On the Role of Low-Level Linguistic Tasks for Reading Time Prediction

Franck Dary, Abdellah Fourtassi, Alexis Nasr

On the Role of Low-Level Linguistic Tasks for Reading Time Prediction

Franck Dary, Abdellah Fourtassi, Alexis Nasr
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
{franck.dary,alexis.nasr,abdellah.fourtassi}@lis-lab.fr

Abstract

It has been shown that complexity metrics, computed by a syntactic parser, is a predictor of human reading time, which is an approximation of human sentence comprehension difficulty. Nevertheless, parsers usually take as input sentences that have already been processed or even manually annotated. We propose to study a more realistic scenario, where the various processing levels (tokenization, PoS and morphology tagging, lemmatization, syntactic parsing and sentence segmentation) are predicted incrementally from raw text. To this end, we propose a versatile modeling framework, we call the Reading Machine, that performs all such linguistic tasks and allows to incorporate cognitive constrains such as incrementality. We illustrate the behavior of this setting through a case study where we test the hypothesis that the complexity metrics computed at different processing levels predicts human reading difficulty, and that when cognitive constraints are applied to the machine (e.g., incrementality), it yields better predictions.

Keywords: reading time; cognitive modeling; entropy; surprisal

Introduction

Over the last couple of decades, there has been a growing interest in using Natural Language Processing tools to develop cognitively-plausible models of human sentence processing (see Hale (2017) for a review). Researchers have proposed to interface NLP models with human processing through complexity measures. These measures hypothesize that the probability of a word given its linguistic context (as calculated by the model) can predict the difficulty with which humans process this word (as measured, e.g., by the time it takes to read it). Several previous studies have shown that the probability with which a word fits into a syntactic parse predicts human processing difficulty (Demberg & Keller, 2008; Boston, Hale, Kliegl, Patil, & Vasishth, 2008; Boston, Hale, Vasishth, & Kliegl, 2011; Crabbé, Fabre, & Pallier, 2019).

Despite being successful in accounting for some empirical data, much of these previous studies remains short of accounting for the full complexity of human sentence processing. In particular, they have generally focused on the syntactic parsing while assuming other levels of sentence processing to have been independently and successfully analyzed (e.g., words are typically assumed to come with their parts of speech labels). Nevertheless, when reading a sentence, humans perform many linguistic analyses besides syntactic parsing: they segment the stream of characters into words, categorize these units into parts of speech, analyze their morphology, and determine the sentence boundaries.

Critically, each of these processing steps involve making decisions and, thus, induce a processing cost that may influence the reading time. In order to test this hypothesis, we need to build a model that mirrors the complexity of human sentence processing by "constructing" all these linguistic levels, making precise predictions about the contribution of each level of processing on the reading time. The current work is an attempt to achieve this goal.

We start from a standard syntactic parser and we extend it by making it able to construct various linguistic levels, assuming only the ability to process a string of characters. For each word in a given linguistic context, our model is able to produce complexity measures across all processing levels, allowing us to test the extent to which each level influences the reading time of this word. Two measures are compared. The first is *surprisal*, a standard complexity measure for parsing, which is based on the probability of the action performed by a parser at a given time. The second one is the *entropy* of the probability distribution over the actions that the parser can execute at a given time.

We chose as a starting point the Transition Based Parsing (TBP), an incremental model of syntactic parsing that produces dependency structures (Yamada & Matsumoto, 2003; Nivre, 2003). This choice was motivated by the fact that the underlying formalism is quite simple and transparent, while at the same time providing an adequate model of human syntax processing (Boston et al., 2008, 2011).

We provide an extension of TBP that we call the Reading Machine (RM). This extension is flexible in the sense that it can be modulated to instantiate various processing hypotheses. For example, we can choose the order in which the linguistic levels are processed or the the amount of information that the model has access to. In the current work, the goal is not to contrast all possible processing strategies, but to test whether different levels of sentence processing influence reading time. That said, it could be useful to compare at least two strategies that vary in terms of their psychological plausibility.

