynomial time
  - Polynomial time
    - Decision problem
    - Polynomial time
    - Polynomial time
  - Circuit satisfiability problem
    - Polynomial time
    - CNF formula
  - Vertex cover

Noisy-channel:

- Shortest paths
  - Running time
    - Shortest paths
    - Braces
    - Finding a shortest path
  - Reducing the algorithm
    - Combinational circuit
    - Conjunctive normal form
  - Dynamic programming algorithm

Local Discriminative:

- Worst case
  - NP completeness
    - Polynomial time
    - Programs
    - Using a decision problem
  - Circuit satisfiability problem
    - NP completeness
    - Conjunctive normal form
  - Optimal solutions to

Hierarchical Discriminative:

- Polynomial time
  - NP completeness
    - Abstract problems
    - Programs
    - Using a decision problem
  - Formula satisfiability problems
    - Satisfiability problem
    - CNF satisfiability problem
  - NP complete problems

Figure 4-3: Fragments of tables-of-contents generated by our method and the three baselines along with the corresponding reference.

	Unigram	Bigram	Full
Reference	83.5%	54.4%	35.5%
Hierarchical	**100.0%**	**66.1%**	**56.7%**
Noisy-channel	100.0%	64.4%	42.3%
Local	100.0%	71.0%	61.5%
Keyword	100.0%	100.0%	100.0%

Table 4.3: The ratio of titles that fully overlap with words in the corresponding segment. The overlap is measured by unigram, bigram and full matches.

	ROUGE-1	ROUGE-L	ROUGE-W	Full Match
Hierarchical	**0.29**	**0.29**	**0.25**	**14**
Noisy-channel	0.14	0.13	0.12	3
Local	0.25	0.25	0.22	12
Keyword	0.22	0.22	0.21	9

Table 4.4: Comparison with reference titles.

Reference	4.82	4.88	4.71	4.82
Hierarchical	**2.91**	**3.53**	**3.18**	**3.00**
Noisy-channel	1.97	3.00	2.53	2.59
Local	2.68	3.44	3.03	2.65
Keyword	2.62	3.32	2.88	2.88

Table 4.5: Average ratings assigned to each system by each of the four judges.

keyword extraction exhibit a high degree of redundancy. In fact, 40% of the titles produced by this method are repeated more than once in the table-of-contents. In contrast, our method yields 11% of the titles as duplicates, as compared to 9% in the reference table-of-contents.[9] The full output of the Hierarchical model is shown in Appendix B.

Second, the fragments show that the two discriminative models — Local and Hierarchical — have a number of common titles. However, adding global dependencies to rerank titles generated by the local model changes 35.5% of the titles in the test set. The titles produced by the rest of the methods exhibit relatively little mutual similarity.

Finally, some titles in Figure 4-3 are extracted verbatim from the corresponding sections. Table 4.3 provides statistics on the extent of extraction in the reference and automatically produced tables-of-contents. As the first row of this table shows, most of the reference titles (64.5%) do not appear verbatim in the text. The ratio of titles appearing verbatim in the text varies significantly across the systems: while all the titles produced by the keyword method appear in full in the segment, only 57% of the titles produced by our method fully match a phrase in the text. The noisy-channel model has a lower ratio of extracted titles than our system. We attribute this difference to the fact that our model incorporates noun phrase dependencies which encourage selection at the phrase level.

[9]Titles such as *"Analysis"* and *"Chapter Outline"* are repeated multiple times in the text.

Comparison with reference tables-of-contents Table 4.4 shows the ROUGE scores for the four automatic methods. The hierarchical method consistently outperforms the three baselines according to all ROUGE metrics.

