# Finding Chunks In Symbolic Time Series

Jason L. Hutchens, Michael D. Alder

***Abstract*— The instantaneous entropy of a predictive model is generally recognised as a good indicator of where the boundary between two words lies in natural language text. In the current paper, we present various methods for finding *chunks* in symbolic time series using measures from information theory, and we show how these chunks may be used to *UpWrite* the series to a more abstract representation, which brings distant symbols closer together to alleviate the long-distance dependency problem faced by certain classes of predictive models, such as Hidden Markov Models. We show how these chunking methods may be used to segment natural language text, and we speculate on applications in adaptive data compression.**

*Keywords*—**Data compression, entropic chunking, text segmentation, grammatical inference.**

## I. Introduction

We consider the problem of finding *chunks*—more or less well-defined strings of symbols—in symbolic time series; that is, finding sequences of symbols which constitute intermediate symbols of a rewrite stochastic dictionary grammar. This may be regarded as a form of stochastic grammatical inference for a class of models which would include Hidden Markov Models as a special case. An example is finding English words in a sequence of ASCII characters. If we have a known separator symbol such as whitespace then this, of course, reduces to triviality, but we have to consider the case where no such symbol has been previously identified, or the case where no such symbol exists.

More formally, let $\mathcal{A}$ denote a finite set called the *alphabet*, containing at least two distinct elements called the *symbols*. The term *data* will refer to some sequence $s : \mathbb{Z}^+ \longrightarrow \mathcal{A}$, also referred to as a *stream*, or a *symbolic time series*, or some finite sample of such a sequence which is long compared with the alphabet size. We use $|\mathcal{A}|$ to denote the cardinality of the alphabet.

A *predictive model*, $\mathcal{M}$, is an algorithm which, when presented with a sequence of symbols $s$, gives a prediction about the next symbol $w$ in the form of a probability distribution over the alphabet $\mathcal{A}$, written $P(w|\mathcal{M}, s)$. We require that the only information available to the model is $s$, the sequence of symbols seen thus far. There is no requirement that the probability distribution be smoothed; it is allowable for some symbols to be assigned a zero probability. Clearly an $n^{th}$–order markov model, for $n > 0$, satisfies these requirements.

We obtain the Maximum Likelihood markov model of order $n$ by gathering statistics about equivalence classes containing $n$ symbols, as in equation 1, where $C(x)$ is a counting function which returns the number of occurrences of $x$ in the data.

$$P(w_i|\mathcal{M}, w_1, w_2, \ldots, w_{i-1}) \approx \frac{C(w_{i-n}, \ldots, w_{i-1}, w_i)}{C(w_{i-n}, \ldots, w_{i-1})} \quad (1)$$

The information of a symbol with respect to a predictive model and a history of symbols is defined in equation 2. We may refer to this as the *information supplied to the model by the symbol following s*. It specifies the minimum number of bits required to describe the symbol in an unambiguous way, and may be understood informally to represent the "surprise" the model receives upon discovering what the next symbol in the sequence actually is.

$$I(w|\mathcal{M}, s) = -\log_2 P(w|\mathcal{M}, s) \quad (2)$$

The expected value of $I(w|\mathcal{M}, s)$, which we shall call the *instantaneous entropy of the model after s*, is then given by equation 3. We intuitively think of $H(\mathcal{M}, s)$ as the "uncertainty" the model has about the next symbol in the sequence.

$$H(\mathcal{M}, s) = \sum_{w \in \mathcal{A}} P(w|\mathcal{M}, s) I(w|\mathcal{M}, s) \quad (3)$$

It is well-known that both the instantaneous entropy (i.e. the expected information) of a predictive model, and the information supplied by the symbol to a model, tend to be high at the beginning of words in the case of natural language text and $n^{th}$–order markov models for small $n$. This fact may be used to extract the hypothesised chunking structure by simply thresholding the entropy. The obvious problem is to determine a good threshold value if one exists. An alternative is to threshold, perhaps dynamically, the information supplied to the model by the data. We shall refer to either process as *thresholded entropic chunking*.

In this paper we explore the use of thresholded entropic chunking, and propose two alternative entropic chunking methods which avoid the requirement of selecting a threshold value. We develop these methods on simple artificial corpora, before showing that they may be used to discover words in unsegmented natural language text. We conclude with a discussion on the use of the chunker in an adaptive data compression system.

## II. Chunking Methods

*Ad hoc* statistical techniques have been occasionally used to segment data sequences for linguistic applications [1], [2], and it has long been known that the entropy of a markov

J. Hutchens is with the Centre for Intelligent Information Processing Systems (CIIPS) within the Department of Electrical & Electronic Engineering at the University of Western Australia. E-mail: hutch@ciips.ee.uwa.edu.au

M. Alder is with the Department of Mathematics at the University of Western Australia. E-mail: mike@maths.uwa.edu.au

model for natural language gives a reasonably reliable indicator of where the boundary between two words lies. In this section, we present some chunking methods inspired by this observation with the intention of showing later how these methods may be applied to natural language text.