We chose to instantiate two strategies that differ in whether or not they are incremental, i.e., in whether or not the processing of each word at every linguistic level is done the moment the word is discovered. The incremental strategy processes a given word at various linguistic levels before switching to the next word. In contrast, the non-incremental strategy process the whole text at a given linguistic level before changing level.

The incremental strategy is more psychologically plausible, thus, we expect it to fare better than the non-incremental strategy in terms of its ability to produce metrics predicting human processing difficulty.

To sum up, the paper attempts at making several contributions: 1) investigating how complexity measures of low-level linguistic processing correlate with reading time, 2) testing whether introducing more cognitively plausible processing improve the predictions, and 3) comparing two complexity measures (i.e., suprisal and entropy) in terms of how they fare in connecting our reading architecture to human data.

The structure of the paper is the following. In section The Reading Machine, we give an overview of TBP and define our extension of this model, which we call the Reading Machine (RM). Section Complexity Metrics introduces the two complexity metrics that will be used in our experiments. In section Experiments, the models are evaluated with respect to their ability to predict human reading time. Section Conclusion provides concluding remarks.

The Reading Machine

The model we propose in this paper is an extension of Transition Based Parsing (TBP). This algorithm builds the dependency tree of a sentence by scanning it word by word, in reading order. At each step, an *action*, (also called *transition*) is predicted, that adds, in general, a new dependency to the tree being produced. The actions are predicted by a classifier that takes as input the current *configuration* of the parser and computes a probability for every possible action. The action selected is applied to the current configuration and yields a new one.

The algorithm is greedy[1]. At each step, a single decision is taken corresponding to a local maximum (the action with the highest probability), it does not guarantee that the tree produced at the end is the one with the highest probability, where the probability of the tree is simply the product of the probabilities of the actions that led to its construction.

From TBP to RM

The original TBP focuses on the syntactic level of processing, taking as input a text which has already been tokenized, PoS tagged and segmented in sentences[2]. As we explained in the introduction, this falls short of accounting for the complexity of human behavior which processes multiple linguistic levels simultaneously. In the following paragraph, we explain how we extended the TBP framework to account for this complexity.

The main modification to the TBP framework that we propose is to consider every processing decision (such as PoS tagging, lemmatization, segmentation ...) as a transition that performs an action (such as selecting a PoS tag for a word) and changes the current *state* of the machine. Each state corresponds to a processing level, it is associated with a classifier that predicts the next transition to take.

Although a large number of linguistic processing tasks can be incorporated in a RM, we will be dealing in this work with six of them: tokenization (TOK), part of speech tagging (POS), lemmatization (LEM), morphological analysis (MRF), syntactic parsing (SYN) and sentence segmentation (SEG), each of which corresponds to a state in the RM.

A RM can be seen as a deterministic finite state automaton. The structure of the automaton (its states and transitions) defines the order in which the predictions are made. We will refer to the structure of a RM as its *strategy*. Two different strategies are represented in figure 1. The one above, referred to as INCR, implements an incremental strategy while the one below, ¬INCR, implements a standard Natural Language Processing pipeline strategy (which is non-incremental because it has access to the low-level right-context predicted linguistic annotations when making a decision). The actions that label the transitions have been omitted for readability.

The main difference between the two strategies comes from the loops on all states of the ¬INCR strategy. These loops model the non-incremental behavior of the RM: the whole text is processed at a given level before switching to the next linguistic level (the next state of the machine). The transition to the next state of the machine can only be traversed when the end of the input tape is reached. Traversing this transition resets the character and the character index and the word index to zero. In contrast, in the INCR strategy, a word is processed at every level before switching to the next word.

The action performed by a transition generally consists in writing a symbol on a *tape*. There are tapes for every type of prediction. Tapes can be read tapes, write tapes or both. A RM has one input tape, which is a read tape and an arbitrary number of output tapes which are read/write tapes.

The input tape contains the text to parse. It is character based: each cell of the tape contains a character. The text has not been linguistically pre-processed: it has not been segmented into sentences nor into words. The current position of the input tape's reading head is called the *character index*.

Output tapes are word based: each cell of a tape refers to a word of the input text. Output tapes are used to write the predictions made by the machine, generally one tape per type of prediction. These tapes are synchronized: at all times, the head is at the same position for all tapes. This position is called the *word index*.