At the same time, these results also show that only a small ratio of the automatically generated titles are identical to the reference ones. In some cases, the machine-generated titles are very close in meaning to the reference, but are verbalized differently. Examples include pairs such as *("Minimum Spanning Trees", "Spanning Tree Problem")* and *("Wallace Tree", "Multiplication Circuit")*.[10] While measures like ROUGE can capture the similarity in the first pair, they cannot identify semantic proximity between the titles in the second pair. Therefore, we supplement the results of this experiment with a manual assessment of title quality as described below.

Human assessment To evaluate agreement between human judges, we analyze the relative rating of the systems for each example. We measure inter-evaluator agreement using the Kendall coefficient of concordance W [47], which expresses the degree of association among k rankings ($k \geq 2$) of n objects. This measure varies from 0 to 1, where 1 indicates perfect agreement. In our experiment, we compute Kendall's W coefficient among the four judges for each training instance and then average the results. The overall value of Kendall's W is 0.83.

The results of manual evaluation by the four judges are shown in Table 4.5. As expected, all the participants rated the reference titles the highest. Among the automatic systems, our method yields the best result. In general, the results obtained in this experiment are similar to the results obtained through automatic evaluation. In fact, the ranking produced by the first three judges exactly mirrors the ranking derived from ROUGE: Reference > Hierarchical > Local > Keyword > Noisy-channel.

[10] A Wallace Tree is a circuit that multiplies two integers.

4.8 Conclusions

We present a method for the automatic generation of a table-of-contents. The key strength of our method lies in its ability to track dependencies between generation decisions across different levels of the tree structure. The results of automatic evaluation and manual assessment confirm the benefits of joint tree learning: our system is consistently ranked higher than non-hierarchical baselines.

We also plan to expand our method for the task of slide generation. Like tables-of-contents, slide bullets are organized in a hierarchical fashion and are written in relatively short phrases. From the language viewpoint, however, slides exhibit more variability and complexity than a typical table-of-contents. To address this challenge, we will explore more powerful generation methods that take into account syntactic information.

Chapter 5

Conclusions and Future Work

In this thesis, we explored three decoding methods in the context of natural language generation and discourse processing. The first two methods have strong theoretical guarantees on their decoding performance, while the third method incorporates decoding into the learning procedure. We applied these methods to three tasks in the areas of natural language generation and discourse processing and found that stronger decoding strategies result in better system performance as compared to existing commonly used methods.

The first decoding strategy we examined was Integer Linear Programming (ILP), an optimization technique theoretically guaranteed to return the optimal solution. We assessed the performance of this exact decoding method by applying this model on the task of temporal graph induction. The results of our experiments showed that ILP produced more accurate temporal graphs than two greedy decoding algorithms.

Next, we investigated the use of a novel randomized algorithm, based on the idea of color-coding [2], for decoding in selection-and-ordering problems. Though we are not assured that the algorithm will find the optimal solution on any one run, we proved theoretical bounds on the number of iterations required to guarantee any desired likelihood of finding the correct solution. We evaluated this algorithm against ILP and beam search on the task of title generation. We demonstrated that the randomized algorithm was able to efficiently find the optimal solution, approximating the performance of ILP, while maintaining a low running time. We also showed that exact decoding methods were able to produce better titles than those produced by inexact algorithms.

Finally, we explored how the incorporation of the decoder into the learning algorithm affects system performance. Rather than following a traditional approach where the learner is unaware of the decoding procedure, we implemented a learning algorithm that is fully cognizant of the actual decoder. We tested this algorithm on the task of automatic generation of tables-of-contents. By decoding while learning, and accounting for intricate dependencies across various levels of table-of-contents hierarchy, our model was able to perform better than standard non-hierarchical models. Our increased performance was confirmed by both automatic and human evaluations.

An important direction for future research is the investigation of the type of constraints that these decoding algorithms can handle. For example, the randomized decoder can account only for local constraints under the current formulation, but ideally we would also want to consider global constraints. We believe that these constraints could be added to the randomized decoder without increasing its complexity.