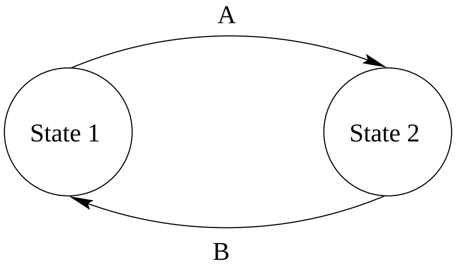


Fig. 1. A zero-information source.

In order to present the various chunking methods, we introduce a simple information source and two synthetic corpora which are realisations of it. The source, illustrated in figure 1, generates a deterministic repeating sequence. Corpus 1 rewrites A$\mapsto$`000` and B$\mapsto$`111`, resulting in the data sequence given in figure 2, while corpus 2 replaces A with one of {`aa`, `bb`, `cc`, `dd`}, chosen uniformly at random, and B with one of {`ww`, `xx`, `yy`, `zz`}, also chosen uniformly at random, and additionally appends a whitespace, which is represented by $\wedge$ throughout this paper. This results in the data sequence given in figure 3.

... `000111000111000111000111000111000111` ...

Fig. 2. A section of corpus 1.

A markov model of order 3 or greater is sufficient to detect that corpus 1 is deterministic, but it is also possible for a lower–order process to accomplish this identification, by finding the two chunks, replacing each chunk by a new symbol, and then applying a low order model to the new data stream. We shall call this inference process, of replacing chunks by new symbols, *UpWriting* the original data [3]. It is a process which can be iterated. Conversely, the usual rewrite process is referred to as the *DownWrite*.

... `aa`$\wedge$`yy`$\wedge$`cc`$\wedge$`yy`$\wedge$`aa`$\wedge$`ww`$\wedge$`bb`$\wedge$`xx`$\wedge$`dd`$\wedge$`zz`$\wedge$`cc`$\wedge$ ...

Fig. 3. A section of corpus 2.

Finding chunks in corpus 2 and inferring the model which in fact generated it is possible and of course the data may also be modelled by markov models of various orders. It is plain that there are advantages in modelling the UpWritten data rather than the original data; models for natural languages start with words, or at least stems and inflections, rather than letters of an alphabet. It would also be advantageous to model the UpWritten form of corpus 2 by finding the two symbol classes present, although this is beyond the scope of the present discussion.

## A. Segmentation

As we have previously noted, the instantaneous entropy of a predictive model fluctuates as the model scans across a sequence of symbols, and we observe in natural language text that this entropy reaches maximum values at the beginning of words. This is illustrated by figure 4, which plots the entropy of a $2^{nd}$–order markov model, trained on a corpus of Sherlock Holmes stories (at the letter level), as it scans across a single sentence (which was not present in the training corpus). We refer to this corpus as the *Sherlock corpus* throughout. [1]

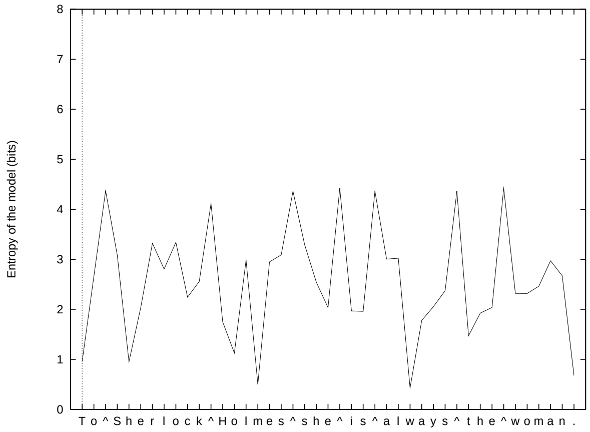


Fig. 4. Entropy across a text segment with respect to a $2^{nd}$–order markov model inferred from the Sherlock corpus.

The information provided to the model by a symbol exhibits similar properties, although the words boundaries aren't marked as clearly, as figure 5 illustrates. Clearly, a single threshold can not be used to segment this sequence into words; other approaches must be explored if $I(w|\mathcal{M}, s)$ is to be used as a chunk boundary indicator.

