Neural heuristics for scaling constructional language processing

Paul Van Eecke[1,2]*, Jens Nevens*[1]*, and Katrien Beuls*[3]
[1] Vrije Universiteit Brussel
[2] KU Leuven
[3] Université de Namur

ABSTRACT

Constructionist approaches to language make use of form-meaning pairings, called constructions, to capture all linguistic knowledge that is necessary for comprehending and producing natural language expressions. Language processing consists then in combining the constructions of a grammar in such a way that they solve a given language comprehension or production problem. Finding such an adequate sequence of constructions constitutes a search problem that is combinatorial in nature and becomes intractable as grammars increase in size. In this paper, we introduce a neural methodology for learning heuristics that substantially optimise the search processes involved in constructional language processing. We validate the methodology in a case study for the CLEVR benchmark dataset. We show that our novel methodology outperforms state-of-the-art techniques in terms of size of the search space and time of computation, most markedly in the production direction. The results reported on in this paper have the potential to overcome the major efficiency obstacle that hinders current efforts in learning large-scale construction grammars, thereby contributing to the development of scalable constructional language processing systems.

Keywords: neuro-symbolic AI, neural heuristics, language processing, computational construction grammar

1 INTRODUCTION

Constructionist approaches to language (Goldberg 2003) analyse all linguistic knowledge that is necessary for language comprehension and production in terms of constructions. A construction is defined as a conventionalised pairing between a linguistic form and its meaning (Goldberg 1995; Kay and Fillmore 1999). There exists no restriction in the nature of the form and the meaning that a construction can capture (Fillmore 1988, p. 36). The form pole of a construction can include morphemes and word forms, as well as larger patterns that range from idiomatic expressions (e.g. "Break a leg!"), over partially instantiated structures (e.g. "X takes Y for granted"), to fully abstract schemata (e.g. the ditransitive "X VERB Y Z" as instantiated in "Simon sent his parents a postcard"). The meaning pole of a construction can contain any semantic or pragmatic information that is associated with a particular form, including lexical and phrasal meaning, the assignment of semantic roles, and the composition of logical structures.

According to the constructionist paradigm, the different constructions that constitute a construction grammar can freely combine in order to collaboratively map between a natural language utterance and a representation of its meaning (Goldberg 2006, p. 22). Due to the unrestricted nature of a construction grammar, the non-locality of constructions, and the fact that the application of a construction does not necessarily correspond to a tree-building operation (van Trijp 2016), constructional language processing cannot straightforwardly be implemented in a faithful way using common techniques such as chart parsing and chart generation (see e.g. Pereira and Warren 1983; Shieber 1988; Kay 1996). Instead, current systems implement the process of finding a sequence of constructions that perform an adequate mapping between a linguistic expression and a representation of its meaning as a search process (Bleys *et al.* 2011; Van Eecke and Beuls 2017). This search process is combinatorial in nature and becomes intractable as grammars increase in size. The intractability of construction grammars is a consequential problem as it constitutes a major obstacle that hinders ongoing research in learning large-scale construction grammars. It thereby limits their usability in both usage-based linguistics research and language technology applications.

Previous approaches to overcoming this intractability problem have either only been partially effective, as in the case of priming networks (Wellens and De Beule 2010; Wellens 2011), or have imposed a global order on constructions, which goes against the constructional idea of "*allowing constructions to combine freely as long as there are no conflicts*" (Goldberg 2006, p. 22), as in the case of construction sets (Beuls 2011).

In this paper, we introduce a novel methodology for learning heuristics that substantially optimise the search processes involved in constructional language processing. The heuristics are based on sequence-to-sequence models that are trained to estimate at any point in processing the probability that the application of a particular construction will lead to a solution. We evaluate the methodology on the CLEVR benchmark dataset (Johnson *et al.* 2017) and show that it outperforms state-of-the-art approaches, both in terms of size of the search space and time of computation.

The remainder of this paper is structured as follows. Section 2 precisely defines the search problem involved in constructional language processing, discusses state-of-the-art approaches, and introduces the dataset and grammar that we will use. Section 3 presents our neural methodology for learning heuristics, which constitutes the main contribution of the paper. Section 4 describes the setup of our experiments and presents the evaluation results. Finally, the method and results are discussed in Section 5. An interactive web demonstration accompanying this paper can be consulted at `https://emergent-languages.org/demos/neural-heuristics`. The web demonstration provides examples of the methodology introduced in this paper in full detail.

PROBLEM DEFINITION 2

We first define constructional language processing as a state-space search problem, which is a class of problems that has a long history in the field of artificial intelligence (Newell and Simon 1956; Nilsson 1971). For doing this, we adopt the terminology that is used in Fluid Construction Grammar (FCG – `https://www.fcg-net.org`) (Steels

2011; van Trijp *et al.* 2022; Beuls and Van Eecke 2023), the leading computational construction grammar implementation. We then discuss the merits and limitations of state-of-the-art approaches, in particular the use of priming networks and the use of construction sets. Finally, we introduce the CLEVR dataset, which will be used as a benchmark to evaluate our methodology against the state of the art in Section 4.

2.1 *Constructional language processing*

Constructional language processing is the process in which the different constructions of a construction grammar combine in order to comprehend or produce natural language expressions. Comprehension refers to the process of mapping a natural language expression to a representation of its meaning, while production refers to the inverse process of mapping a semantic representation to a natural language utterance. Both processes are performed by the same grammar, i.e. the same inventory of constructions. Constructional language processing, as operationalised in the FCG framework, revolves around two basic concepts: 'transient structures' and 'constructions'.

- **Transient structures** A transient structure is a feature structure that represents all that is known about a linguistic expression at a given point during processing. Transient structures correspond to state representations in the classical problem solving paradigm. Before processing has started, the transient structure, which is at that point called 'initial transient structure', only contains the input to the comprehension or production process. In comprehension, the input consists of an utterance; while in production, it consists of a semantic representation.
- **Constructions** A construction (CXN) is a feature structure that represents a bidirectional mapping between the formal and the semantic aspects of a linguistic entity. Constructions correspond to operators in the problem solving paradigm and consist of preconditions and postconditions. The preconditions can be 'matched' against a transient structure and if matching succeeds, the postconditions can be 'merged' into the transient structure. Matching

is a first-order syntactic unification operation that checks the compatibility of two feature structures, whereas merging is a unification operation that combines the information contained in two feature structures. For a formal definition of matching and merging, see Steels and De Beule (2006) and Sierra Santibáñez (2012).