We would also like to explore the applicability of these decoding methods in a wider context. In this thesis, we only analyzed the performance of the algorithms for tasks in the areas of natural language generation and discourse processing. We believe that other areas of NLP stand to benefit from the described methods.

Appendix A

Examples of Automatically Constructed TDAGs

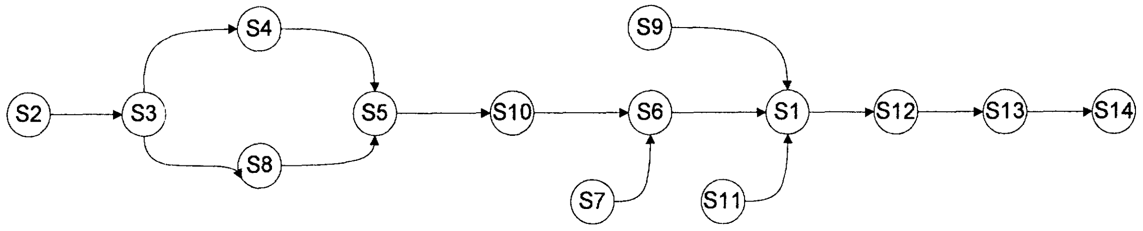


Figure A-1: The reference TDAG

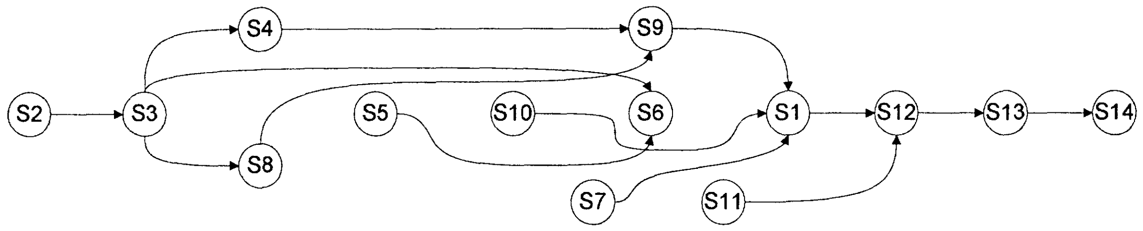


Figure A-2: ILP generated TDAG with an accuracy of 84.6%

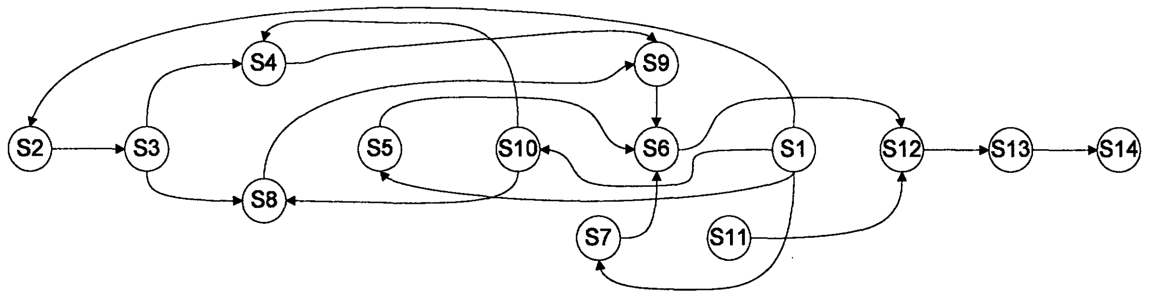


Figure A-3: BF generated TDAG with an accuracy of 71.4%; NRO produces the same graph for this example.