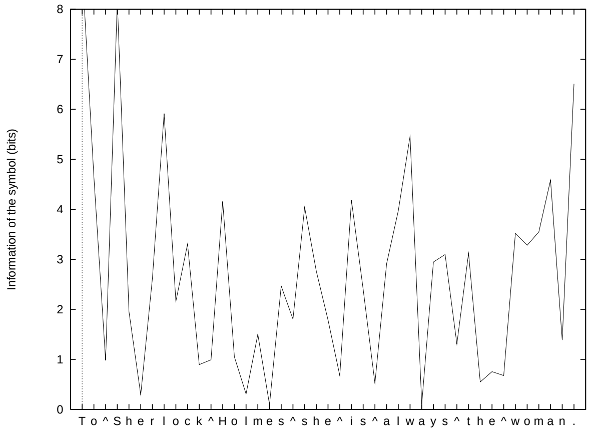


Fig. 5. Information of symbols in a text segment with respect to a $2^{nd}$–order markov model inferred from the Sherlock corpus.

It may also be observed that word boundaries are not clearly marked in natural language data which is devoid of whitespace and punctuation, as is apparent from figures 6 and 7. These plots were generated from a version of the

Sherlock corpus which had all whitespace and punctuation removed, and in which all characters were converted to their uppercase equivalents. We refer to this corpus as the *SHERLOCK corpus* throughout.

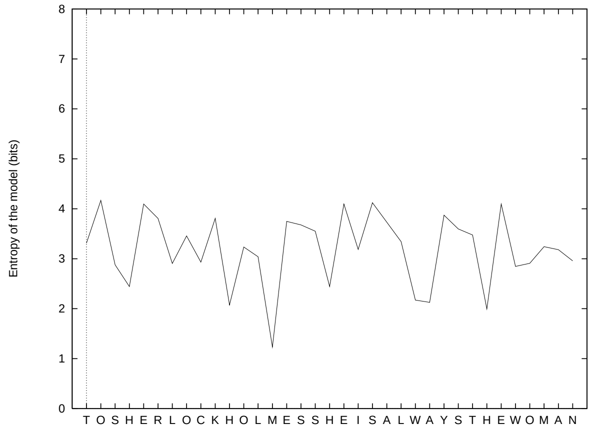


Fig. 6. Entropy across a text segment with respect to a $2^{nd}$–order markov model inferred from the SHERLOCK corpus.

It is obvious that techniques other than thresholded entropic chunking are required to successfully segment these examples.

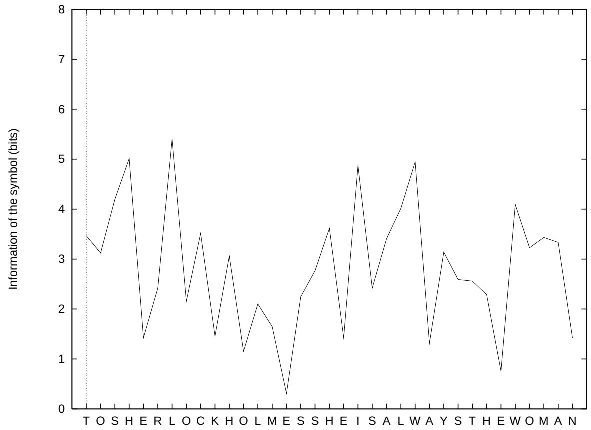


Fig. 7. Information of symbols in a text segment with respect to a $2^{nd}$–order markov model inferred from the SHERLOCK corpus.

Let us now apply the technique of thresholded entropic chunking to the two artificial corpora, beginning with corpus 1. The entropy of a $1^{st}$–order markov model remains constant over corpus 1, making segmentation impossible. A $2^{nd}$–order markov model, on the other hand, exhibits a suitably fluctuating instantaneous entropy, and the chunks `00` and `11` will evidently be formed for any threshold value between 0 and 1 bits.

If $I(w|\mathcal{M}, s)$ is thresholded rather than $H(\mathcal{M}, s)$, it is found that, for the $1^{st}$–order markov model, any threshold between $-log_2\frac{2}{3}$ and $-log_2\frac{1}{3}$ bits will recover the chunks `000` and `111`, which correspond exactly to our meta-alphabet. Inference of a $1^{st}$–order markov model from the UpWritten corpus is sufficient to conclude that the data is deterministic.

Turning to corpus 2, we find, for a $1^{st}$–order markov model, that thresholding either $I(w|\mathcal{M}, s)$ or $H(\mathcal{M}, s)$ with a threshold value which lies between 1 and 3 bits yields the chunks `aa`$\wedge$, `bb`$\wedge$, ..., `zz`$\wedge$. The meta-alphabet has been extracted perfectly, allowing restoration of the higher-level symbol sequence after a single UpWrite. A $2^{nd}$–order model achieves the same results for thresholds between 0 and 2 bits.

Thresholded entropic chunking can very successfully segment the two artificial corpora given, and, as we will show later, thresholding $H(\mathcal{M}, s)$ for low–order markov models works particularly well on natural language text, provided that whitespace is present between words. The only problem with this form of chunking is that a threshold needs to be selected *a priori*. If it is set too low, no chunks will be found; if it is set too high, the entire sequence will be chunked into a single symbol. We shall not introduce any strategies for estimating good thresholds here, apart from mentioning that, without any additional information about the data source, a threshold of $log_2\|A\|$ has been found to be as good as any other (and would function well for the two artificial corpora used here).