Constructional language processing consists in the sequential application of constructions to a transient structure. Each individual construction application thereby expands the transient structure with new information. Initially, the transient structure only contains the input utterance or input meaning representation, and only constructions that match this information can apply. Through their application, these constructions can contribute additional information to the transient structure, which can in turn satisfy the preconditions of other constructions. Analogous to the use of goal tests in the classical problem solving paradigm, goal tests in constructional language processing verify whether a given transient structure qualifies as a solution to the search problem. Typical goal tests for constructional language processing include (i) checking whether no more constructions can apply, (ii) verifying whether the input utterance or input meaning representation has been fully processed, and (iii) checking whether the meaning comprehended so far consists of a fully connected network of predicates linked through their arguments. When all goal tests succeed for a given transient structure, it qualifies as a solution and the resulting meaning representation (in comprehension) or the resulting utterance (in production) are extracted.

An illustrative example of a construction application process is shown in Figure 1. Note that the constructions used in this example were created for didactic purposes, and do not necessarily correspond to insightful linguistic analyses. From left to right, the figure shows the transient structures and constructions involved in the processing of the utterance *Sam cycles* in comprehension and production. The transient structures shown in the top-left and bottom-left corners (i.e. the green boxes labelled with the number 1) are the initial transient structures in comprehension and production respectively. The initial transient structure in comprehension contains an input unit with a number of predicates representing the utterance. The initial transient structure in production contains an input unit with the meaning representation of

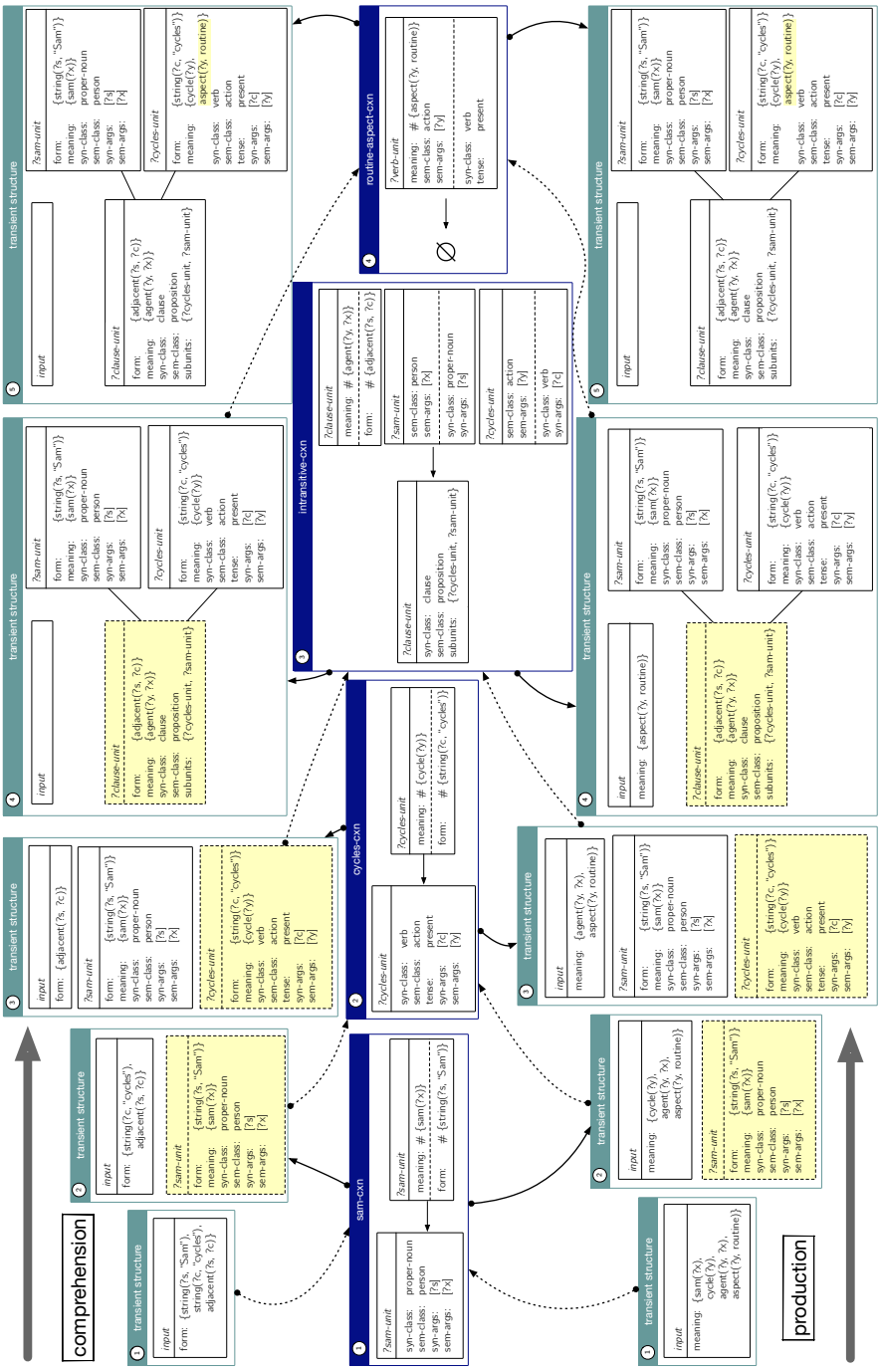


Figure 1: Schematic representation of how different constructions combine in order to comprehend and produce a natural language expression. Here, four constructions collaboratively process the utterance *Sam cycles* in comprehension (top, left-to-right) and production (bottom, left-to-right). Dotted arrows represent matching relations and solid arrows represent merging relations. New units added during the merging phase are highlighted using dashed boxes. Symbols preceded by a question mark denote logic variables

the utterance in predicate notation. The middle-left box in the figure represents the SAM-CXN, which matches both initial transient structures. Conventionally, the preconditions and postconditions of a construction are separated by a left-pointing arrow. The preconditions are written on the right-hand side of the arrow, while the postconditions are specified on its left-hand side. As constructions support language processing in both the comprehension and production direction, they contain two sets of preconditions on their right-hand side. The preconditions that are active in comprehension are always specified under a dashed line and the preconditions that are active in production are specified above it. The preconditions of one direction become postconditions in the opposite direction, and are as such treated in the same way as the information specified on the left-hand side. In this case, the construction matches the string *Sam* in comprehension and adds the meaning predicate above the dashed line along with the semantic and syntactic features specified on the left-hand side. The resulting transient structure (labelled with the number 2) is shown just right of the initial transient structure. In production, an analogous process takes place. Here, the construction matches a meaning predicate that is present in the initial transient structure, and contributes a string predicate along with the same semantic and syntactic features as in comprehension.

