Tag Insertion Complexity

Stuart Yeates, Ian H. Witten and David Bainbridge
Department of Computer Science,
University of Waikato, Hamilton, New Zealand
{s.yeates, i.witten, d.bainbridge}@cs.waikato.ac.nz

1 Introduction

Lewis Carroll's poem *Jabberwocky* opens with these lines:

> 'Twas brillig, and the slithy toves
> Did gyre and gimble in the wabe:
> All mimsy were the borogoves,
> And the mome raths outgrabe.

Despite the fact that of the words having four or more letters only two appear in the dictionary, fluent speakers of English have no difficulty accepting the text as English or, given a little preparation, reading it aloud. What is it about this text that makes it seem so natural while containing so few English words? The outer structure helps, the line breaks, the capital letters at the start of each, the rhyme at the end. But far more can be inferred:

> The toves were obviously gyring as well as gimbling; the raths, while engaged in outgrabing (or perhaps outgribing), were in some way related to mome, or had the quality of momeness, and so on. In fact, the markers enable you to place every nonsense word but two in its proper syntactic category. [1]

These inferences exploit syntactic information that native speakers pick up unconsciously from the original poem, based on lexical features such as the framework of function words and word endings. This information can be encoded by marking up the text:

> 'Twas brillig, and the <a>slithy <n>toves</n>
> Did <v>gyre</v> and <v>gimble</v> in the <n>wabe</n>:
> All <a>mimsy were the <n>borogoves</n>,
> And the mome <n>raths</n> <v>outgrabe</v>.

This paper is about inferring markup information, a generalization of part-of-speech-tagging. We use compression models based on a marked-up training corpus and apply them to fresh, unmarked, text. In effect, this technique builds filters that extract information from text in a way that is generalized because it depends on training text rather than preprogrammed heuristics.

As illustrated above, we use SGML tags to represent the extracted information. However, we work in a more controlled textual environment: we use bibliographic text rather than plain English and mark up entities such as author, date, and titles rather than syntactic parts of speech. Such entities are generically called "metadata"—data about data—and form an important component of the information present in a bibliography.

The aim of our work is to automatically enhance bibliographies with metadata tags, based on a training corpus of annotated bibliography entries. In previous work, we have applied

similar techniques to the word segmentation problem [6], acronym detection [10] and generic entity extraction [8] approaching the heuristics informally, with little or no regard for the size of the search spaces involved. Some preliminary studies of the complexity of tag insertion were reported at a workshop [11]; the present paper provides a deeper and more complete analysis.

The basic methodology is to form compression models from the pre-tagged training text, and to seek a tagging of the test text that allows it to be compressed into the smallest number of bits using the models. Different models are used for each tag type: thus there will be separate author, date, and title models. We use the PPMD [9] character-based compression method, that is, the well-known PPM technique in conjunction with Howard's escape method D [4]. In order to insert tags into unmarked text, we employ a Viterbi search. Tag insertion is computationally expensive because the search inevitably involves a large space.

The most important determiner of success is the size of the training corpus and how representative of the test text. It is difficult to quantify these factors. At one end of the spectrum, the bibliographic references that the system is tested on may have already been present in the training corpus, in which case they will have been incorporated into the models. At the other, it is possible (though unlikely) that the training corpus and the test set have no words in common. Between these extremes lie many intermediate possibilities.

The intent of this paper is not to evaluate the overall success of the markup operation. Instead, we focus on techniques for reducing the size of the search space. Simple properties of the SGML standard [3], such as nesting, can be used to reduce the search. In this paper we develop a succession of provably correct and heuristic techniques that prune it further: Viterbi Search, One-Tag-at-a-Time, Automatic Tokenization and Maximum Lookahead. The improvement is significant, reducing search space size from $\mathcal{O}(t^{tn})$ to $\mathcal{O}(tn)$ where t is the size of the tag set and n is the length of the test text, in characters.

Some of the techniques we describe have no effect at all on the result of the markup operation. For ones that do, the price paid can be assessed in terms of the standard measures of recall and precision, relative to a correct markup of the test text [11]. Recall is the proportion of tags in the known correct markup that were also in the markup produced by the system. Precision is the proportion of the tags produced by the system that were also in the the known correct markup.