Table 1 represents the tapes of a machine after processing

[1] This is true of the original algorithm. There have been many propositions for exploring in parallel a reduced number of alternative analyses using a beam (Huang & Sagae, 2010).

[2] Once again, this is true for the original TBP approach. There have been several attempts to extend TBP, mainly in order to realize simultaneously several linguistic tasks. Both Bohnet and Nivre (2012) and Alberti, Weiss, Coppola, and Petrov (2015) shows how a transition system can be extended and trained to jointly predict POS tags and the dependency tree, improving both the accuracy of the tagging and the parsing. Constant and Nivre (2016) extends the arc-standard transition system in order to jointly predict the syntactic tree and some aspects of the tokenization process: the identification of multiword expressions. The three above-mentioned systems take a pre-processed (tokenized and segmented into sentences) text as an input.

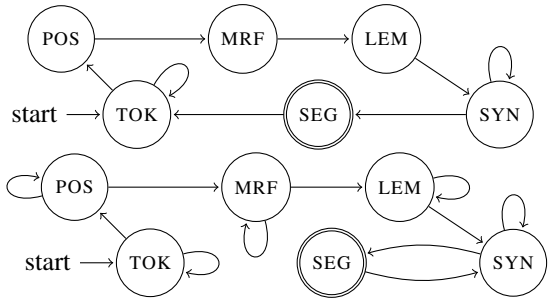

Figure 1: Two RM strategies defining the order of predictions. Above, the INCR strategy and below the ¬INCR strategy.

Table 1: Input and output tapes of a RM after processing the text *The boy hits the ball*.

SEG	NO	NO	NO	NO	NO	YES
SYN	DET	SUB	ROOT	DET	OBJ	PCT
GOV	+1	+1	0	+1	-2	-3
LEM	@	@	s@	@	@	@
MRF	DEF	SG	P3S	DEF	SG	-
POS	DET	N	V	DET	N	PCT
TOK	the	boy	hits	the	ball	.
INPUT	t h e _	b o y _	h i t s _	t h e _	b a l l	.

the text *The boy hits the ball.* The machine has 7 output tapes[3] and one input tape, represented at the bottom. When the input text is processed by the ¬INCR machine, the table is filled line by line, bottom-up. In contrast, when the input text is processed by the INCR machine, the table is filled column by column, from left to right. It is important to note that, in the case of INCR, predictions that have been made in the past for upper levels are taken into account for current lower level predictions. Predicting PoS tag for word *i*, for example, can use the syntactic structure predicted up to word $i-1$. This feature allows to take into account some top-down dependencies across linguistic levels.

Casting the six different linguistic tasks we are dealing with as predicting a transition is straightforward for tagging tasks, such as PoS tagging and morphological tagging. Corresponding transitions simply write a symbol (such as a PoS tag) on the corresponding tape at the word index position.

The case of lemmatization, tokenization and sentence segmentation ask for some explanation. It is not convenient to see the lemmatization task as a classification task, due to the large number of classes (potentially all the lemmas of a language). Besides, lemmatization is, to a large extent, regular. In order to capture this regularity, the classifier that realizes the lemmatization task predicts editing rules of the form `s1@s2` where `s1` is a suffix of the word to lemmatize and `s2` the suffix of the lemma[4]. When applied to a word *w*, such a rule strips off suffix `s1` from *w* and concatenates the result with `s2`, as in the following example $apply(\texttt{s@}, hits) = hit$. The actions predicted by the tokenizer are of four types: ADD_n adds the *n* next characters of the input tape to the current word and moves the character index *n* positions to the right, IGNORE ignores the current character (typically spaces) and moves the character index to the right, WORD marks the current word as complete and SPLIT_W^w action moves the character index $|w|$ positions to the right and adds the word sequence *W* in the buffer. This last action is used to expand contractions such as *don't* → *do not*. Sentence segmentation is realized by a binary action EOS(YES/NO) which tags the current word as the end of the current sentence or not. When an end of sentence is predicted, the syntactic root of the current sentence is set to the deepest stack element without a governor, and the stack is emptied[5]. In Table 2 we list all actions used in our RM architecture.