Next, the CYCLES-CXN applies in the same way to the transient structure that was just created, adding new information related to the string *cycles* in comprehension and to the predicate *cycle*(?y) in production. After that, the INTRANSITIVE-CXN (labelled with the number 3) can apply, as its preconditions are now satisfied by information from the input unit, in combination with information that was contributed by the SAM-CXN and the CYCLES-CXN. The INTRANSITIVE-CXN maps between the adjacency of a proper noun and a verb, and the agentive relation between the person and action they represent. Finally, the ROUTINE-ASPECT-CXN (labelled with the number 4) maps between an action verb in the present tense and a meaning predicate denoting that the aspectual structure of the action corresponds to a routine.

From the final transient structure, shown in the top-right and bottom-right corners of the figure and labelled with the number 5, the result of the construction application process can be extracted.

In comprehension, this is the combination of all 'meaning' features in the transient structure, while in production, it is the combination of all 'form' features. Note that the construction application process is entirely bidirectional. The output in comprehension is equal to the input in production and vice versa. Moreover, the exact same set of constructions has been applied, in this case even in the same sequential order.

This illustrative example shows how constructions can collaboratively map between an utterance and a representation of its meaning, both in the comprehension and the production direction, with examples of constructions that can apply based on the input only (SAM-CXN and CYCLES-CXN), constructions that build hierarchical structures (INTRANSITIVE-CXN) and constructions that only contribute non-hierarchical information (ROUTINE-ASPECT-CXN). What the example doesn't show is how a constructional language processing engine can determine that it is exactly this combination of constructions that needs to apply. Construction grammars that exceed the size of these toy examples are immediately faced with constructions that are in competition with each other, and in particular with sequences of constructions that can apply but do not ultimately lead to a solution. This challenge, which is central to the problem solving paradigm, can be solved by backtracking to earlier transient structures in case of failure and possibly, in the worst case, exploring the entire search space, i.e. trying out all possible combinations of construction applications. It is this process of construction application and backtracking that makes constructional language processing intractable for larger grammars.

As can be seen in the figure, the constructions that constitute construction grammars differ in many aspects from the rules that constitute traditional formal grammars. First of all, constructions do not necessarily correspond to tree-building operations, as exemplified by the ROUTINE-ASPECT-CXN and discussed in van Trijp (2016). Constructions are also non-local, in the sense that they can match information that is present anywhere in the transient structure. As a consequence, constructional language processing cannot straightforwardly be optimised using well-known techniques for efficiently processing formal grammars, such as chart parsing and chart generation (see e.g. Pereira and Warren 1983; Shieber 1988; Kay 1996).

State-of-the-art approaches 2.2

In the computational construction grammar literature, a number of techniques for reducing the search space created by all possible construction applications have been proposed. A straightforward optimisation that is almost always used consists in checking whether a new transient structure is different from all other transient structures that already occur in the search tree. If this is not the case, the duplicate transient structure can immediately be pruned away. A second common optimisation consists in hashing constructions that match string predicates or meaning predicates in the input unit, which reduces the search problem to abstract constructions only.

When it comes to the choice of the baseline search strategy, a depth-first search algorithm with backtracking is often chosen. For constructional language processing, depth-first search generally outperforms breadth-first search for two reasons. First, solutions are typically found deep in the search tree (after many constructions have been applied) and there is no inherent preference for shorter solutions, like for example in the case of planning problems. Second, there often exist many correct orders in which constructions can apply, which can lead to a high branching factor and an abundance of duplicate transient structures, some of which can only be detected deep in the search tree.

Two more advanced approaches that go beyond the depth-first search with backtracking, duplicate detection and hashing baseline have been proposed in the literature: 'construction sets' and 'priming networks'.

- **Construction sets** This approach consists in subdividing the construction inventory into (possibly overlapping) sets of constructions. Two global orders of construction sets are specified, one for comprehension and one for production. The basic idea is that constructions of a later set are not applied before constructions of an earlier set have at least been matched against the transient structure (Beuls 2011). The use of construction sets can drastically decrease the size of the search space, but comes with a number of important drawbacks. Construction sets are inherently in disagreement with the constructionist idea that constructions can

freely combine as long as there are no conflicts, which is crucial for supporting open-ended and creative language use (Van Eecke and Beuls 2018; Goldberg 2006, p. 22). Also, the global ordering of construction sets and the allocation of constructions to particular sets is difficult to learn, as it presupposes that the general architecture of a grammar is known beforehand. Finally, scaling grammars that make use of construction sets is even difficult in the case of hand-crafted grammars, as the grammar engineer then does not only need to encode the necessary linguistic knowledge, but also needs to determine the order in which constructions need to be scheduled for matching.

- **Priming networks** Priming networks are inspired by the psychological phenomenon whereby current behaviour is nonconsciously influenced by exposure to past experiences (see e.g. Schacter and Buckner 1998). In the case of computational construction grammar, this approach argues that the application of a construction can prime the application of another construction. In this way, frequent co-occurrences of constructions can be captured in the form of a priming network (Wellens and De Beule 2010; Wellens 2011). Priming links can be learned in a usage-based fashion by extracting the frequency of co-occurrences of constructions from successful branches of the search trees generated during past construction application processes. These links can then be used to guide the search process by always expanding the transient structure created by the construction that was most strongly primed. The priming links are based either on the order of the constructions themselves or on dependencies between a construction's preconditions and the postconditions of other constructions by which they were satisfied. Two priming networks are learned for a construction inventory, one for use in comprehension and the other for use in production. The main advantage of priming networks is that they can be learned in a straightforward way. However, an important disadvantage is that if only local priming links are taken into account, priming is only partially effective, and if longer-distance links are taken into account, the networks are often not efficacious as they suffer from sparsity problems.

Another solution that has been proposed in the literature is to process construction grammars using existing systems for implementing generative grammar formalisms (Müller 2017). A major disadvantage of this approach is that this is only possible for generative grammar formalisms that include constructional properties (e.g. constructional HPSG and SBCG). These formalisms are inherently limited to local constructions that correspond to tree-building operations (van Trijp 2016; van Trijp *et al.* 2022). As a consequence, this approach does not satisfy the methodological needs of the construction grammar community at large.

The CLEVR dataset and grammar 2.3

We use the CLEVR dataset (Johnson *et al.* 2017) and CLEVR construction grammar (Nevens *et al.* 2019) to benchmark the effect of the heuristics that we propose in this paper. This choice is motivated by three main reasons. First of all, with its nearly 1,000,000 utterances, the CLEVR dataset is sufficiently large to train even the most data-intensive heuristics. Second, there exists a computational construction grammar that, given infinite computation time, covers the entire dataset, in both the comprehension and production direction (Nevens *et al.* 2019). This means that this grammar achieves 100% accuracy on the tasks of mapping from utterances to their meaning representation and vice versa. This allows us to evaluate the effect of the proposed heuristics in isolation. Finally, the grammar gives rise to a search space that can only be processed efficiently using powerful heuristics.