For data which is not as simple as our two examples, such as natural language text, it is generally impossible to find a single ideal threshold value. It is common for very long anomalous[2] chunks to be extracted, along with chunks we would consider to be useful, and procedures are then required for identifying and rejecting these anomalous chunks. We briefly discuss some procedures which may be employed when discarding anomalous chunks at the end of this section.

### *B. Identifying Separator Symbols*

In natural language text we find that the entropy of a predictor peaks when the whitespace symbol is the most recently encountered one. This suggests that we may be able to identify the symbols in the alphabet which function as whitespace, by simply finding the symbols which, on average, result in a high level of uncertainty in the model. If such symbols are found, UpWriting the data becomes a simple parsing process.

Identifying separator symbols is a matter of calculating, for each symbol in the alphabet, the average entropy of the model when the symbol is the most recently encountered one, according to equation 4. Symbols assigned a high value of $H_{av}$ generally cause confusion to the model when they occur, and may therefore function as whitespace.

$$H_{av}(w_i, \mathcal{M}) = \sum_{w=w_i} P(w_i|\mathcal{M}, s) H(\mathcal{M}, s, w_i) \qquad (4)$$

On corpus 1, using a $1^{st}$–order markov model, $H_{av}(\mathtt{0}, \mathcal{M})$ and $H_{av}(\mathtt{1}, \mathcal{M})$ are identical, which means that we cannot confidently say that a separator symbol exists in the corpus. This is true for higher-order models as well.

If we measure the average entropy due to each symbol for corpus 2, again with a $1^{st}$–order markov model, we find that $H_{av}(\wedge, \mathcal{M}) = 3$ bits, while $H_{av} = 1$ bit for all other symbols. This large discrepancy provides strong evidence that the whitespace character is a separator symbol, and the discrepancy exists for higher-order models as well.

### C. Agglutination

One final method of chunking which we will present here is that of creating a new chunk directly from the model. We find that some data, such as the SHERLOCK corpus, is difficult to segment using thresholded entropic chunking. Rather than attempting to recover all structure in the data in a single pass, a compromise is to extract only those chunks for which there is a lot of evidence, UpWrite the data using them, and iterate the process.

We can measure the entropy of an $n^{th}$–order model with respect to each of the symbol sequences of length $n$ encountered during training. The symbol sequences which result in the highest entropy may be formed into new chunks, used to UpWrite the data, and the entire process may be repeated. This approach has several advantages; both the chunks and the data become hierarchically structured, and no chunking threshold needs to be specified. Disadvantages are that a model or order 2 or higher needs to be used, that many iterations may be required to completely recover the higher-level structure of the data, and that many chunks may be formed merely to be used as "scaffolding" for building longer chunks. Such scaffolding chunks may be removed from the model at the end of processing, if they are deemed anomalous.

... `(00)0(11)1(00)0(11)1(00)0(11)1(00)0` ...

Fig. 8. An UpWritten section of corpus 1.

On corpus 1, a $2^{nd}$–order markov model finds `00` and `11` as chunks, as these two symbol sequences result in the highest model entropy. After an UpWrite, the data looks as in figure 8, where the symbols shown between the brackets constitute a chunk. No further iterations are possible, as each of the symbol sequences in the corpus result in a zero entropy, with respect to a $2^{nd}$–order markov model.

... `a(a∧)y(y∧)c(c∧)y(y∧)a(a∧)w(w∧)b(b∧)` ...

Fig. 9. Corpus 2 after the first UpWrite.

On corpus 2, the chunks `a∧`, `b∧`, ... , `z∧` are found, resulting in the UpWritten data shown in figure 9. The process may be repeated on the UpWritten corpus to find the chunks shown in figure 10.

... `(a(a∧))(y(y∧))(c(c∧))(y(y∧))(a(a∧))` ...

Fig. 10. Corpus 2 after the second UpWrite.

At this stage, additional chunks may be formed, but doing so does not decrease the average entropy of the model. As an example, although the symbol sequence `(a(a∧))(y(y∧))` results in a non-zero entropy with respect to a $2^{nd}$–order markov model, forming a new symbol out of it does not mean that the model can better predict what follows a sequence such as **zz∧aa∧yy∧**. The additional context provided by the new chunk does not significantly decrease the average entropy of the model. This suggests a rough heuristic which may be used as a stopping criterion, although we cannot conclude that such chunks won't eventually become useful, if the agglutination process was allowed to continue indefinitely.