The states of the RM are linked to a neural network classifier. In this work we chose to share the same feature function for all states. This feature function transforms the current configuration (tape contents) into an embedding: a sliding window of size [-3,0] for the INCR machine and [-3,2] for ¬INCR, is centered on the *word index* and placed over the tapes, yielding the current context, and each tape component of this context is fed into its own Bi-LSTM encoder. The resulting embedding is then given to a specific decision layer (one for each state), predicting a probability distribution over the possible actions.

Complexity Metrics

Surprisal

Various complexity metrics have been proposed in the literature, that measure the difficulty for a parser to process a word in its sentential context, and relate this difficulty to human behavior when reading the same word (e.g., reading time). Here we use *Surprisal*, a measure that was originally introduced in the context of phrase structure grammars by Hale (2001), but which has then been adapted to the TBP framework by Boston et al. (2011).

Surprisal (Attneave, 1959) has its roots in information theory, it is defined as the logarithm of the reciprocal of a probability. It represents the intuitive idea that low probability events are surprising. Surprisal has been used as a measure of the processing difficulty at a word. Given a sentence $S = w_1 \dots w_n$, surprisal at word *i* is defined as follows: $Surprisal(i) = -\log_2(\frac{\alpha_i}{\alpha_{i-1}})$, where α_i is the probability of the prefix $w_1 \dots w_i$. Given the prefix $w_1 \dots w_{i-1}$, if word w_i is unexpected, the ratio $\frac{\alpha_i}{\alpha_{i-1}}$ will tend to be low and therefore, the surprisal of word w_i, high.

[3] Both tapes SYN and GOV are filled by the parser (the SYN state of the RM).

[4] Of course, such a simple form of rules can only deal with suffixal flexional morphology, which is the case of English. More complex morphology phenomena ask for more elaborate types of rules, such as the prediction of templates for templatic morphology.

[5] If there remain — beside root — stack elements without a governor, they are automatically attached to the predicted root word before emptying the stack.

Table 2: Actions used in our RM architecture. b.0 stands for the current word and s.0 for the word on top of the stack.

State	Action	Description
TOK	ADD_n	Adds the n next symbols to b.0.
TOK	IGNORE	Ignores the next symbol.
TOK	WORD	Marks b.0 as complete.
TOK	SPLIT_W^w	Consume symbol sequence w. Add word sequence W in buffer.
POS, MRF	TAG_L(t)	Writes tag t to b.0 on tape L.
LEM	s@s′	b.0 lemma := form $-\,s+s'$.
LEM	CASE_{ul}	b.0 lemma to upper/lower case.
SYN	REDUCE	Pop the the stack.
SYN	SHIFT	Push b.0 on the stack.
SYN	RIGHT_l	Adds arc (s.0,b.0,l). Push b.0 on the stack.
SYN	LEFT_l	Adds arc (b.0,s.0,l). Pop the stack.
SEG	EOS(Y/N)	Mark b.0 as an end of sentence, set sentence root, attach orphans to root then empty stack.

In its original formulation by Hale, α_i is computed as the sum of the probabilities of the trees that a Probabilistic Context Free Grammar associates to the prefix $w_1 \dots w_i$. In other words all the possible syntactic structures that the grammar associates to the prefix. α_i can be efficiently computed using a dynamic programming algorithm, the algorithm of Earley, in the case of Hale. Surprisal has been adapted to the framework of transition based parsing by Boston et al. (2011). The general definition remains unchanged, but the way α_i is computed is modified: it is the sum of the opposite of the probability logarithms of the transitions predicted by the parser up to word i[6]. In other words, instead of summing over all the possible parses up to word i, the new definition of α_i only considers the probability of the tree built by the parser after processing word i. Remember that, due to the greedy nature of the transition based parsing algorithm, this probability does not correspond to the probability of the highest scoring tree up to word i.