The next section discusses the testbed for our techniques, a set of bibliographic data, and indicates how it was generated. Next we formalize the problem as one of tag insertion, and present theoretical bounds on the search space when operating under various assumptions. Section 5 gives experimental results that confirm the theoretical analysis and evaluate the effect of the heuristics we introduce. We conclude with a summary of our findings.

2 Marking up bibliographies

Figure 1a shows an extract from a bibliography, and Figure 1b shows the same section in marked-up form. We have chosen to identify seven types of metadata: last names, first names, titles, dates, years, pages and numbers. We have adopted the SGML convention for markup. Tags are expressed in angle brackets. Each tagged item is terminated with a corresponding closing tag, beginning with the "/" character. Tagged items may be nested: for example, the "year" markup is nested within "date".

Some arbitrary decisions must be made when tagging. For example, we have chosen to include terminating punctuation—commas following a name, periods following an initial or a title—within the tagged items. We have chosen to mark initials as first names, and to mark a sequence of consecutive first names together. Dates are marked without the enclosing parentheses. Our system is fairly impervious to such decisions, provided they are made

(a)

```
[10] Stein, J. Constraints in the association-object data model. TR 87-011\n
(Aug. 1987). \n
[11] Stenkiste, P. A., and Henessy, J. L. A simple interprocedural register\n
allocation algorithm and its effectiveness in LISP. ACM Trans. on\n
Programming Languages and Sys. (Jan. 1989).\n
[12] Stockmeyer, L. J. The polynomial-time hierarchy. Theoretical CS 3\n
(1977), 1. (Located in packet #1, UWisc CS-830, Eric Bach).\n
```

(b)

```
[10] <last>Stein,</last> <first>J.</first> <title>Constraints in the association-object data
model.</title> TR 87-011\n
(<date>Aug. <year>1987</year></date>).\n
[11] <last>Stenkiste,</last> <first>P. A.,</first> and <last>Henessy,</last> <first>J. L.-
</first> <title>A simple interprocedural register \n
allocation algorithm and its effectiveness in LISP.</title> ACM Trans. on\n
Programming Languages and Sys. (<date>Jan. <year>1989</year></date>).\n
[12] <last>Stockmeyer,</last> <first>L. J.</first> <title>The polynomial-time hierarchy.-
</title> Theoretical CS 3\n
(<date><year>1977</year></date>), <pages>1.</pages> (Located in packet #1,UWisc CS-
830, Eric Bach).\n
```

(c)

```
[Pin, 1995] Pin, J.-E. (1995). A negative answer to a\n
question of Wilke on varieties of !-languages. Information\n
Processing Letters, 56(4):197–200.
```

Figure 1: Bibliographic markup. (a) A section of bibliography. (b) The same section marked-up. (c) A bibliography entry that presents challenges.

consistently.

To ensure consistency, and to generate a substantial volume of tagged text, Figure 1a and b are both generated using a LaTeX/BibTeX system from a large body of references. We wanted to simulate all features of real reference lists, including hyphenation, line breaks, and page breaks—both the white space that they generate, and "noise" features such as pagination and page numbers. Because of this, the process is more complex than would first seem to be necessary.

In order to generate realistic bibliographic text, references were grouped into files containing about 25 items each. These files were LaTeX'd and PostScript was generated, creating page images of a corresponding typeset reference list. Then, text was extracted from the PostScript file using a standard text extraction utility. This exactly simulated the effect of a computer-generated list of references in a research paper. Eight different bibliographic style files were used, and LaTeX was used to lay out pages of references for four different kinds of publication style—book, journal article, proceedings article and report. The result was that 32 different styles were available. Two hundred files were processed using each of these different styles. With about 25 references per file, this generated approximately 160,000 references.

For Figure 1b, the LaTeX macros were modified to insert tags during the process of laying out the references. The result was then correlated with the information generated above for untagged references, in order to ensure that line breaks occurred at "natural" places even though the text was tagged. Thus lines end at the same place in Figure 1b as they do in Figure 1a; to emphasize this they are marked explicitly.

Figure 1c shows a single bibliographic entry before markup, such as might appear in test text. This example presents a number of challenging problems. First, the bibliography key that appears in square brackets, "[Pin, 1995]", contains the author surname and year, rather than a plain numeric label. Punctuation appears in an unusual place: within the title of the document it occurs as a letter in the word "!-languages." Moreover, the document's title

contains the name of another author ("Wilke"), which means that this word may compress better using the author model than the title model.