Assuming that agglutination is halted after the second pass, it is found that the chunks are hierarchical in nature, and, if they are DownWritten to their lowest level, correspond exactly to the symbols in the original alphabet.

### D. Procedures for Removing Anomalous Chunks

As with all problems which involve real-world data, we cannot expect chunking to be completely successful. In fact, a large proportion of the chunks formed may be anomalous, either because of a poor choice of threshold (in the case of thresholded entropic chunking), or because of the existence of scaffolding chunks (in the case of agglutination). Furthermore, chunks may exist which we consider to be erroneous, even though they occur frequently in the data. An example is the concatenation of two or more words which frequently occur together.

Assuming that anomalous chunks have been identified by, for instance, measuring their frequency of occurrence, or by eye-balling the results ourselves, we may proceed to remove them from the model altogether. This can be done either by simply discarding the chunks, replacing them with the sequence of root-symbols out of which they are formed, or by hierarchically decomposing them.

The process of hierarchical decomposition is nothing more than a DownWrite; the anomalous chunk is DownWritten to a sequence of symbols, each of which may be either another chunk, or a root symbol. Any of the chunks in the DownWritten sequence which are themselves anomalous are also DownWritten, until the original anomalous chunk is decomposed into a sequence of non-anomalous chunks and root-symbols. This sequence is substituted for the anomalous chunk in the data, from which a new model may be inferred.

Although it may be beneficial to remove anomalous chunks from the model, we intentionally neglect to do so in the following section, in order to allow the performance of the chunker to be evaluated realistically.

## III. Experimental Results

We have introduced several methods of discovering chunks in an unsegmented symbolic time series—now we turn to the problem of finding words in English text. For the purpose of these experiments, we use the previously introduced Sherlock and SHERLOCK corpora. The majority of each corpus was used for inference; the 10 kilobytes or so not used for inference was reserved for testing the chunker.

### A. Segmentation

Entropic chunking is found to work particularly well on natural language text when whitespace is present between words. In fact, using a threshold of $\log_2 ||A||$ bits, which was chosen arbitrarily, and a $2^{nd}$–order markov model, it is found that the Sherlock corpus is segmented perfectly into a sequence of valid English words.

On the SHERLOCK corpus, segmentation was considerably poorer. Around 10% of the chunks found corresponded exactly to frequently occurring English words, such as `HAVE`, `WOULD` and `THE`. Many chunks corresponding to parts of words were also found, including `AD`, `SE` and `LD`. The remainder of the chunks consisted of sequences of valid words, such as `ISALWAYSTHE`, but, much more often, consisted of anomalous sequences such as `SULTSGRITINASE`. The fact that words, parts of words, and concatenations of words and word-parts were all discovered is disheartening, as it means that words can never be completely recovered via this process alone.

$$H_{net}(\mathcal{M}, s, w_x) = H(\mathcal{M}, s) - H(\mathcal{M}, s, w_x) \tag{5}$$

Clearly, straight entropic chunking cannot be used to find English words in a text such as this, as figure 6 made apparent. One solution is to use a low entropy threshold, and iterate the process, hoping to build up words by gradually chunking the chunks. Another solution is to threshold some other property, such as the net difference in entropies, as given in equation 5. Doing this yields superior results, especially when higher-order models are used. This method of segmentation was inspired by an examination of the entropy plot, and is therefore, by our own admission, extremely *ad hoc*.

In order to evaluate segmentation by thresholding of the net-entropy, we introduce three measures, defined in equations 6, 7 and 8, where $\mathcal{W}$ is the correct alphabet of words, $\mathcal{C}$ is the set of chunks formed, and $s = w_1, w_2, \ldots, w_n$ is the sequence of $n$ words used for testing. The measures of *accuracy* and *recall* have previously been used to evaluate the success of algorithms for discovering sequential structure in language [4].

$$\mathit{recall} = \mathit{Mike, Ineedhelpforthisone.} \tag{6}$$

$$\mathit{accuracy} = \frac{||\mathcal{W} \cap \mathcal{C}||}{||\mathcal{C}||} \tag{7}$$

$$\mathit{coverage} = \frac{||\mathcal{W} \cap \mathcal{C}||}{||\mathcal{W}||} \tag{8}$$

The *recall* is the proportion of words in $s$ which were correctly identified by the chunker; the *accuracy* is the proportion of chunks which correspond to correct words, and the *coverage* is the proportion of correct words which were found by the chunker. For the testing corpus, $n = 206$ and $||\mathcal{W}|| = 128$.

Note that both the *recall* and *coverage* are monotonically increasing measures, since the number of words in $s$, and the number of correct words, are both constant. This by no means suggests that both values will eventually approach 100%; merely that they won't get any worse as subsequent iterations are performed.