The prefix probability α as defined by Boston et al. (2011) is much poorer than the original definition of Hale, since it is only based on a single analysis for a given prefix. In order to take into account more than one possible syntactic structure of the prefix when computing its probability, Boston et al. (2011) define the quantity α_i^k which is the sum of the probabilities of all configurations in a beam of width k. In the case of $k = 1$, the prefix probability is, as mentioned before, the probability of a single transition sequence. Boston et al. (2011) showed that surprisal models parsing difficulty was reflected in human fixation durations for various values of k ($1 \leq k \leq 100$).

We chose to keep the simplest setting ($k = 1$) in our experiments. The computation of α_i in this situation is simply the sum of negative log probabilities of the transitions predicted up to word i and $\mathit{Surprisal}(i)$ is simply the part of this sum for the transitions that the parser associates to word i.

Surprisal generalizes easily to all kind of predictions that are made by the RM. It is extremely simple to compute for PoS tagging, morphological tagging and lemmatization since all these processes exactly predict one transition for every word of the sentence. Surprisal, in such a case, is simply $-\log(p_t)$ where t is the transition that has been selected. In the case of the tokenizer, the surprisal is computed on the sequence of transitions that add characters to the current word, until it is completed.

Entropy

As one can see, surprisal is computed based only on the probability of a single transition or a sequence of transitions. It's not a perfect way to measure how much the machine "hesitates" between several transitions that have close probabilities. In order to introduce this information in the complexity metrics, we also use the entropy[7] of the distribution computed by a classifier over the possible transitions. Entropy measures how uniformly the probability mass is distributed over the different transitions. We will refer to this complexity measure for word i simply as $\mathit{Entropy}(i)$.

In the case of a tagging task (such as PoS tagging, morphological tagging, lemmatization and segmentation), there is a single transition associated to word i and therefore a single distribution. In this case, $\mathit{Entropy}(i)$ is simply the entropy of this distribution. In the case of the parser, we associate to w_i the transition t that attaches w_i to its syntactic governor[8] . In this case, $\mathit{Entropy}(i)$ is the entropy of the distribution from which t was drawn. In the case of the tokenizer, $\mathit{Entropy}(i)$ is the mean of the entropies of the distributions of the transitions that led to identifying w_i.

Experiments

We test the hypothesis that model-derived complexity metrics predict human reading time across several linguistic levels, and not just syntax. We test this hypothesis across both complexity measures (i.e., surprisal and entropy), and across processing strategy (i.e., INCR and ¬INCR).

[6]Since surprisal is a word based measure, one has to decide, in a sequence of transitions the parser predicts for a sentence, which transition should be associated to which word. We follow Boston et al. (2011) and consider that all transitions that are predicted while word w_i is the first word of the parser's buffer are associated to w_i.

[7]Not to be confused with the entropy as operationalized, e.g., in Hale (2006) or Keller (2004) (.i.e. how much information words convey).We call our complexity metric Entropy because it is the entropy of the probability distribution produced by our neural network.

[8]It is important to note that the entropy of word i can take into account words not seen yet (this is what happens for left dependencies). This could be problematic for we cannot, in an incremental set-up, base the fixation time for a word on future events. The reason why this can be done here is that the variable we are predicting is the total reading time which is a complex measure that sums all fixations made by the reader, including fixations made when re-reading a region.

The Provo Corpus

We have used for our experiments the Provo corpus (Luke & Christianson, 2018), which consist of the recording of the eye movements of 84 skilled and native American English readers while reading 55 texts (2744 words). For each word and for each participant, several reading time related variables are computed. The variable we use is the total reading time: summation of the duration across all fixations on any given word. The data has been cleaned by the Provo team, removing fixations shorter than 80 ms and longer than 800 ms. We removed words that have been skipped and words for which no data were available.

Producing Complexity Metrics for the Corpus

The two machines INCR and ¬INCR, defined in section The Reading Machine, have been trained on the English Web Treebank from Universal Dependencies (Zeman et al., 2019) that consist in English sentences collected from the internet. This corpus is linguistically annotated on all six levels of interest to our system, namely: tokenization, PoS tagging, morphological tagging, lemmatization, dependency tree and sentence segmentation.