S1	A 32-year-old woman was admitted to the hospital because of left subcostal pain...
S2	The patient had been well until four years earlier,
S3	when she began to have progressive, constant left subcostal pain, with an intermittent increase in the temperature to 39.4°C, anorexia, and nausea. The episodes occurred approximately every six months and lasted for a week or two;
S4	they had recently begun to occur every four months.
S5	Three months before admission an evaluation elsewhere included an ultrasonographic examination, a computed tomographic (CT) scan of the abdomen...
S6	Because of worsening pain she came to this hospital.
S7	The patient was an unemployed child-care worker. She had a history of eczema and of asthma...
S8	She had lost 18 kg in weight during the preceding 18 months.
S9	Her only medications were an albuterol inhaler, which was used as needed,
S10	and an oral contraceptive, which she had taken during the month before admission.
S11	There was no history of jaundice, dark urine, light stools, intravenous drug abuse, hypertension, diabetes mellitus, tuberculosis, risk factors for infection with the human immunodeficiency virus, or a change in bowel habits. She did not smoke and drank little alcohol.
S12	The temperature was 36.5°C, the pulse was 68, and the respirations were 16...
S13	On examination the patient was slim and appeared well... An abdominal examination revealed a soft systolic bruit... and a neurologic examination was normal...
S14	A diagnostic procedure was performed.

Figure A-4: An example of a case summary.

Appendix B

Generated Tables of Contents

Table of Contents Generated by the Hierarchical Model

Grade School
- Combinational Circuits
  - Combinational Circuits
  - Combinational Circuits
  - Truth Table
  - Comparison Networks
  - Combinational Circuits
- Ripple Carry
  - Using an Element with a Parallel Prefix Circuit
  - Computing an Input and Output
- Wallace Tree
  - Lower
  - Inputs
- Clock Period
  - Periods
  - Replicated Bit of a Combinational Circuit Elements
  - Bit Numbers Takes Constant Time
  - Using a Linear Time Algorithm

Spanning Tree Problem
- Finding a Minimum Spanning Tree
- Elaborations of a Minimum Spanning Tree
  - Minimum Spanning Tree
  - Minimum Spanning Tree

Probabilistic Analysis and Randomized Algorithms
- Counting the Number
  - Situation in Which
  - Running Time

- Depth First Search

Dictionary Operations
- Direct Address Table
- Computer Memory
    - Worst Case Running Time
    - Searching for a Given Set
- Hash Functions
    - Real Numbers and a Hash Function
    - Creating a Set of Keys
    - Hash Functions and the Number of Keys
- Hash Table
    - Hash Function
    - Hash Function
    - Double Hashing
    - Hash Table with Load Factor

Polynomial Time
- Classes of a Directed Graph of NP Completeness
- Using the Technique of a Directed Graph Algorithms
- NP Completeness
- NP Completeness
    - Abstract Problems
    - Programs
    - Using a Decision Problem
- Algorithms for Which
    - Hamiltonian Cycle
    - Verification Algorithm
    - Complexity Class of Languages
- Why Theoretical Computer Scientists
    - Problems
    - NP Completeness
    - NP Completeness
- Formula Satisfiability Problems
    - Satisfiability Problem
    - CNF Satisfiability Problem
- NP Complete Problems

Reference Table of Contents

Bibliography

[1] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.

[2] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the Association for Computing Machinery (JACM)*, 42(4):844–856, July 1995.

[3] Roxana Angheluta, Rik De Busser, and Marie-Francine Moens. The use of topic segmentation for automatic summarization. In *Proceedings of the Association for Computational Linguistics (ACL)-2002 Workshop on Automatic Summarization*, 2002.

[4] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 277–283, 1995.

[5] Egon Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.

[6] Michele Banko, Vibhu O. Mittal, and Michael J. Witbrock. Headline generation based on statistical translation. In *Proceedings of the Association for Computational Linguistics*, pages 318–325, 2000.

[7] Regina Barzilay, Noemie Elhadad, and Kathleen McKeown. Inferring strategies for sentence ordering in multidocument news summarization. *Journal of Artificial Intelligence Research*, 17:35–55, 2002.

[8] Yves Bestgen and Wietske Vonk. The role of temporal segmentation markers in discourse processing. *Discourse Processes*, 19:385–406, 1995.