3 Tag insertion and its search space

Briefly stated, the general tag insertion problem is this:

> Given a sequence of characters $C_0 \ldots C_{n-1}$ and a set of tags $T_0 \ldots T_{t-1}$, how should the tags be inserted into the sequence so that the tags reveal the metadata implicit in it?

Four natural assumptions, built into the SGML standard, allow the tags to be interpreted as defining particular kinds of data items in the sequence. They must be satisfied for a tagging to be syntactically correct.

1. Each tag in the tag set has a begin tag symbol T_i and an end tag symbol $T_{/i}$.
2. In the text, tags must be balanced—there must be the same number of T_i as $T_{/i}$ tags.
3. In any prefix of the tagged text, the number of end tags $T_{/i}$ cannot exceed the number of begin tags T_i.
4. Within a tag pair any nested tags must be balanced.

As explained above, we evaluate a particular tagged version of the text by calculating its entropy with respect to a set of compression models. One compression model is formed for every tag type, from pre-tagged training data. Each putative tagging of the sequence is evaluated by calculating its entropy, where the contents of each tag are compressed by the appropriate model. A fundamental assumption is that, of all possible taggings, the one with minimum entropy best reveals the metadata implicit in the sequence, relative to the corpus on which the compression models were trained. We use PPMD compression models.

Three further assumptions can be made that greatly reduce the number of possible ways to insert tags into the sequence.

5. T_i tags and $T_{/i}$ tags must alternate.
6. At least one character must occur between a T_i and a $T_{/i}$.
7. Tags have no attributes.

Assumption 5 forbids recursive tags: items cannot contain the same kind of item as a component. However, nested tagging is permitted. Assumption 6 forbids null tags. Assumption 7 restricts the scope of the problem by making each tag atomic. These three assumptions are easily encoded in an SGML Document Type Definition.

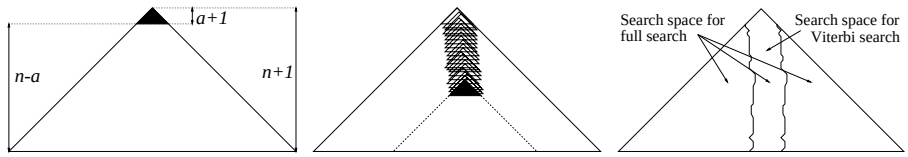
4 The search space

In the general tag insertion problem, there are

$$\sum_{r=0}^{t} \frac{t!}{(t-r)!}$$

ways of selecting zero or more tags from a vocabulary of t tags. Suppose the source text contains n characters and tags may be inserted between any two characters, before the first and after the last character. Then there are approximately

$$\left[\sum_{r=0}^{t} \frac{t!}{(t-r)!}\right]^{n} \qquad \mathcal{O}(t^{tn}) \qquad (1)$$



(a) (b) (c)

Figure 2: Viterbi search of a large search space, with the active part of the search space at each section marked in black.

possible encodings of the sequence with a tag set of size t.

In terms of a search space, the problem is represented as a tree with a branching factor of $\sum_{r=0}^{t} \frac{t!}{(t-r)!}$ and a depth of n. This is a very large space, especially considering that real world tag sets (HTML, XML *etc.*) commonly have $t \geq 50$ and that real world documents commonly have $n \geq 5,000,000$. Indeed, searching spaces of this order is intractable.

We have not been able to calculate an exact equation for this and for our purposes an exact equation is not necessary—we are only interested in the relative magnitude of these intractable searching spaces.

4.1 Viterbi Search

Viterbi search [7, 5] is a long-standing algorithm for the incremental processing of linear data streams. It provides a non-heuristic entropy-minimizing search, which is illustrated in Figure 2a–c. Only a small portion (coloured black) is searched initially (a). The lowest entropy leaf is found and the root node is pruned to leave only the branch connected to the lowest entropy leaf; then, all leaves on that branch are expanded. At each step the root of the active search space is pruned, leaving only the branch that is connected to the lowest entropy leaf, and all unpruned leaves are expanded (b). The dashed lines indicate the portions of the search space pruned by this step. When the search reaches the end of the sequence, only a subset (c) of the space has been searched. The application of Viterbi search to tag insertion means rather than processing the complete sequence, incrementally processing sections of characters.