The utterances in the CLEVR dataset are synthetically generated English questions about images of scenes depicting different configurations of geometrical figures. Each question is annotated with a semantic representation that captures the logical meaning that underlies it. The question-annotation pairs embrace various aspects of reasoning, including attribute identification (*There is a large cube; what is its color?*), counting (*How many green spheres are there?*), comparison (*Are there an equal number of large cubes and small things?*), spatial relationships (*What size is the cylinder that is right of the yellow shiny thing*

that is left of the cube?) and logical operations (*How many objects are either red cubes or yellow cylinders?*). The average length of the questions is 18.4 words with a maximum length of 42 words.

The CLEVR grammar consists of 170 constructions, of which 55 are morphological and lexical constructions. Apart from these, the grammar also contains 115 grammatical constructions that capture phenomena including referential expressions, spatial relations, coordination and subordination structures, and a wide range of interrogative structures. On average, 25 constructions should be applied in order to successfully comprehend or produce an utterance from the dataset. This means that the average solution is found at depth 25 in the search tree.

The size of the search space for an average sentence amounts thus in theory to 170^{25} construction applications. In practice, most of these construction applications are not possible given the dependencies between the preconditions and postconditions of the constructions. Still, when using the baseline depth-first strategy with backtracking, duplicate detection and hashing, the search tree in comprehension includes on average more than 3.5 times the number of construction applications than were needed to find a solution. While this might still be manageable to a certain extent, this number grows to more than 29 in production. In practice, this means that many solutions cannot be found in a reasonable amount of time without the use of suitable heuristics.

We make use of the same splits as the original dataset, with the training, validation and test sets consisting of 699,989 utterances, 149,991 utterances and 149,988 utterances respectively.

3 METHODOLOGY

We will now introduce our novel methodology for learning heuristics that substantially optimise the search processes involved in constructional language processing. These heuristics take the form of neural networks that are trained to estimate at any point in processing the probability that the application of a particular construction will lead to a solution. Our approach is inspired by recent successes obtained using

neural heuristics in other domains that typically employ the problem solving paradigm, in particular games (Mnih *et al.* 2015; Silver *et al.* 2016) and planning (Takahashi *et al.* 2019; Wang *et al.* 2019; Ferber *et al.* 2020).

General architecture 3.1

We design neural heuristics that can be used to assign after each construction application a score to the resulting transient structure. This score, called 'heuristic value', reflects how close a transient structure is to a solution. It can then be used to decide on the order in which transient structures are expanded, with the goal of minimizing the average number of construction applications that is needed to reach a solution. Intuitively, this score is influenced by both the input utterance (in comprehension) or meaning representation (in production) and the sequence of constructions that have been applied so far in the same branch of the search tree.

For each direction of processing, this intuitive idea is operationalised using two recurrent neural networks (RNNs) that are organised in an encoder-decoder constellation. Before processing starts, the encoder RNN encodes the input utterance or meaning representation into a context vector. During processing, the decoder RNN is called for each transient structure, just before its expansion. The input to the decoder RNN is the sequence of names of constructions that have been applied so far in that branch of the search tree, along with the output of the encoder RNN (context vector) and its hidden states. The output of the decoder RNN is at each decoding timestep a probability distribution over all constructions in the construction inventory. The constructions are then applied and the heuristic values of the resulting transient structures are computed as the sum of the heuristic value of their parent transient structure and the probability score returned by the decoder RNN. The heuristic values of the transient structures are used in combination with a beam search algorithm. This process is graphically depicted in Figure 2, where the beam size is set to three for clarity reasons.

The choice for an RNN-based encoder-decoder architecture is motivated by two main reasons. First of all, the problem of mapping

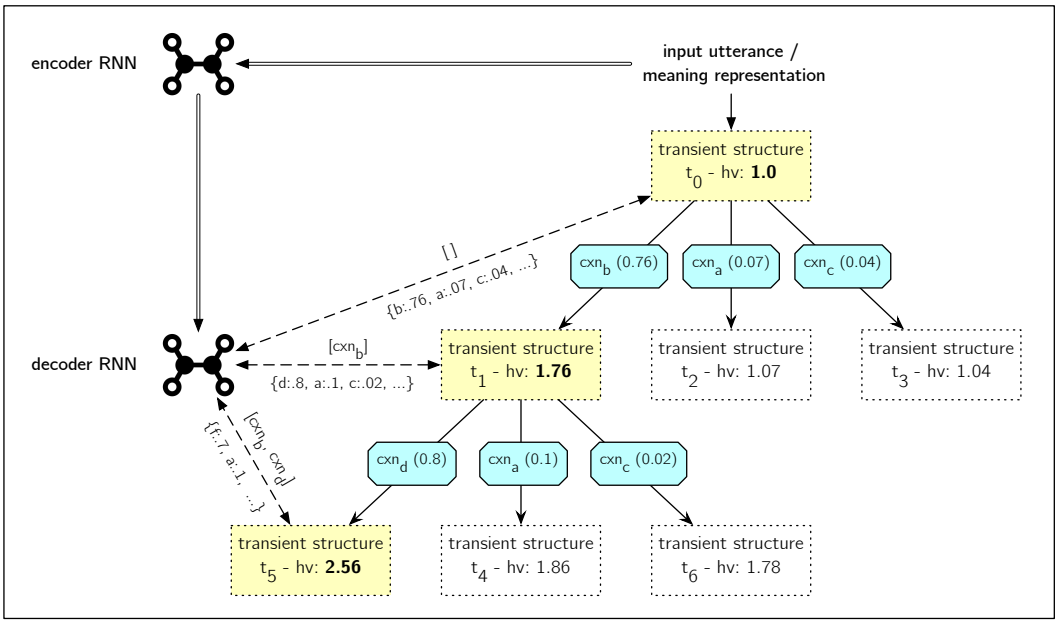


Figure 2: Schematic representation of the integration of the neural heuristics in constructional language processing. Before a node in the search tree is expanded, the encoder-decoder model is queried, and a probability distribution over all constructions of the construction inventory is returned. The heuristic values are then calculated and used by a beam search algorithm, in this case with the beam size set to three. 'Hv' stands for 'heuristic value'

an input utterance or meaning representation to a sequence of constructions can be naturally framed as a sequence-to-sequence problem. RNN-based architectures are typically good at handling this class of problems (Sutskever *et al.* 2014), although also CNN-based (Gehring *et al.* 2017) and transformer-based (Vaswani *et al.* 2017) architectures have more recently been successfully applied to the same class of problems. Second, and most importantly, the sequential nature of the RNN-based architecture allows us to query the decoder RNN while already providing a partial sequence of predictions. This is necessary for integrating the neural architecture as a heuristic in the construction application process, while being able to keep the benefits of the existing search and backtracking facilities. Indeed, the neural networks are used to make the search process created by the grammar more efficient, unlike their use in end-to-end neural semantic parsers, where

they perform the actual mapping from utterances to their semantic representation (see e.g. Jia and Liang 2016; Konstas *et al.* 2017; van Noord *et al.* 2018; Yu and Gildea 2022).