[9] Branimir Boguraev and Rie Kubota Ando. Timeml-compliant text analysis for temporal reasoning. In *Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 997–1003, 2005.

[10] Branimir Boguraev and Mary S. Neff. Discourse segmentation in aid of document summarization. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, pages 3004–3014, 2000.

[11] Wallace Chafe. The flow of thought and the flow of language. In Talmy Givon, editor, *Syntax and Semantics: Discourse and Syntax*, volume 12, pages 159–182. Academic Press, 1979.

[12] Eugene Charniak. A maximum-entropy-inspired parser. In *Proceedings of the Human Language Technology (HLT)-North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 132–139, 2000.

[13] William Cohen, Robert Schapire, and Yoram Singer. Learning to order things. *Journal of Artificial Intelligence*, 10:243–270, 1999.

[14] Michael Collins. Head-driven statistical models for natural language parsing, 1999.

[15] Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the ACL*, 2004.

[16] Carlo Combi and Yuval Shahar. Temporal reasoning and temporal data maintenance in medicine: Issues and challenges. *Computers in Biology and Medicine*, 27(5):353–368, 1997.

[17] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Intoduction to Algorithms.* The MIT Press, 1992.

[18] Simon Corston-Oliver, Michael Gamon, Eric Ringger, and Robert Moore. An overview of amalgam: A machine-learned generation module. In *Proceedings of International Natural Language Generation (INLG)*, pages 33–40, 2002.

[19] Brooke Cowan, Ivona Kucerova, and Michael Collins. A discriminative model for tree-to-tree translation. In *Proceedings of the Empirical Methods in Natural Language Processing Conference (EMNLP)*, pages 232–241, 2006.

[20] Hal Daumé and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 169–176, 2005.

[21] Bonnie Dorr, David Zajic, and Richard Schwartz. Hedge trimmer: a parse-and-trim approach to headline generation. In *Proceedings of the Human Language Technology (HLT)-North American Chapter of the Association for Computational Linguistics (NAACL) Workshop on Text summarization*, pages 1–8, 2003.

[22] Rod G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness II: On completeness for $W[1]$. *Theoretical Computer Science*, 141(1–2):109–131, 1995.

[23] David R. Dowty. The effects of aspectual class on the temporal structure of discourse: Semantics or Pragmatics? *Linguistics and Philosophy*, 9:37–61, 1986.

[24] Jason Eisner and Roy W. Tromble. Local search with very large-scale neighborhoods for optimal permutations in machine translation. In *Proceedings of the Human Language Technology (HLT)-North American Chapter of the Association for Computational Linguistics (NAACL) Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*, 2006.

[25] Noemie Elhadad and Kathleen R. McKeown. Towards generating patient specific summaries of medical articles. In *Proceedings of North American Chapter of the*

Association for Computational Linguistics (NAACL) Workshop on Automatic Summarization, pages 31–39, 2001.

[26] Richard Fikes, Jessica Jenkins, and Gleb Frank. A system architecture and component library for hybrid reasoning. Technical report, Stanford University, 2003.

[27] Jenny Rose Finkel, Trond Grenager, and Christopher D. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the Association for Computational Linguistics (ACL)*, 2005.

[28] Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. Fast decoding and optimal decoding for machine translation. In *Proceedings of the European Chapter of the Association for Computational Linguistics (EACL)/ Association for Computational Linguistics (ACL)*, pages 228–235, 2001.

[29] Marti Hearst. Multi-paragraph segmentation of expository text. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 9–16, 1994.

[30] Marti Hearst. Multi-paragraph segmentation of expository text. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 9–16, 1994.

[31] Chung Hee Hwang and Lenhart K. Schubert. Tense trees as the "fine structure" of discourse. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 232–240, 1992.

[32] Fred Jelinek. A fast sequential decoding algorithm using a stack. *IBM Research Journal of Research and Development*, 1969.