Pruning the search space is possible because of a proof outlined in [7], which gives bounds on how much context need be taken into account when considering whether or not to insert a tag. When inserting a tag with a maximum length of l_{max} using a model of maximum order o_{max}, the maximum lookahead is $a = l_{max} + o_{max}$.

This reduces the search space from Equation 1 to:

$$(n-a)\left[\sum_{r=0}^{t} \frac{t!}{(t-r)!}\right]^{a+1} \qquad \mathcal{O}(t^{ta}) \qquad (2)$$

The reason for the factor $n - a$ is shown graphically in Figure 2a—it is the number of increments in which the incremental search space is searched. This reduction makes the tag insertion problem tractable, for sufficiently low order PPMD models (typically 1–5 characters), sufficiently short maximum tag lengths (typically in the range of 5–15) and sufficiently small tag sets (typically 1–3).

4.2 One Tag at a Time Search (OTT)

Further reduction in the search space can be achieved using heuristic assumptions. One heuristic, which we call One Tag at a Time (OTT), is to perform the search in multiple passes, once for each pair of tags T_i and $T_{/i}$, rather than simultaneously inserting all the tags.

This further reduces the search space from Equation 2 to:

$$\sum_{i=1}^{t}(n-a_i+c)3^{a_i+1} \qquad \mathcal{O}(t3^{max(a_1,\cdots,a_t)}) \qquad (3)$$

by reducing the search space to a set of t ternary trees, one per tag, which are explored sequentially using the Viterbi algorithm. c is the number of tags inserted by previous passes and is bounded by $tn^2 \geq c \geq 0$, but can be expected to be $tn \geq c \geq 0$ for reasonable tag sets and documents. a_i is the maximum lookahead for tag i and is the sum of the maximum length for that tag and the maximum order of the model.

This heuristic may introduce errors into the markup (lower recall and/or precision) where two or more tags contain similar text, such as journal volume numbers, journal part numbers and journal page numbers.

4.3 Adjacency of tags

A further simplification in our implemented system was to assume that no $T_{/i}$ was immediately followed by a T_i. While this goes beyond the SGML assumptions given above, if it holds true in the markup, it reduces the search space from Equation 3 to:

$$\sum_{i=1}^{t}(n-a_i+c)2^{a_i+1} \qquad \mathcal{O}(t2^{max(a_1,\cdots,a_t)}) \qquad (4)$$

We intend to remove this assumption in further work because we feel that this assumption may not hold generally.

4.4 Best First Search

In the previous sections we assumed that the Viterbi search completely explored each sub-tree to the leaves before selecting the lowest entropy leaf and pruning based on it. If we perform a depth-first search expanding the lowest entropy nodes first, it is possible that some non-leaf nodes cannot conceivably lead to the lowest entropy leaf because we have already seen a leaf node with lower entropy. Such nodes do not need not be expanded, greatly reducing the search space. This is effectively a best first search.

The reduction in search space is entirely dependent on how well-trained the PPMD model is. Empirically it is found to be substantial. The effect is to reduce the search space from Equation 4 to:

$$\sum_{i=1}^{t}(n-a_i+c)2^{p(a_i+1)} \qquad \mathcal{O}(t2^{max(a_1,\cdots,a_t)}) \qquad (5)$$

where p is bounded by $1 \geq p \geq 0$.

4.5 Automatic Tokenization

Automatic Tokenization (AT) is a heuristic based on the observation that tags often occur in certain places. Names, titles and dates do not begin in the middle of words but on word boundaries. By noticing during training that no tag occurs between a pair of lowercase letters or between a pair of digits, as is the case for all the examples mentioned in this paper, the search can be pruned. Using Unicode defined categories [2], we classify characters as letter (with the partial subsets uppercase-letter and lowercase-letter), digit, punctuation and white space.