Experiments were performed on the SHERLOCK corpus using a variety of model orders and net-entropy thresholds. The best results were achieved with a $5^{th}$–order markov model, using a threshold of 1 bit. The *recall* was 47.6%, the *accuracy* was 39.4%, and the *coverage* was 50.8%. The chunks formed included `ADMIRABLE`, `INSTRUMENT` and `DISTURBING`, as well as more frequent English words such as `THE`, `OF` and `ALL`. Proper names, such as `IRENE` were formed, as were rare words, such as `SNEER`, which occurred only 8 times in the 659852-word training corpus.

The chunker failed by concatenating several words together to form phrases, as in `TOSHERLOCKHOLMES`, `IHAVE` and `HEWAS`. These chunks may be hierarchically decomposed into their constituent words, providing that such failures can be identified. Some chunks corresponded to parts of words, as in `ING`, `PRE` and `ED`, while some were completely erroneous, such as the non-words in the incorrect segmentations `(HE)(RUN)(DER)` (rather than `(HER)(UNDER)`), and `(PLACE)(DHIM)(SELF)` (rather than `(PLACED)(HIMSELF)`). These errors may be attributed to the fact that the chunker is unaware of future symbols, and therefore must immediately decide whether a chunk has been formed or not, without the advantage of look-ahead. We speculate on ways of improving results by introducing look-ahead near the end of this paper.

$$I_{net}(w_x|\mathcal{M}, s, w_{x-1}) = I(w_x|\mathcal{M}, s, w_{x-1}) - I(w_{x-1}|\mathcal{M}, s) \tag{9}$$

We have generally found that using the information of a symbol as a measure of where a chunk begins produces inferior results. Using an arbitrary threshold of $\log_2 ||A||$ bits on the Sherlock corpus yields a host of anomalous chunks, along with more useful chunks such as `It∧was∧`, `but∧` and `ing∧`. A mere 8% of the chunks found corresponded exactly to words. Similarly, the chunks found in the SHERLOCK corpus were poor, and generally spanned word boundaries. Segmenting $I_{net}$, as defined in equation 9, produces superior results, particularly with higher-order models; the *recall* of a $4^{th}$–order markov model with a net-information threshold of 1 bit on the SHERLOCK corpus was 46.6%, the *accuracy* was 27.6% and the *coverage* was 39.1%. Eyeballing of the results confirms that a better segmentation is achieved when the net-entropy of the model is thresholded, rather than the net-information.

### B. Identifying Separator Symbols

Figure 11 plots the average entropy of the twenty characters which cause the most confusion to a $2^{nd}$–order markov model for the Sherlock corpus. It can be seen that whitespace character $\wedge$ has a significantly higher average entropy than the other characters: $H_{av}(\wedge, \mathcal{M}) = 4.595$ bits, while $H_{av}(\mathtt{E}, \mathcal{M}) = 3.932$ bits—a difference of 0.663 bits. This is strong evidence that the whitespace symbol is a chunk separator.

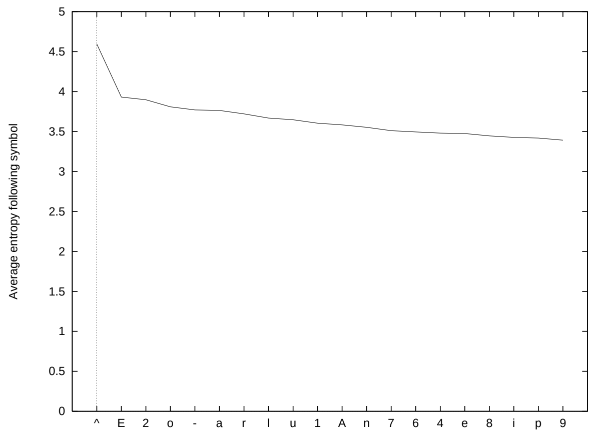


Fig. 11. Average entropy of a $2^{nd}$–order markov model caused by the specified symbol on the Sherlock corpus.

The average entropy of characters in the SHERLOCK corpus, as shown in figure 12, indicates that no separator symbol exists in the text, simply because the difference between the two highest average entropies is a mere 0.078 bits. Without a significant drop-off, we cannot confidently conclude that separator symbols exist.

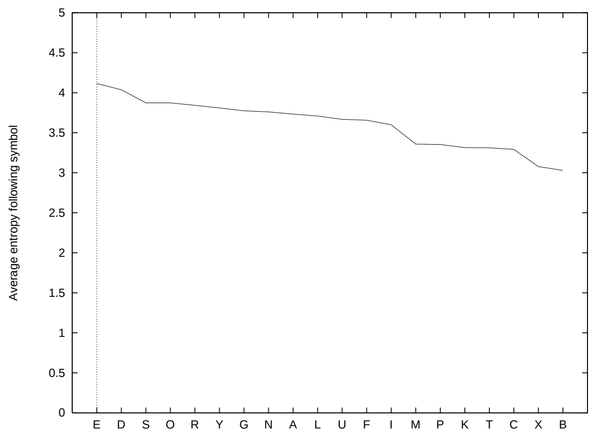


Fig. 12. Average entropy of a $2^{nd}$–order markov model caused by the specified symbol on the SHERLOCK corpus.