Our neural encoder-decoder architecture is based on the neural machine translation architecture proposed by Bahdanau *et al.* (2015). It consists of an encoder with bidirectional single-layer gated recurrent units (GRUs), a decoder with single-layer GRUs and an attention mechanism that attends over the encoder's hidden states at every decoder time step. The attention mechanism ensures that the decoder does not need to rely on a single high-dimensional representation of the entire input sequence (the context vector). Instead, the decoder has access to all encoder hidden states and learns to use a subset of these hidden states. Intuitively, the decoder chooses at every timestep to pay attention to specific parts of the input utterance or meaning representation.

The basic idea underlying our methodology is somewhat reminiscent of the use of recurrent neural networks for guiding dependency parsing (Kiperwasser and Goldberg 2016; Dozat and Manning 2017, 2018). In this line of research, RNNs are also used to predict sequences of actions (e.g. transitions) based on utterances and previous actions. The main difference resides in the correspondence between the length of the input and the length of the predictions. In the case of dependency parsing, an action needs to predicted for each input word. In the case of constructional language processing, however, the number of constructions that needs to be predicted is not tied to the length of the input utterance or meaning representation. Indeed, multiple words or meaning predicates (even organised in non-contiguous patterns) can be covered by the application of a single construction, and single words or meaning predicates can give rise to the application of multiple constructions (see e.g. the ROUTINE-ASPECT-CXN in Figure 1). In order to accommodate for this asymmetry between the length of the input pattern and the length of the output pattern, we have opted for two RNNs in an encoder-decoder constellation instead of directly using an RNN for prediction. This allows us to effectively decouple the length of both sequences.

3.2 *Training*

Training the neural encoder-decoder architecture requires a dataset of input utterances (in comprehension) or meaning representations (in production), paired with, for each utterance or meaning representation, a sequence of names of constructions of which the application would lead to a solution. Annotating the original CLEVR dataset, which contains utterances along with a representation of their meaning, in this format is a non-trivial task, as we face at this moment the very search problem that we are aiming to optimise. We therefore adopted a spiral approach. For both comprehension and production, we started processing the data from CLEVR's training and validation splits using the depth-first search strategy with backtracking, duplicate detection and hashing, setting a time limit of 400 seconds. We collected the sequences of construction names for all input utterances or meaning representations that were successfully processed within this time frame. Then, we trained a first version of the sequence-to-sequence heuristic and used it to process more utterances using the same time limit. After three iterations, the entire dataset could be successfully annotated.

The encoder-decoder architecture requires that input utterances or meaning representations are represented as sequences. For utterances, this is naturally done by using sequences of tokens. For meaning representations, this is somewhat more complicated as they come in the form of networks of predicates that share variables. We therefore transformed the predicate networks into sequences notated in reverse Polish notation. In this notation, predicate names follow their arguments. Since the arity of each predicate is known, the notation is unambiguous without the need for variables and their equalities to be explicitly represented.

We trained the encoder-decoder models for 100,000 time steps with a batch size of 64, using the Adam optimisation algorithm with a learning rate of 5e-4 and weight decay of 1e-6. We used cross-entropy as the loss function and used a teacher forcing ratio of 1. We included a dropout layer after the embedding layer in both the encoder and the decoder. We ran a hyperparameter optimisation process for the embedding size (100, 200, 300), the hidden layer size (64, 128, 256, 512) and the dropout probability (0.0, 0.1, 0.2, 0.5). We found that

best performance was achieved using the model with an embedding size of 100 in comprehension and 300 in production, a hidden layer size of 512 in comprehension and 256 in production and a dropout probability of 0.2 in comprehension and 0.1 in production. Note that for efficiency reasons, the hyperparameters were optimised based on the gold standard annotation of the dataset, and not based on their performance as a heuristic in FCG.

EXPERIMENTS 4

In order to benchmark the efficiency of our methodology and compare it against the state of the art, we conducted two experiments that evaluate the use of the proposed neural heuristics in constructional language processing. The first experiment is concerned with the comprehension direction, while the second experiment is concerned with the production direction.

Experimental setup 4.1

Both experiments consist in processing the test split of the CLEVR dataset using three different search strategies. The first strategy makes use of FCG's standard search algorithm, namely depth-first search with backtracking, duplicate detection and hashing. The second strategy makes use of priming networks as proposed by Wellens and De Beule (2010). The third strategy evaluates the encoder-decoder methodology that we introduced above, with an unrestricted beam size.

The strategies are evaluated in terms of the size of the search space and the time that is required to reach a solution. The size of the search space is defined as the total number of transient structures that were created during processing, divided by the number of transient structures in the branch of the solution. The optimal size of the search space is thus equal to one, indicating that a solution was found without any backtracking taking place. The time of computation is measured in seconds, spanning from the creation of the initial transient structure

until the resulting meaning representation (in comprehension) or utterance (in production) has been extracted from the solution transient structure. In general, the size of the search space is the most accurate measure for gauging the performance of a search strategy, but it does not take into account the computational overhead caused by the heuristic itself. The computation time metric includes both factors, but should be interpreted with extreme caution, as it is also influenced by external factors.

For the purposes of this paper, we have chosen to focus on the fundamental issue of reducing the search space, and have not included any other time-related optimisations. Such optimisations could include the implementation of a more efficient protocol for communication between the FCG engine and the neural networks, deploying the neural networks on GPUs, or not using the neural heuristics for utterances under a maximum number of words. The reason that we include the computation time metric is to show that even without these optimisations, a reduction in the search space already corresponds to a reduction in processing time.

If no solution was found within 400 seconds, the search process was halted and the result was logged as 'no solution found'.

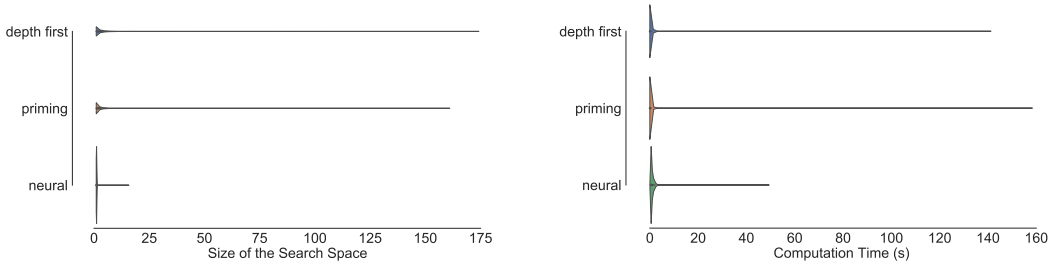
The evaluation was carried out using computing nodes with 2×20-core Intel Xeon Gold 6148 (Skylake) CPUs and 16GB of RAM.