After training, the two machines were used to process the Provo corpus. For each word of the corpus, and for each linguistic level, a prediction is performed (e.g., selecting the PoS tag of the word) and both surprisal and entropy are computed.[9]

The output of the machines consited of 6 surprisal values and 6 entropy values (one for each RM state). These measures were merged with the eye tracking recording, yielding our final data frame. It is composed of 96,111 data-points, each point consisting in the following information (in addition to the linguistic measures obtained by our machines): word being fixated, word length (wd_length), word frequency[10] (freq), next word frequency (next_freq), ID of the human reader, total fixation time (in ms).

Experimental Setup

The methodology used is the following: we fit multiple linear mixed-effects regression models on the total reading time, where each model is a combination of a metric (surprisal or entropy) and a RM (¬INCR or INCR). These models are defined in table 3. Then we compared goodness of fit to determine which model was better. To this end, we used the difference in Akaike Information Criterion (AIC) as a criterion to discriminate between linear mixed effect models, following in this regard some similar previous work (Shaw & Kawahara, 2019; Frank, Otten, Galli, & Vigliocco, 2013).

[9]Due to the lack of manual linguistic annotations for the Provo corpus, we were not able to estimate the quality of the linguisitic predictions made by our machines. Nevertheless, we tested the accuracy of these machines on the test dataset of the UD corpora and found that its performances were in the typical range of similar (greedy) models.

[10]Estimated using Google's *One billion words benchmark for language modeling* corpus.

[11]Two way interaction.

Table 3: Linear Mixed Models used in our experiments.

Name	Fixed Effect
VANILLA	freq+next_freq+wd_length+freq:wd_length[11]
INCR_SUR	{VANILLA}+INCR_SUR_{6 linguistic levels}
INCR_ENT	{VANILLA}+INCR_ENT_{6 linguistic levels}
¬INCR_SUR	{VANILLA}+¬INCR_SUR_{6 linguistic levels}
¬INCR_ENT	{VANILLA}+¬INCR_ENT_{6 linguistic levels}
Random Effect: participant/sentence	

Table 4: Top 3 correlations for each metric and each machine.

ENTROPY			
INCR		**¬INCR**	
Variables	**Corr**	**Variables**	**Corr**
lemma/morpho	0.48	lemma/morpho	0.46
morpho/tagger	0.40	morpho/tagger	0.37
parser/tagger	0.19	morpho/parser	0.30
SURPRISAL			
INCR		**¬INCR**	
Variables	**Corr**	**Variables**	**Corr**
lemma/morpho	0.43	morpho/tagger	0.26
morpho/tagger	0.27	morpho/parser	0.22
morpho/parser	0.24	lemma/morpho	0.19

As a baseline, we use standard predictors for modeling reading times: frequency and word length. We tried to incorporate bigram frequencies, but this resulted in a slightly worse fit than the current and next word unigram frequencies. These frequencies have been log transformed and all other variables have been scaled to fit in the same intervals. Models were fitted using the lmer function of the lme4(v.1.1-23) package of the R Project. Using as the random effect's grouping factor the nested relation PARTICIPANT_ID/SENTENCE_ID. Note that to allow model comparison via likelihood ratio tests, models were fitted using the maximum likelihood method instead of restricted maximum likelihood method which is the default in lmer.

Results

We investigate first the extent to which metrics produced at the different linguistic levels carry redundant information. In order to do so, we compute for each machine (INCR and ¬INCR) and each complexity metric (surprisal and entropy), the Pearson correlation coefficient for every pair of the six linguistic predictors. As one can see in Table 4, the correlation between the complexity values among linguistic levels are quite low, showing that the information carried by these variables is not redundant. We also computed the correlations between our complexity metrics at the word segmentation level (tokenization) and word frequencies, and found no correlation (The highest absolute value being 0.025 between word frequency and tokenization entropy produced by INCR).

Table 5: Estimates of fixed effects for all 4 models.