Note, however, that in this case, the symbols with the highest average entropy tend to occur at the end of words. In fact, the three symbols with the highest average entropy occur at the end of over 42% of the words in the corpus. This is an encouraging vindication of our choice of entropy as the primary indicator of the location of chunk boundaries.

*C. Agglutination*

To test the process of agglutination, a $2^{nd}$–order markov model was inferred from the SHERLOCK corpus. The symbol sequence found to result in the highest model uncertainty was `AD`, and this was added to the alphabet and used to UpWrite the data. After 118 iterations, the *recall* was 57.3%, the *accuracy* was 16.1%, and the *coverage* was 47.2%. We would expect further iterations to increase both the *recall* and the *coverage* at the expense of the *accuracy*, merely because not all possible words were formed after so few iterations.

These results compare favourably with those found using thresholded entropic chunking with a $5^{th}$–order markov model. The higher value of the *recall* is encouraging, as it means that more of the words in the corpus were correctly identified. The considerably lower value of the *accuracy* is due to the fact that many of the chunks in the alphabet serve merely as scaffolding for longer chunks. For instance, the original chunk, `AD`, occurs only within longer chunks at the end of the process.

The fact that comparable results have been achieved with a significantly simpler markov model prompts us to conclude that agglutination is preferable to thresholded entropic chunking if separator symbols are found not to exist in the data, and we can afford to perform many iterations over the data in order to form chunks.

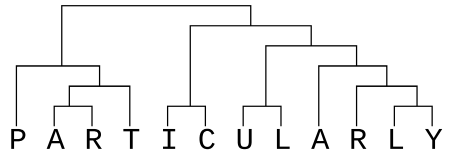


Fig. 13. Hierarchical decomposition of the chunk `PARTICULARLY` as formed by the agglutination process.

To illustrate the hierarchical nature of the chunks formed by the agglutination process, consider the decomposition of the chunks `PARTICULARLY`, shown in figure 13, and `QUESTIONABLE`, show in figure 14. Note that in both cases, the words are composed of sensible sub-parts—`QUESTIONABLE`, for instance, is composed of the sub-parts `QUESTION` and `ABLE`. This is a feature shared by all of the words which were correctly identified by the chunker.

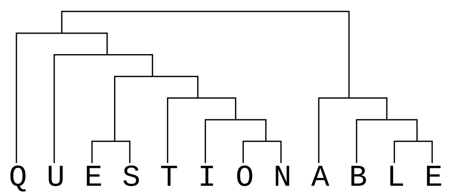


Fig. 14. Hierarchical decomposition of the chunk `QUESTIONABLE` as formed by the agglutination process.

The agglutination process does make mistakes. Consider the decompositions show for the chunks `SHERLOCKHOLMES`, shown in figure 15, and `ITWAS`, shown in figure 16. In both cases, the second word has been correctly identified, but the first word has not. This is due to the nature of the agglutination process; symbols pairs are joined together to form a chunk if the pair causes the model to exhibit a high uncertainty.

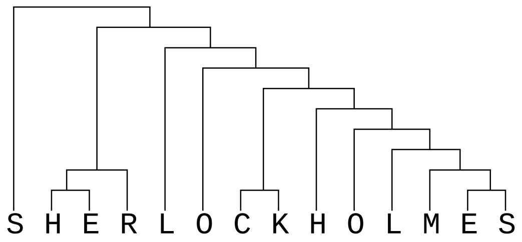


Fig. 15. Hierarchical decomposition of the chunk `SHERLOCKHOLMES` as formed by the agglutination process.

It seems that the model is generally pretty good at predicting what follows the symbol sequence `SHERLOCK` (it is usually an `H`), but considerably poorer at predicting what follows the chunk `HOLMES`. Therefore the model erroneously forms a chunk from `CK` and `HOLMES`, in the belief that this will enable it to better predict what follows, rather than forming the chunk `SHERLOCK` first. These kind of mistakes are unavoidable, and are a side-effect of the method of agglutination used.

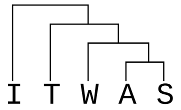


Fig. 16. Hierarchical decomposition of the chunk `ITWAS` as formed by the agglutination process.

Erroneous chunks therefore almost always exhibit a similar structure; they consist of a cascade, resulting from symbols being iteratively glued onto the left hand side of a valid chunk. Whether this property is useful for detecting anomalous chunks is debatable, and open to investigation.