4.2 *Experimental results: comprehension*

The results of the comprehension experiment are presented in Table 1 and visualised through violin plots in Figure 3. The table provides the mean values, standard deviation and maximum values of the search space size and the computation time for the depth-first, priming and neural strategies. The plots show the probabilistic density of the search space size (Figure 3a) and computation time (Figure 3b) for the three strategies. When it comes to the size of the search space, the results show that the neural strategy greatly outperforms the depth-first and priming strategies. More density mass is situated close to a search space size of one, which is the theoretical minimum. The average size of the search space is 1.16 in the case of the neural strategy, 3.21 in the case of the priming strategy and 3.69 in the case of the depth-first strategy. Importantly, the performance gain obtained through the

Table 1: Performance of the different strategies in the comprehension direction

	Search space size			Computation time (s)			# Timed out
	mean	sd	max	mean	sd	max	> 400 s
Depth-first	3.69	7.09	174.26	0.84	4.42	141.28	0
Priming	3.21	5.98	161.09	0.72	3.73	158.48	0
Neural	1.16	0.19	15.84	0.91	0.80	49.48	0



(a) Search space size per strategy

(b) Computation time per strategy

Figure 3: Visualisation of the results of the comprehension experiment

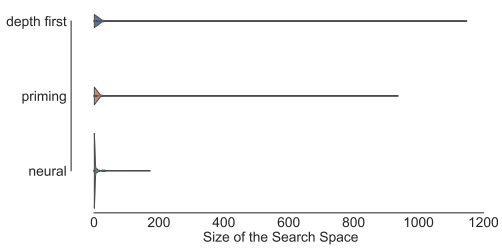
neural strategy also extends to sentences that otherwise require a large search space. The largest search space required by the neural strategy is 15.82, while the depth-first and priming strategies require search space sizes of up to 174.26 and 161.09, respectively. The results obtained through the computation time metric are in line with those obtained through the search space size metric. Even the most difficult sentences take less than 50 seconds using the neural strategy, whereas they take more than 140 seconds using the depth-first and priming strategies. In sum, we can conclude from the comprehension experiment that the neural strategy outperforms the state of the art both in terms of size of the search space and in terms of time of computation. Importantly, the greatest reduction in search space and time of computation is achieved for the most difficult sentences.

4.3 *Experimental results: production*

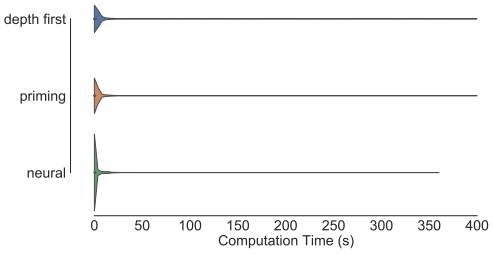
The results of the production experiment are presented in Table 2 and visualised through violin plots in Figure 4. Figure 4a shows the

Table 2: Performance of the different strategies in the production direction

	Search space size			Computation time (s)			# Timed out
	mean	sd	max	mean	sd	max	(> 400 s)
Depth-first	29.08	90.40	1149.74	8.84	37.57	400.00	1325
Priming	20.81	67.78	938.25	6.40	29.38	400.00	475
Neural	6.35	16.85	173.90	3.64	12.32	360.25	0



(a) Search space size per strategy



(b) Computation time per strategy

Figure 4: Visualisation of the results of the production experiment

average size of the search space for each strategy. We can immediately observe that the search problem in production is considerably more difficult than the search problem in comprehension, and that the performance gain that is obtained through the neural strategy is even larger. In the case of the neural strategy, the density mass is concentrated around a lower mean value (6.35) than in the case of the priming (20.81) and depth-first (29.08) strategies. The maximum value is reduced from 1149.74 (depth-first) and 938.25 (priming) to 173.90 (neural). When it comes to computation time (Figure 4b), the results are analogous. The average processing time is reduced from 8.84 (depth-first) and 6.40 (priming) seconds to 3.64 seconds (neural). The maximum processing time that was needed amounts to 360.25 seconds for the neural strategy. For the other two strategies, not all sentences could be produced within the maximum time frame of 400 seconds.

An analysis of the utterances for which the neural strategy could not reduce the search space to under 5 reveals an interesting limitation of the methodology that we have introduced. The decoder RNN takes as input a sequence of constructions that have so far been applied

during the application process and returns as output a probability distribution over all constructions in the construction inventory. In other terms, it makes predictions about which constructions should be applied at which moment in time. However, it does not make any predictions about the way in which the constructions should apply, in particular to which units in the transient structure. As a consequence, only the ambiguity that arises from multiple applicable constructions (i.e. multiple transient structures resulting from the application of different constructions) can be solved, not the ambiguity that arises from multiple ways in which a single construction can apply (i.e. multiple transient structures resulting from the application of a single construction). While this ambiguity is far less substantial than the ambiguity that stems from multiple applicable constructions, it explains why the search space is not consistently reduced to around 1 even if every prediction by the neural network is optimal.

In sum, we can conclude that the production experiment confirms the results obtained in the comprehension experiment. The neural strategy outperforms the state of the art both in terms of size of the search space and in terms of time of computation, especially when it comes to processing the most difficult sentences of the dataset.

DISCUSSION AND CONCLUSIONS 5

Constructionist approaches to language, as originally laid out by, among others, Fillmore (1988), Goldberg (1995), Kay and Fillmore (1999) and Croft and Cruse (2004), consider form-meaning mappings, called constructions, to be the basic unit of linguistic analysis. Apart from the fact that they constitute form-meaning mappings, constructions are subject to very few restrictions. First of all, constructions do not necessarily correspond to tree-building operations (van Trijp 2016). Second, constructions are non-local in the sense that they can access all information that is known during processing. Third, constructions can involve units of arbitrary size, both on the form and the meaning side. Finally, constructions are not restricted to continuous constituents and are not even required to include word order constraints. As a consequence, constructional language processing cannot

straightforwardly be implemented in a faithful way using common grammar processing techniques, such as chart parsing and generation (see e.g. Pereira and Warren 1983; Shieber 1988; Kay 1996). Instead, faithful computational construction grammar implementations implement constructional language processing as a state-space search problem (Bleys *et al.* 2011; Van Eecke and Beuls 2017).

In order to reliably scale to large problems, state-space search methods rely on heuristics that can estimate the likelihood that a given state will lead to a solution. While certain optimisations have in the past been applied to the case of computational construction grammar, including construction sets (Beuls 2011) and priming networks (Wellens and De Beule 2010; Wellens 2011), a lack of general and powerful heuristics remained a major obstacle to ongoing construction grammar research, in particular to research on representing, processing and learning large-scale construction grammars.