	ESTIMATE			
	ENTROPY		SURPRISAL	
Effect	¬INCR	INCR	¬INCR	INCR
(intercept)	344_{***}	$\mathbf{341}_{***}$	345_{***}	343_{***}
segmenter	-1.7_{**}	$\mathbf{4.8}_{***}$	-1.9_{***}	4.3_{***}
parser	1.1_{-}	$\mathbf{2.6}_{***}$	1.2_{*}	0.3_{-}
lemma	2.1_{**}	$\mathbf{0.4}_{-}$	0.7_{-}	1.4_{*}
morpho	-1.0_{-}	$\mathbf{2.4}_{***}$	0.4_{-}	1.3_{*}
tagger	3.9_{***}	$\mathbf{1.8}_{**}$	2.8_{***}	2.0_{***}
tokenizer	1.1_{*}	$\mathbf{3.5}_{***}$	1.2_{*}	2.6_{***}
freq:wd_length	-3.7_{***}	$\mathbf{-3.7}_{***}$	-3.8_{***}	-3.7_{***}
freq	-8.7_{***}	$\mathbf{-8.2}_{***}$	-8.9_{***}	-8.6_{***}
wd_length	46.9_{***}	$\mathbf{47.6}_{***}$	47.4_{***}	47.3_{***}
next_freq	-1.8_{***}	$\mathbf{-1.7}_{***}$	-1.8_{***}	-1.7_{***}

Table 6: Pairwise comparisons of our models on the basis of their AIC, with likelihood ratio tests of nested models.

Model 1	**Model 2**	Δ_{AIC}	χ^2	P($>\chi^2$)
VANILLA	¬INCR_SUR	33	45.35	3.984e−8
¬INCR_SUR	¬INCR_ENT	25	25.16	NA
VANILLA	INCR_SUR	115	127.22	<2.2e−16
INCR_SUR	INCR_ENT	82	82.18	NA
VANILLA	¬INCR_SUR	33	45.35	3.984e−8
¬INCR_SUR	INCR_SUR	82	81.87	NA
VANILLA	¬INCR_ENT	58	70.51	3.207e−13
¬INCR_ENT	INCR_ENT	139	138.88	NA

In table 5, we reported the output of the summary function of the lmerTest package. For each fixed effect, its coefficient indicates both the magnitude of the effect (absolute value) and its direction (sign). The symbols above each value indicate its significance: *** for $p < 0.001$, ** for $p < 0.01$, * for $p < 0.05$, − for $p > 0.05$. Conclusions to be drawn from these tables are that most of our complexity metrics have significant effect on the prediction of the total reading time, except for the 6 grayed-out cells. However, while we expected the effect sign to be positive across all levels (meaning that higher surprisal/entropy leads to more reading time), coefficient of the parameter corresponding to segmentation complexity in models ¬INCR_ENT and ¬INCR_SUR is negative. More work is needed to understand this unexpected result, but we can already note that the coefficients are positive when the metrics are produced by INCR_ENT and INCR_SUR. This is another clue that the INCR strategy (which is obviously more cognitively plausible) is more explainable from the point of view of human behavior than ¬INCR. One way to make more sense of this difference would be to extract from the dataset both the sentences that were most difficult to segment in INCR and ¬INCR, in order to compare them in a qualitative analysis.

In table 6 we reported the output of the anova function of the lmerTest(v.3.1-3) package where each line is a direct comparison of the goodness of fit of a pair of models. The first two columns contain the names of the models being compared, the third column contains the difference in Akaike Information Criterion (AIC) between the models (positive difference indicates that model 2 fits better than model 1) and columns 4&5 includes a χ^2 test indicating that the differences in goodness of fit are indeed significant. By ordering the models by their ability to fit the data (lower AIC) we get:[12] INCR_ENT > ¬INCR_ENT > INCR_SUR > ¬INCR_SUR > VANILLA, we can see that all of the 4 models that use our complexity metrics are better than the baseline model. Therefore we can conclude that our complexity metrics are useful for predicting human reading time. Having observed that INCR_SUR > ¬INCR_SUR and INCR_ENT > ¬INCR_ENT, we can also conclude that our more psychologically motivated model INCR produces better complexity metrics than ¬INCR. Finally, by observing that ¬INCR_ENT > ¬INCR_SUR and INCR_ENT > INCR_SUR, we can conclude that our entropy complexity metric, introduced in section Complexity Metrics overcomes the shortcomings of surprisal and performs better as a predictor.