The search-space reduction given by AT is difficult to model exactly. Assume that tags are only inserted either side of non-alphanumeric characters, and that words average five characters with one white space character and one punctuation character between each word. Then the depth of the search tree is reduced by a factor of $\frac{3}{7}$, and the search space becomes:

$$\sum_{i=1}^{t} (n - a_i + c) 2^{\frac{3}{7} p(a_i+1)} \qquad \mathcal{O}(t 2^{max(a_1, \cdots, a_t)}) \qquad (6)$$

AT has interesting implications for initials and names. For example, the string `John McMahon and Francis J. Smith` has approximately the same sized search space as the string `J. McMahon and F. J. Smith`, because while the extra characters could not have had tags inserted between them, the added "." could have a tag inserted before it. More generally, on long terms AT works as well as or better than abbreviations of those terms, since long terms are effectively abbreviated by the tokenizer.

Because there are relatively few contexts in which to see a tag, the chances of encountering all correct tag contexts, when training on enough data to build effective PPMD models, is extremely high. While it is possible to construct artificial situations in which this heuristic fails, naturally occurring examples are unlikely.

4.6 Maximum Lookahead

The heuristics developed so far are extremely sensitive to changes in a, which is closely related to the observed maximum length of a tag, l_{max}. Because the data with which we are working includes a few extremely long tags due to errors in the bibliographies, we found it beneficial to replace the calculation for l_{max} with one based on the average tag length $l'_{max} = average(l) + 3\sigma$, where σ is the standard deviation. This reduces the search space from Equation 6 to:

$$\sum_{i=1}^{t} (n - a'_i + c) 2^{\frac{3}{7} p(a'_i+1)} \qquad \mathcal{O}(tn) \qquad (7)$$

Here, a'_i is the new value of a_i based on a calculated l'_{max}.

5 Experimental Search Space Sizes

The different variants of the search algorithm described above have been evaluated on a dataset of bibliographies. As described in Section 2, we had available as training and test data a corpus of BibTeX databases that had been passed through a system that inserted SGML tags as well as doing standard bibliographic processing and layout.

For training we used 2000 bibliography files, each containing approximately 25 entries, with a total of 49,000 references. They were marked up in the manner of Figure 1b, with tags for titles (article or book titles), dates, years, first names (in general, these were initials), last names, page ranges, and numbers (in general, numbers of technical reports). Separate models were generated for each of these seven data types.

PPMD models were used throughout. We experimented with models of different orders, and found that second-order models worked best, probably because the amount of training data for some of the models was not large. Table 1 indicates the size of the training data that was used for each model.

The models were tested on 10 randomly selected files that did not form part of the training data, again containing about 25 bibliography entries each; the number of tags involved is given