[33] Rong Jin and Alexander G. Hauptmann. Automatic title generation for spoken broadcast news. In *Proceedings of the Human Language Technology (HLT)*, pages 1–3, 2001.

[34] Philipp Koehn. Pharaoh: A beam search decoder for phrase-based statistical machine translation models. In *Proceedings of Association for Machine Translation in the Americas (AMTA)*, pages 115–124, 2004.

[35] Mirella Lapata and Alex Lascarides. Inferring sentence-internal temporal relations. In *Proceedings of the Human Language Technology (HLT)-North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 153–160, 2004.

[36] Alex Lascarides and Nicholas Asher. Temporal interpretation, discourse relations, and commonsense entailment. *Linguistics and Philosophy*, 16:437–493, 1993.

[37] Alex Lascarides and John Oberlander. Temporal connectives in a discourse context. In *Proceeding of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 260–268, 1993.

[38] Inderjeet Mani, Barry Schiffman, and Jianping Zhang. Inferring temporal ordering of events in news. In *Proceedings of the Human Language Technology (HLT)-North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 55–57, 2003.

[39] Mark Moens and Mark J. Steedman. Temporal ontology in natural language. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 1–7, 1987.

[40] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, New York, NY, 1995.

[41] Rebecca J. Passonneau. A computational model of the semantics of tense and aspect. *Computational Linguistics*, 14(2):44–60, 1988.

[42] James Pustejovsky, Patrick Hanks, Roser Sauri, Andrew See, David Day, Lissa Ferro, Robert Gaizauskas, Marcia Lazo, Andrea Setzer, and Beth Sundheim. The timebank corpus. *Corpus Linguistics*, pages 647–656, 2003.

[43] Hans Reichenbach. *Elements of Symbolic Logic.* Macmillan, New York, NY, 1947.

[44] Dan Roth and Wen-tau Yih. A linear programming formulation for global inference in natural language tasks. In *Proceedings of the Conference on Computational Natural Language Learning*, pages 1–8, 2004.

[45] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[46] Robert E. Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.

[47] Sidney Siegel. *Nonparametric Statistics for the Behavioral Sciences.* McGraw-Hill, New York, NY, 1956.

[48] Simone Teufel and Marc Moens. Summarizing scientific articles: Experiments with relevance and rhetorical status. *Computational Linguistics*, 28(4):409–445, 2002.

[49] Nina Wacholder, David K. Evans, and Judith Klavans. Automatic identification and organization of index terms for interactive browsing. In *Proceedings of Joint Conference on Digital Libraries (JCDL)*, pages 126–134, 2001.

[50] Ruichao Wang, John Dunnion, and Joe Carthy. Machine learning approach to augmenting news headline generation. In *Proceedings of the International Joint Conference on Natural Language Processing (IJCNLP)*, 2005.

[51] Bonnie L. Webber. The interpretation of tense in discourse. In *Proceedings of the Association for Computational Linguistics (ACL)*, pages 147–154, 1987.

[52] Bonnie L. Webber. Tense as discourse anaphor. *Computational Linguistics*, 14(2):61–73, 1988.

[53] George Wilson, Inderjeet Mani, Beth Sundheim, and Lisa Ferro. A multilingual approach to annotating and extracting temporal information. In *Proceedings of the Association for Computational Linguistics (ACL) 2001 Workshop on Temporal and Spatial Information Processing*, pages 81–87, 2001.

[54] Yaakov Yaari. Segmentation of expository texts by hierarchical agglomerative clustering. In *Proceedings of the Recent Advances in Natural Language Processing (RANLP)*, pages 59–65, 1997.

[55] Li Zhou, Carol Friedman, Simon Parsons, and George Hripcsak. System architecture for temporal information extraction, representation and reasoning in clinical narrative reports. In *Proceedings of American Medical Informatics Association (AMIA)*, pages 869–873, 2005.