Conclusion

In this paper we studied the effect of several types of linguistic predictions on reading time data from the Provo corpus. To this end, we developed a versatile modeling framework, called the Reading Machine, that allows to perform an incremental processing of a text starting with the string of characters that corresponds to the text, up to its syntactic parsing. This integrated framework allows to measure and compare the complexity of each of the linguistic tasks involved in this process. The Reading Machine also allows us to implement various processing strategy. Here we focused on two: 1) a standard, NLP oriented, non incremental strategy, which corresponds to a sequential pipeline of modules and, 2) a more psychologically plausible incremental strategy. Our experiments with this new framework showed that various linguistic levels contributed to the prediction of the human reading time, that the measures produced by the incremental strategy were better predictors than those produced by the non incremental strategy, and finally, that — at least in this specific framework — Entropy performed better than Surprisal.

Acknowledgments

This work was granted access to the HPC resources of IDRIS under the allocation 2020-AD011011708 made by GENCI.

References

Alberti, C., Weiss, D., Coppola, G., & Petrov, S. (2015). Improved transition-based parsing and tagging with neural networks. In *Proc. EMNLP* (pp. 1354–1359).

Attneave, F. (1959). *Applications of information theory to psychology: A summary of basic concepts, methods, and results.* Henry Holt.

[12] Symbol ">" reads as "*fits better than*".

Bohnet, B., & Nivre, J. (2012). A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proc. EMNLP* (pp. 1455–1465).

Boston, M. F., Hale, J., Kliegl, R., Patil, U., & Vasishth, S. (2008). Parsing costs as predictors of reading difficulty: An evaluation using the potsdam sentence corpus. *Journal of Eye Movement Research.-ISSN*, *2*(1), 1–12.

Boston, M. F., Hale, J. T., Vasishth, S., & Kliegl, R. (2011). Parallel processing and sentence comprehension difficulty. *Language and Cognitive Processes*, *26*(3), 301–349.

Constant, M., & Nivre, J. (2016). A transition-based system for joint lexical and syntactic analysis. In *Proc. ACL* (Vol. 1, pp. 161–171).

Crabbé, B., Fabre, M., & Pallier, C. (2019). Variable beam search for generative neural parsing and its relevance for the analysis of neuro-imaging signal. In *Proc. EMNLP-IJCNLP* (pp. 1150–1160).

Demberg, V., & Keller, F. (2008). Data from eye-tracking corpora as evidence for theories of syntactic processing complexity. *Cognition*, *109*(2), 193–210.

Frank, S. L., Otten, L. J., Galli, G., & Vigliocco, G. (2013). Word surprisal predicts n400 amplitude during reading. In *Proc. ACL* (pp. 878–883).

Hale, J. (2001). A probabilistic Earley parser as a psycholinguistic model. In *Proc. ACL.*

Hale, J. (2006). Uncertainty about the rest of the sentence. *Cognitive Science*, *30*(4), 643-672.

Hale, J. (2017). Models of human sentence comprehension in computational psycholinguistics. In *Oxford research encyclopedia of linguistics.*

Huang, L., & Sagae, K. (2010). Dynamic programming for linear-time incremental parsing. In *Proc. ACL* (pp. 1077–1086).

Keller, F. (2004). The entropy rate principle as a predictor of processing effort: An evaluation against eye-tracking data. In *Proc. EMNLP* (pp. 317–324).

Luke, S. G., & Christianson, K. (2018). The provo corpus: A large eye-tracking corpus with predictability norms. *Behavior research methods*, *50*(2), 826–833.

Nivre, J. (2003). An efficient algorithm for projective dependency parsing. In *Proc. IWPT* (pp. 149–160).

Shaw, J. A., & Kawahara, S. (2019). Effects of surprisal and entropy on vowel duration in japanese. *Language and speech*, *62*(1), 80–114.

Yamada, H., & Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In *Proceedings of iwpt* (Vol. 3, pp. 195–206).

Zeman, D., et al. (2019). *Universal dependencies 2.5.* (Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University)