The neural methodology that we have presented in this paper introduces a general and effective way to learn heuristics that substantially optimise the search processes involved in constructional language processing. Analogous to recent successes in many subfields of artificial intelligence, including game playing (Mnih *et al.* 2015; Silver *et al.* 2016) and planning (Takahashi *et al.* 2019; Wang *et al.* 2019; Ferber *et al.* 2020), the methodology combines the predictive strengths of neural networks with the expressive representations, sound logic operations and backtracking abilities of traditional search and unification methods.

An integration of the proposed method in the Fluid Construction Grammar system (Steels 2011; van Trijp *et al.* 2022; Beuls and Van Eecke 2023) and an evaluation of the method using the CLEVR benchmark dataset (Johnson *et al.* 2017) and the CLEVR construction grammar (Nevens *et al.* 2019) show that the neural heuristics indeed outperform the state-of-the-art priming strategy and can substantially reduce the search space and processing time in both the comprehension and the production direction, especially in the case of utterances that otherwise gave rise to a large search space.

We posit that this general methodology for learning neural heuristics that optimise the search processes involved in constructional language processing constitutes a promising contribution towards the scaling of constructionist approaches to language. It thereby has both

theoretical and practical implications. On the theoretical side, scalable processing models will allow construction grammarians to go beyond the study of constructions in isolation, and model the intricate interactions that take place between constructions as part of a larger grammar. On the practical side, the scaling of constructional language processing paves the way for achieving breakthroughs in ongoing research on learning large-scale construction grammars (Nevens *et al.* 2022; Doumen *et al.* 2023), which has in turn major implications on research in usage-based linguistics (Diessel 2015), models of language acquisition (Tomasello 2003) and the use of construction grammar in language technology applications (Willaert *et al.* 2020, 2021; Beuls *et al.* 2021; Verheyen *et al.* 2022).

ACKNOWLEDGEMENTS

We would like to thank the three anonymous JLM reviewers for their rigorous yet constructive examination of our paper. The research reported on in this paper was financed by the Research Foundation Flanders (FWO) through a postdoctoral grant awarded to Paul Van Eecke (75929) and the European Union's Horizon 2020 research and innovation programme under grant agreement number 951846 (Meaning and Understanding in Human-centric AI – `https://www.muhai.org`).

CONFLICT OF INTEREST STATEMENT

On behalf of all authors, the corresponding author states that there is no conflict of interest.

REFERENCES

Dzmitry BAHDANAU, Kyunghyun CHO, and Yoshua BENGIO (2015), Neural machine translation by jointly learning to align and translate, in *International Conference on Learning Representations (ICLR 2015)*, pp. 1–15.

Katrien BEULS (2011), Construction sets and unmarked forms: A case study for Hungarian verbal agreement, in Luc STEELS, editor, *Design Patterns in Fluid Construction Grammar*, pp. 237–264, John Benjamins, Amsterdam, Netherlands.

Katrien Beuls and Paul Van Eecke (2023), Fluid Construction Grammar: State of the art and future outlook, in *Proceedings of the First International Workshop on Construction Grammars and NLP (CxGs + NLP, GURT/SyntaxFest 2023)*, pp. 41–50, Association for Computational Linguistics, Washington, D.C., USA.

Katrien Beuls, Paul Van Eecke, and Vanja Sophie Cangalovic (2021), A computational construction grammar approach to semantic frame extraction, *Linguistics Vanguard*, 7(1):20180015.

Joris Bleys, Kevin Stadler, and Joachim De Beule (2011), Search in linguistic processing, in Luc Steels, editor, *Design Patterns in Fluid Construction Grammar*, pp. 149–179, John Benjamins, Amsterdam, Netherlands.

William Croft and D. Alan Cruse (2004), *Cognitive linguistics*, Cambridge University Press, Cambridge, United Kingdom.

Holger Diessel (2015), Usage-based construction grammar, in Ewa Dąbrowska and Dagmar Divjak, editors, *Handbook of Cognitive Linguistics*, pp. 295–321, Mouton de Gruyter, Berlin, Germany.

Jonas Doumen, Katrien Beuls, and Paul Van Eecke (2023), Modelling language acquisition through syntactico-semantic pattern finding, in *Findings of the Association for Computational Linguistics: EACL 2023*, forthcoming.

Timothy Dozat and Christopher D. Manning (2017), Deep biaffine attention for neural dependency parsing, in *5th International Conference on Learning Representations, ICLR 2017*, pp. 1–8.

Timothy Dozat and Christopher D. Manning (2018), Simpler but more accurate semantic dependency parsing, in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 484–490, Association for Computational Linguistics, Melbourne, Australia, doi:10.18653/v1/P18-2077.

Patrick Ferber, Malte Helmert, and Jörg Hoffmann (2020), Neural network heuristics for classical planning: A study of hyperparameter space, in *24th European Conference on Artificial Intelligence*, pp. 2346–2353.

Charles J. Fillmore (1988), The mechanisms of "construction grammar", in *Annual Meeting of the Berkeley Linguistics Society*, volume 14, pp. 35–55.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin (2017), Convolutional sequence to sequence learning, in *Proceedings of the 34th International Conference on Machine Learning*, pp. 1243–1252.

Adele E. Goldberg (1995), *Constructions: A construction grammar approach to argument structure*, University of Chicago Press, Chicago, IL, USA.

Adele E. Goldberg (2003), Constructions: A new theoretical approach to language, *Trends in Cognitive Sciences*, 7(5):219–224.

Adele E. Goldberg (2006), *Constructions at work: The nature of generalization in language*, Oxford University Press, Oxford, United Kingdom.

Robin Jia and Percy Liang (2016), Data recombination for neural semantic parsing, in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12–22, Association for Computational Linguistics, Berlin, Germany, doi:10.18653/v1/P16-1002.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick (2017), CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning, in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2901–2910, IEEE Computer Society, Los Alamitos, CA, USA.

Martin Kay (1996), Chart generation, in *34th Annual Meeting of the Association for Computational Linguistics*, pp. 200–204, Association for Computational Linguistics.

Paul Kay and Charles Fillmore (1999), Grammatical constructions and linguistic generalizations: The what's x doing y? construction, *Language*, 75(1):1–33.

Eliyahu Kiperwasser and Yoa Goldberg (2016), Simple and accurate dependency parsing using bidirectional LSTM feature representations, *Transactions of the Association for Computational Linguistics*, 4:313–327, doi:10.1162/tacl_a_00101, https://aclanthology.org/Q16-1023.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer (2017), Neural AMR: Sequence-to-sequence models for parsing and generation, in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 146–157, Association for Computational Linguistics, Vancouver, Canada, doi:10.18653/v1/P17-1014.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, *et al.* (2015), Human-level control through deep reinforcement learning, *Nature*, 518(7540):529–533.