## IV. Better Segmentation with Different Models

In this paper we have concentrated on chunking methods which use predictive models; such methods are necessarily constrained by the fact that they must chunk the data without reference to the symbols which occur in the future. Examination of the experimental results suggests that superior chunking performance may be attainable should this restriction be relaxed.

One rather *naïvé* way of incorporating information about the future is to use two predictive models: one working from the start of the time series to the end, and the other working in the reverse direction. Using measures from both models may reduce the number of bad segmentations made, especially those of the form `(PLACE)(DHIM)(SELF)`. Exactly how measures from more than one model may be combined in order to improve results is not obvious, but it seems that using a second model to verify the chunk boundary locations postulated by the first model may prove worthwhile.

Another solution is to tailor the nature of the model to the segmentation problem. One such model may use a context which surrounds the symbol being predicted. Such models would hopefully register a higher entropy when the two segments of the context lie within different words.

We are reluctant to pursue work in this area, as our aim is to formulate a generic acquisition device which incorporates the processes of chunking and UpWriting. Such a device may be used for a variety of applications, some of which will certainly not deal with natural language text. Tailoring the model in order to improve segmentation of natural language text is a violation of this, as it presupposes the structure of the data to be segmented, and therefore incorporates undesirable assumptions about the data into the model.

## V. Adaptive Data Compression

Adaptive data compression uses predictive models to encode symbols for efficient transmission or storage. The statistics of the models are updated on the fly, to ensure that the model available to the encoder is identical to that available to the decoder at all stages of the process. The fact that the model is automatically inferred from the data precludes its transmission, resulting in text compression which is state of the art.

The chunking methods presented in this paper may be used in adaptive statistical models, perhaps by periodically using the model to look for chunks in the data which has already been processed, and adding any chunks to the model's alphabet if they are deemed to be useful (if, for example, their use would improve the compression performance of the model).

Although it seems non-intuitive, adaptive data compression systems may also look-ahead, if this is found to improve performance. The transmitter may look ahead as far in the data stream as it needs to, as part of the chunking process, so long as it notifies the receiver, by transmitting a special control symbol perhaps, whenever a chunk is formed. This is necessary because the receiver does not have the advantage of look-ahead.

We hope to demonstrate the feasibility of using a chunker as part of an adaptive data compression algorithm in a future paper, and preliminary work suggests that text compression performance should improve [5].

## VI. Conclusion

In this paper we have introduced several methods for discovering chunks in symbolic time series, and we have demonstrated how these methods may be used to discover words in natural language text. Although these methods do discover words rather well, particularly when whitespace is present in the text, we did not explicitly design the chunking methods to solve this particular problem. Rather, we have developed methods for finding chunks in arbitrary data, and we consider a chunk to represent structure inherent in the data which may be beneficial to incorporate as part of the modelling process. Future work includes applying chunking to the task of adaptive data compression, and developing "look-ahead" models specifically for the task of finding words in unsegmented natural language text.

## Notes

[1] This corpus was formed from a collection of Sherlock Holmes stories which were obtained from the Oxford Text Archive, located at `http://ota.ox.ac.uk`. The stories were stripped of all SGML processing directives, linefeed characters were removed, and they were concatenated to give a single corpus, around 3.5 megabytes in size [6].

[2] Anomalous in that they are infrequent, often only occurring once in the data.

## References

[1] Zellig S. Harris, "From phoneme to morpheme," *Language*, vol. 31, pp. 190–222, 1955.

[2] J. G. Wolff, "An algorithm for the segmentation of an artificial language analogue," *British Journal of Psychology*, vol. 66, no. 1, pp. 79–90, 1975.

[3] Michael D. Alder, "Inference of syntax for point sets," in *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*, E. S. Gelsema and L. N. Kanal, Eds. June 1994, number 16 in Machine Intelligence and Pattern Recognition, pp. 45–58, Elsevier Science B.V.

[4] Craig G. Nevill-Manning, *Inferring Sequential Structure*, Ph.D. thesis, University of Waikato, 1996.

[5] Jason L. Hutchens, "Finding structure via compression," in *NeMLaP3 / CoNLL98: New Methods in Language Processing and Computational Language Learning, ACL*, David M. W. Powers, Ed., January 1998, pp. 79–82.

[6] Sir Arthur Conan Doyle, "The Sherlock/SHERLOCK corpora," These corpora were formed by Jason Hutchens from a collection of texts downloaded from the Oxford Text Archive, at: `http://ota.ox.ac.uk`

[7] Claude E. Shannon and Warren Weaver, *The Mathematical theory of Communication*, University of Illinois Press, 1949.

[8] Timothy C. Bell, John G. Cleary, and Ian H. Witten, *Text Compression*, Prentice Hall, 1990.

[9] Eugene Charniak, *Statistical Language Learning*, MIT Press, 1993.