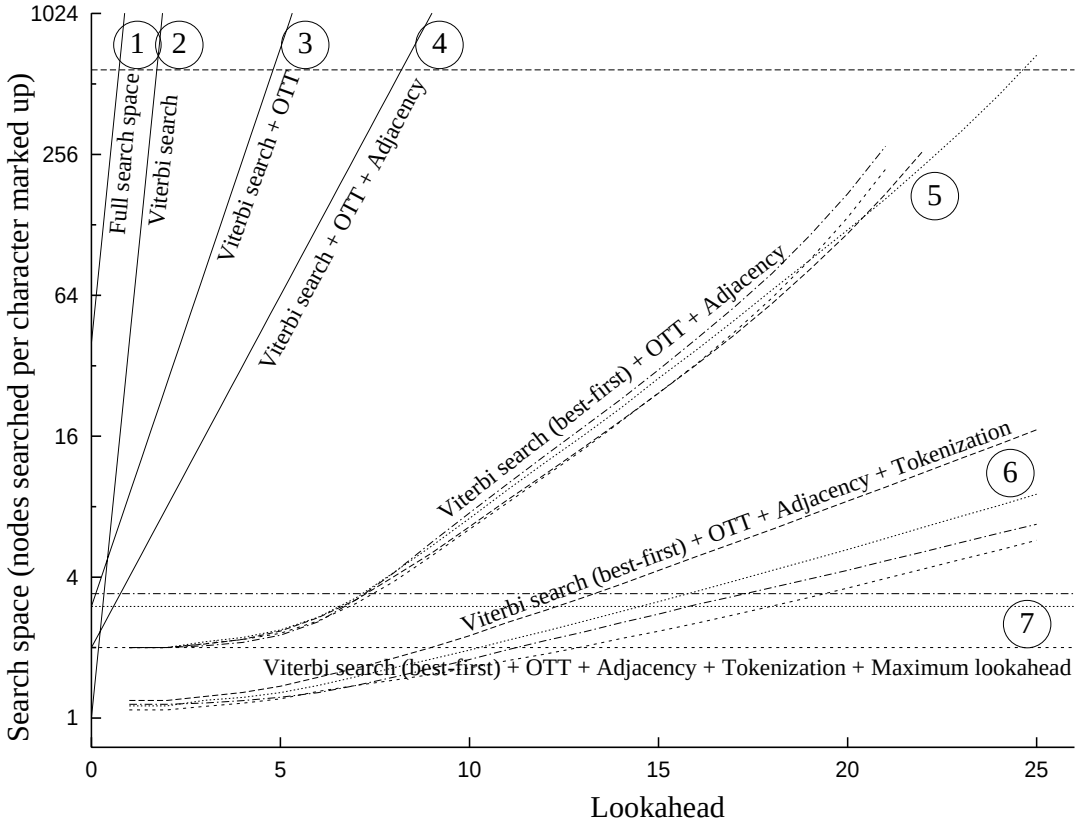


Figure 3: Search space size for each of the seven equations. Equations 1-4, with solid lines, are calculated values. Equations 5-7, with broken lines, are experimentally determined values. The broken horizontal line at the top of the graph is the title tag from Equation 7.

in Table 1. All our experiments are based on the identification of single data types, one tag at a time.

Figure 3 summarizes the growth in search space size, as a function of the amount of lookahead that is permitted, for the various different improvements described in Section 3.

1. **Full search** The full search space (Equation 1) is so large that it is hardly visible, at the top left of the graph.

2. **Viterbi search** The Viterbi method (Equation 2), yields a great improvement, although it is right up against the left side of the graph.

3. **One Tag at a Time (OTT)** The OTT procedure produces further improvement (Equation 3). A heuristic: it may introduce markup errors.

4. **Adjacency of tags** The adjacent tags assumption (Equation 4) produces a slight improvement, although not as significant as either of the previous two. If the assumption holds for a given type of markup, no errors are introduced. If it does not, markup is likely to fail, potentially catastrophically.

5. **Best-first search** The improvement generated by the use of best-first search cannot be evaluated theoretically, but depends on the training data. Equation 5 gives an upper bound. However, Figure 3 is obtained experimentally using the training and test data

	Tag	first name	last name	number	title	pages	year	date	total
Training	# of tags	86,000	87,000	18,000	45,000	39,000	45,000	45,000	365,000
	Tag length	4.0 ± 2.2	7.5 ± 2.5	1.3 ± 1.0	63.3 ± 25.9	7.2 ± 1.9	4.0 ± 0.4	7.1 ± 3.4	
	# of chars	345K	653K	23K	2.8M	283K	180K	319K	4.6M
Test	# of tags	380	382	84	215	181	204	204	1650
Search	a	13	17	7	143/35	15	8	19	
space	Size	9500	32000	5200	2100000	12000	5600	15000	2200000

Table 1: Training and test data, and calculations of search space size. The two search space sizes shown for title are before and after the lookahead pruning described in Section 4.6.

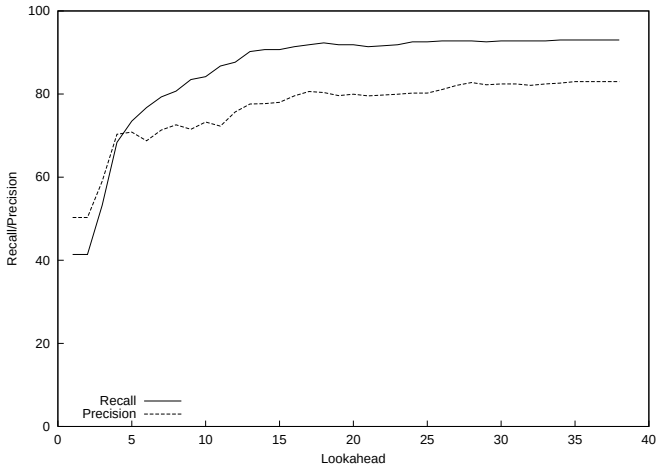


Figure 4: Recall and precision of the title tag as a function of the amount of lookahead.

described above. As can be seen, different tag types (Table 1) yield different improvements, although the general trends are very much the same. The differences are due to differences in effectiveness of the respective PPM models.