Stefan Müller (2017), Head-Driven Phrase Structure Grammar, Sign-Based Construction Grammar, and Fluid Construction Grammar: Commonalities and differences, *Constructions and Frames*, 9(1):139–173.

Jens Nevens, Jonas Doumen, Paul Van Eecke, and Katrien Beuls (2022), Language acquisition through intention reading and pattern finding, in *Proceedings of the 29th International Conference on Computational Linguistics*, pp. 15–25, International Committee on Computational Linguistics, Gyeongju, Republic of Korea, https://aclanthology.org/2022.coling-1.2.

Jens Nevens, Paul Van Eecke, and Katrien Beuls (2019), Computational construction grammar for visual question answering, *Linguistics Vanguard*, 5(1):20180070.

Allen Newell and Herbert Simon (1956), The logic theory machine – a complex information processing system, *IRE Transactions on Information Theory*, 2(3):61–79.

Nils Nilsson (1971), *Problem-solving methods in artificial intelligence*, McGraw-Hill Book Company, New York, NY, USA.

Fernando Pereira and David Warren (1983), Parsing as deduction, in *21st Annual Meeting of the Association for Computational Linguistics*, pp. 137–144, Association for Computational Linguistics.

Daniel L. Schacter and Randy L. Buckner (1998), Priming and the brain, *Neuron*, 20(2):185–195.

Stuart M. Shieber (1988), A uniform architecture for parsing and generation, in *Coling Budapest 1988 Volume 2: International Conference on Computational Linguistics*, pp. 614–619, https://aclanthology.org/C88-2128.

Josefina Sierra Santibáñez (2012), A logic programming approach to parsing and production in Fluid Construction Grammar, in Luc Steels, editor, *Computational Issues in Fluid Construction Grammar*, volume 7249 of *Lecture Notes in Computer Science*, pp. 239–255, Springer, Berlin, Germany.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, *et al.* (2016), Mastering the game of Go with deep neural networks and tree search, *Nature*, 529(7587):484–489.

Luc Steels, editor (2011), *Design patterns in Fluid Construction Grammar*, John Benjamins, Amsterdam, Netherlands.

Luc Steels and Joachim De Beule (2006), Unify and merge in Fluid Construction Grammar, in *International Workshop on Emergence and Evolution of Linguistic Communication (EELC 2006)*, pp. 197–223, Rome, Italy.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le (2014), Sequence to sequence learning with neural networks, in Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27, pp. 3104–3112, Curran Associates, Inc., Red Hook, NY, USA.

Takeshi Takahashi, He Sun, Dong Tian, and Yebin Wang (2019), Learning heuristic functions for mobile robot path planning using deep neural networks, in *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pp. 764–772.

Michael Tomasello (2003), *Constructing a language: A usage-based theory of language acquisition*, Harvard University Press, Harvard, MA, USA.

Paul Van Eecke and Katrien Beuls (2017), Meta-layer problem solving for computational construction grammar, in *The 2017 AAAI Spring Symposium Series*, pp. 258–265, AAAI Press, Palo Alto, CA, USA.

Paul Van Eecke and Katrien Beuls (2018), Exploring the creative potential of computational construction grammar, *Zeitschrift für Anglistik und Amerikanistik*, 66(3):341–355.

Rik van Noord, Lasha Abzianidze, Antonio Toral, and Johan Bos (2018), Exploring neural methods for parsing discourse representation structures, *Transactions of the Association for Computational Linguistics*, 6:619–633, doi:10.1162/tacl_a_00241.

Remi van Trijp (2016), Chopping down the syntax tree: What constructions can do instead, *Belgian Journal of Linguistics*, 30(1):15–38.

Remi van Trijp, Katrien Beuls, and Paul Van Eecke (2022), The FCG editor: An innovative environment for engineering computational construction grammars, *PLOS ONE*, 17(6):e0269708, doi:10.1371/journal.pone.0269708.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2017), Attention is all you need, in I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pp. 6000–6010, Curran Associates, Inc., Red Hook, NY, USA.

Lara Verheyen, Jérôme Botoko Ekila, Jens Nevens, Paul Van Eecke, and Katrien Beuls (2022), Hybrid procedural semantics for visual dialogue: An interactive web demonstration, in *Workshop on semantic techniques for narrative-based understanding: Workshop at IJCAI-ECAI 2022*, pp. 48–52.

Jingyuan Wang, Ning Wu, Wayne Xin Zhao, Fanzhang Peng, and Xin Lin (2019), Empowering A* search algorithms with neural networks for personalized route recommendation, in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 539–547.

Pieter Wellens (2011), Organizing constructions in networks, in Luc Steels, editor, *Design Patterns in Fluid Construction Grammar*, pp. 181–201, John Benjamins, Amsterdam, Netherlands.

Pieter Wellens and Joachim De Beule (2010), Priming through constructional dependencies: a case study in Fluid Construction Grammar, in *The Evolution of Language: Proceedings of the 8th International Conference (EVOLANG8)*, pp. 344–351, World Scientific.

Tom Willaert, Paul Van Eecke, Katrien Beuls, and Luc Steels (2020), Building social media observatories for monitoring online opinion dynamics, *Social Media + Society*, 6(2), doi:10.1177/2056305119898778.

Tom Willaert, Paul Van Eecke, Jeroen Van Soest, and Katrien Beuls (2021), An opinion facilitator for online news media, *Frontiers in Big Data*, 4:1–10.

Chen YU and Daniel GILDEA (2022), Sequence-to-sequence AMR parsing with ancestor information, in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 571–577, Association for Computational Linguistics, Dublin, Ireland, doi:10.18653/v1/2022.acl-short.63.

Paul Van Eecke
0000-0001-9153-9092
paul@ai.vub.ac.be

Artificial Intelligence Laboratory, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium

KU Leuven, Faculty of Arts, Research Unit Linguistics, Blijde Inkomststraat 21, 3000 Leuven, Belgium

KU Leuven, imec research group itec, Etienne Sabbelaan 51, 8500 Kortrijk,Belgium

Jens Nevens
0000-0001-5413-2955
jens@ai.vub.ac.be

Artificial Intelligence Laboratory, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium

Katrien Beuls
0000-0003-4451-4778
katrien.beuls@unamur.be

Faculté d'informatique, Université de Namur, rue Grandgagnage 21, 5000 Namur, Belgium

Paul Van Eecke, Jens Nevens, and Katrien Beuls (2022), *Neural heuristics for scaling constructional language processing*, Journal of Language Modelling, 10(2):287–314
https://dx.doi.org/10.15398/jlm.v10i2.318