A transition-based verbal multiword expression analyzer

Hazem Al Saied, Marie Candito, Mathieu Constant

Chapter 7

A transition-based verbal multiword expression analyzer

Hazem Al Saied
ATILF UMR 7118, Université de Lorraine/CNRS

Marie Candito
LLF UMR 7110, Université Paris Diderot/CNRS

Matthieu Constant
ATILF UMR 7118, Université de Lorraine/CNRS

We describe a robust transition-based analyzer for identifying and categorizing Verbal Multiword Expressions (VMWEs). The system was developed and evaluated using the datasets of the PARSEME shared task on VMWE identification (Savary et al. 2017).
We accommodate the variety of linguistic resources provided for each language, in terms of accompanying morphological and syntactic information. Our system produces very competitive scores, for both VMWE identification and categorization, with respect to the shared task results.

1 Introduction

We present a generic system for the identification and categorization of verbal multiword expressions (hereafter VMWEs). With respect to grammatical or nominal multiword expressions, VMWEs tend to exhibit more morphological and syntactic variation than other MWEs, if only because in general the verb is inflected and can receive adverbial modifiers. Furthermore, some VMWE types, in particular light-verb constructions, allow for the full range of syntactic variation

Hazem Al Saied, Marie Candito & Matthieu Constant. 2018. A transition-based verbal multiword expression analyzer. In Stella Markantonatou, Carlos Ramisch, Agata Savary & Veronika Vincze (eds.), *Multiword expressions at length and in depth: Extended papers from the MWE 2017 workshop*, 209–226. Berlin: Language Science Press.
DOI:10.5281/zenodo.1469561

(extraction, coordination etc…). This renders the VMWE identification task even more challenging than general MWE identification, in which fully frozen and continuous expressions contribute to an increase in the overall performance.

Our objective was to design a data-driven system applicable to several languages, with limited language-specific tuning. We took advantage of the datasets provided within the shared task on automatic identification of VMWEs (Savary et al. 2017) to train and test our system. These datasets concern 18 languages, and consist of tokenized sentences in which VMWEs are annotated. One VMWE instance is a set made either of several tokens, potentially non-continuous, or of one single token (i.e. a multiword token, hereafter MWT, such as amalgamated verb-particle in German).[1] A VMWE may be embedded in another longer one, and two VMWEs can overlap. Each annotated VMWE is tagged with a category among Light-Verb Constructions (LVC), IDioms (ID), Inherently REFLexive Verbs (IReflV), Verb-Particle Constructions (VPC) and OTHer verbal MWEs (OTH). The datasets are quite heterogeneous, both in terms of size and of accompanying resources: 4 languages have none (Bulgarian, Spanish, Hebrew, Lithuanian), for 4 languages, morphological information such as lemmas and POS is provided (Czech, Maltese, Romanian, Slovene), and for the 10 remaining languages, full dependency parses are provided.

The system we describe in the current paper is an extension of the ATILF-LLF system (Al Saied et al. 2017), a one-pass greedy transition-based system which participated in the shared task, obtaining very competitive results (hereafter ATILF-LLF 1). The new system (ATILF-LLF 2) categorizes the VMWEs on top of identifying them, and has an extended expressive power, handling some cases of VMWE embedded in another one. Both for ATILF-LLF 1 and 2, we tuned a set of feature template for each language, relying exclusively on training data, accompanying CoNLL-U files when available, and basic feature engineering.

The remainder of the article is organized as follows: we describe our system in Section 2, and comment its expressive power as opposed to ATILF-LLF 1. We then describe the experimental setup in Section 3, and comment the results in Section 4. Section 5 is devoted to related work. We conclude in Section 6 and give perspectives for future work.

[1]The majority of annotated VMWEs are multi-token. The prevalence of MWTs varies greatly among languages. While absent from seven languages and very rare for nine other languages, they are very frequent in German and Hungarian.

2 System description

The analyzers we developed (ATILF-LLF 1 and 2) are simplified versions of the system proposed by Constant & Nivre (2016). Building on the classic arc-standard dependency parser (Nivre 2004), Constant & Nivre (2016) designed a parsing algorithm that jointly predicts a syntactic dependency tree and a forest of multiword lexical units. Their system uses both a syntactic and a lexical stack and specific transitions merge tokens to create MWEs, as proposed by Nivre (2014). We have simplified this formal apparatus keeping only the lexical stack, and predicting MWEs only.

A transition-based system applies a sequence of actions (usually called *transitions*) to incrementally build the expected output structure in a bottom-up manner. Each transition is usually predicted by a classifier given the current state of the system (namely a *configuration*).

A configuration in our system consists of a triplet $c = (\sigma, \beta, L)$, where σ is a stack containing units under processing, β is a buffer containing the remaining input tokens, and L is a set of output MWEs.