6. **Automatic tokenization** Automatic tokenization (Equation 6) provides a further dramatic reduction in the size of the search space as shown in Figure fig:searchspacesize. The variation between different tag types has increased, because the tokenizer varies radically from one tag type to another. For example, first names contain only letters, whitespace and simple punctuation ('.' and '.') while titles also contain digits, arithmetic symbols and matched punctuation ('[' and ']' *etc.*).

7. **Maximum lookahead** Some tags, such as the title tag, are long, potentially unbounded, constructs. The effect of introducing an artificial maximum length on the size of tags is easy to visualize in Figure 3: beyond the chosen maximum lookahead length, the graph is flat.

The maximum lookahead heuristic needs to be evaluated in terms of its effect on the result of the markup operation. For the title tag, which is by far the longest one, Figure 4 shows how recall and precision vary as the amount of lookahead is altered. As Table 1 shows, the average title length is 63.3 However, the longest title is in fact 3749 letters long. Thus a lookahead of around 3750 characters would be necessary to guarantee correct treatment of all titles. However, the median length is much shorter, and as the Figure 4 shows, recall and precision approach a virtually steady state with a lookahead of about 25 characters.

6 Conclusion

We have presented several properties of SGML and used them to find the number of possible ways to insert a set of tags into a sequence of characters and thus the search space when searching for an optimal tagging of the characters with the tags. We have used Viterbi search, OTT, adjacency of tags, best-first search, automatic tokenization and maximum lookahead to progressively reduce this search space from $\mathcal{O}(t^{tn})$ to the tractable size of $\mathcal{O}(tn)$ for interesting problems. For example, the file from which Figure 1 was taken contains 25 bibliography entries and 3658 characters; marking it up with the seven tag types investigated in this paper reduces the overall search space size from $\approx 10^{11311}$ to approximately $\approx 10^6$.

We have also discussed which techniques introduce errors into the system and under what circumstances. Finally we have presented experimental results for those techniques which are difficult to model mathematically, using a bibliographic example. An implementation, written in Java, of the ideas presented in the paper is available at
http://www.cs.waikato.ac.nz/~say1.

References

[1] Robert Claiborne. *The Life and Times of the English Language—the history of our marvellous native tongue.* Bloomsbury, 1983.

[2] The Unicode Consortium. *The Unicode Standard—Worldwide Character Encoding.* Addison-Wesley, 1992.

[3] Charles F. Goldfarb. *The SGML Handbook.* Oxford, 1990.

[4] Paul Glor Howard. *The Design and Analysis of Efficient Lossless Data Compression Systems.* PhD thesis, Department of Computer Science, Brown University, June 1993.

[5] M. S. Ryan and G. R. Nudd. The Viterbi algorithm. Warwick Research Report RR238, Dept. Computer Science, University of Warwick, Coventry, England, 12 February 1993.

[6] William J. Teahan, Yingying Wen, Roger McNab, and Ian H. Witten. A compression-based algorithm for Chinese word segmentation. *Computational Linguistics*, 26(3):375–393, September 2000.

[7] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.

[8] Ian H. Witten, Zane Bray, Malika Mahoui, and William J. Teahan. Using language models for generic entity extraction. In *Proc ICML Workshop on Text Mining*, 1999.

[9] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes — Compressing and Indexing Documents and Images.* Morgan Kaufmann, 2nd edition, 1999.

[10] Stuart Yeates, David Bainbridge, and Ian H. Witten. Using compression to identify acronyms in text. In James A. Storer and Martin Cohn, editors, *Proceedings of the Data Compression Conference (DCC)*, page 582, Snowbird, Utah, 28–30 March 2000 2000. IEEE Computer Society, IEEE. Full version in Working Paper 00/01, Computer Science, University of Waikato, Hamilton, NZ, January 2000.

[11] Stuart Yeates and Ian H. Witten. On tag insertion and its complexity. In Ah-Hwee Tan and Philip Yu, editors, *Proceedings of International Workshop on Text and Web Mining PRICAI 2000*, pages 52–63, Melbourne, Australia, August 28th 2000.