The *initial configuration* for a sentence $x = x_1, ..., x_n$, i.e. a sequence of n tokens, is represented by c_s as: $c_s(x) = ([], [x_1, \dots, x_n], \emptyset)$ and the set of *terminal configurations* C_t contains any configuration of the form $c_t = ([], [], L)$. At the end of the analysis, the identified VMWEs are simply extracted from L.

Single tokens are brought to the stack by the Shift transition, and are potentially marked as (mono-token) VMWE using the Mark as C transition, whereas trees are formed using merge transitions (cf. §2.1).

The output VMWEs are the units added to L, either by the Mark as C or Merge as C transitions. When one VMWE is embedded in another one, both VMWEs appear separately in L (which is thus redundant).[2]

2.1 Transition set

Our system uses the following transitions:

1. the **Shift** transition moves the first element of the buffer to the stack

   Precondition: the buffer is not empty.

2. the **Reduce** removes the top element of the stack

   Precondition: the stack is not empty.

[2]For instance, if we represent the binary tree in bracketing format and the categorization with a subscript, $((a, b)_{IReflV}, c)_{ID}$ represents an IReflV $a + b$ embedded within an ID $a + b + c$. Both $((a, b)_{IReflV}, c)_{ID}$ and $(a, b)_{IReflV}$ will appear in L.

3. the **White Merge** transition combines the two top elements of the stack into a single element;

   Precondition: the stack contains at least two elements.

4. five **Merge As C** transitions (where **C** stands for a VMWE category) perform a white merge, mark the resulting unit as a VMWE of category C, and add it to *L*.

   Precondition: the stack contains at least two elements.

5. In order to cope with MWTs, we added five **Mark as *C*** transitions, which mark the top stack element as a VMWE, assign to it the category C, and add it to *L*.

   Precondition: The stack is not empty and its head is a non-marked single token.[3]

Figure 1 shows the analysis of a German sentence containing a multiword token VPC embedded within an IReflV.

In the input, each token is associated with linguistic attributes (form, and depending on the data sets, lemma and POS). When a merge transition is applied, the newly created element gets its attributes using basic concatenation over forms, and over lemmas and POS tags when available.[4]

2.2 Parsing algorithm and training oracle

In all the following, at parsing time we use a greedy algorithm, starting with an initial configuration c_s, and applying in sequence the best-scoring legal transition until a terminal configuration is reached.[5]

The training phase learns the transition-scoring classifier. This is done through supervised learning, by converting the training sentences into sequences of [con-

[3]Because mono- and multi-tokens have very different linguistic properties, we preferred to distinguish transitions coping with both kinds of VMWEs. Without this restriction, as noted by a reviewer, Merge as C would be equivalent to White Merge + Mark as C. The effectiveness of this alternative solution remains to be tested.

[4]This would deserve to be improved in future experiments, with finer procedures to predict the lemmatized form and more importantly to predict the POS tag of the merged node, although in the special case of VMWE prediction the POS is verbal.

[5]Sentence analysis is composed of exactly $2n + r$ transitions, with n being the number of tokens and r the number of MWTs. Every single token not entering a VMWE requires a Shift and a Reduce, every multi-token VMWE of length m requires $2m$ transitions (m Shifts, $m-2$ White Merges, one Merge As C and one Reduce), while every MWT requires three transitions: a Shift, a Mark As C and a Reduce.

Transition		Configuration
		[ ], [Damit, müsste, man, sich, nun, herumschlagen], [ ]
Shift	⇒	[Damit], [müsste, man, sich, nun, herumschlagen], [ ]
Reduce	⇒	[ ], [müsste, man, sich, nun, herumschlagen], [ ]
Shift	⇒	[müsste], [man, sich, nun, herumschlagen], [ ]
Reduce	⇒	[ ], [man, sich, nun, herumschlagen], [ ]
Shift	⇒	[man], [sich, nun, herumschlagen], [ ]
Reduce	⇒	[ ], [sich, nun, herumschlagen], [ ]
Shift	⇒	[sich], [nun, herumschlagen], [ ]
Shift	⇒	[sich, nun], [herumschlagen], [ ]
Reduce	⇒	[sich], [herumschlagen], [ ]
Shift	⇒	[sich, herumschlagen], [ ], [ ]
Mark as VPC	⇒	[sich, herumschlagen$_{VPC}$], [ ], [herumschlagen$_{VPC}$]
Merge as IreflV	⇒	[(sich, herumschlagen$_{VPC}$)$_{IReflV}$], [ ], [herumschlagen$_{VPC}$, (sich, herumschlagen$_{VPC}$)$_{IReflV}$]
Reduce	⇒	[ ], [ ], [herumschlagen$_{VPC}$, (sich, herumschlagen$_{VPC}$)$_{IReflV}$]

Figure 1: Transition sequence for tagging the German sentence *Damit müsste man **sich** nun **herumschlagen*** 'With-that must-SUBJUNCTIVE one REFLEXIVE now around-struggle' ⇒ 'One would have to struggle with that' , containing two VMWEs: SICH HERUMSCHLAGEN tagged as inherently reflexive verb (IReflV), in which HERUMSCHLAGEN is itself a multiword token tagged as verb-particle combination (VPC).

figuration, gold transition to apply] pairs, by using a static oracle.[6] Our static oracle returns for a given configuration the first applicable transition using the following priority order: MARK AS C, MERGE AS C, WHITE MERGE, REDUCE and SHIFT. Applicability here means not only the standard preconditions for transitions, but also that the output configuration is compatible with the gold annotations. We added the constraint that the WHITE MERGE is only applicable to the right suffix of a gold VMWE. For instance, suppose we have a gold continuous VMWE *kick the bucket*, when the first two elements are on top of the stack, the WHITE MERGE is not applicable yet, it will be applied when the right suffix *the* and *bucket* is on top of the stack.

To produce our training examples, we start by generating the initial configuration for each sentence, and apply in sequence the transition predicted by the static oracle, until a terminal configuration is reached. The analysis in Figure 1 corresponds to the oracle transition sequence for the example sentence.

[6]Using (Goldberg & Nivre 2013)'s terminology, a static oracle is both incomplete (defined for configurations obtained from previous oracle transitions only) and deterministic (at each such configuration, there is a single oracle transition to apply).

2.3 Expressive power

As far as expressive power is concerned, ATILF-LLF 2 is slightly more powerful than ATILF-LLF 1. ATILF-LLF 2 now performs VMWE categorization and not just identification. Both systems cannot analyze interleaving MWEs, but while ATILF-LLF 1 could cope with no overlapping at all, ATILF-LLF 2 can cope with some cases of embeddings, i.e. some cases of VMWE fully contained in another one.[7]

In effect, ATILF-LLF 1 contained a SHIFT transition, a WHITE MERGE, a MERGE AS C+REDUCE for identifying multi-token VMWEs, and a MARK AS C+REDUCE for MWTs.[8] Because the MERGE AS C+REDUCE transition identifies a MWE and removes it from the stack, no cases of embeddings were covered.

In ATILF-LLF 2, some cases of embeddings are now covered (e.g. the example in Figure 1). More precisely, the covered cases are those where one can form a projective tree by attaching to a fictitious root all the binary trees representing the VMWEs of a sentence, ignoring tokens not belonging to any VMWE. An alternative formulation is that given any VMWE composed of the $t_1 t_2 ... t_m$ tokens and any gap $g_1 g_2 ... g_n$ appearing between a pair of components $t_i t_{i+1}$, the condition is that the g_i tokens cannot belong to a MWE having components outside the set $g_1 ... g_n$. So a non-covered case is found for instance in **LET**$_{1,2}$ THE$_1$ CAT$_1$ **OUT**$_{1,2}$ OF$_1$ THE$_1$ BAG$_1$: the VMWE **LET OUT** has a gap containing tokens THE$_1$ CAT$_1$, which belong to a VMWE with tokens outside the gap.

3 Experimental setup

For a given language, and a given train/dev split, we use the oracle-based resulting transition sequences to train a multi-class SVM classifier.[9] We describe in the next subsections the feature templates we used (§3.1) and how we tuned them (§3.2).

[7] It is worth noting that embedded VMWEs are very rare in the datasets: there are twenty to thirty embedded VMWEs in German, Hebrew and Hungarian and about 150 in Czech.

[8] ATILF-LLF 1 used hard-coded procedures for matching MWTs (if seen in the training set), which we replaced by features used by the classifier.

[9] The whole system was developed using Python 2.7, with 4,739 lines of code, using the Scikit-learn 0.19. We used the Error-correcting output codes framework for the multi-class SVM classifier. The code is available on Github: https://goo.gl/1j8mVu under the MIT license.

Table 1: System setting code descriptions.

Code	Setting description
A	use of POS and lemmas
A'	use of suffixes
B	use of syntactic dependencies
C	use of bigrams S_1S_0, S_0B_0, S_1B_0 and S_0B_1
D	use of the trigram $S_1S_0B_0$
E	use of the S_0B_2 bigram
F	use of transition history (length 1)
G	use of transition history (length 2)
H	use of transition history (length 3)
I	use of distance between S_0 and S_1
J	use of distance between S_0 and B_0
K	use of B_1
L	use of training corpus VMWE lexicon
M	use of stack length
N	use of MWT dictionary

3.1 Feature templates

A key point in a classical transition-based system is feature engineering, known to have great impact on performance. We have gathered feature templates into groups, for which we provide short descriptions in Table 1, along with code letters that we use in §4 to describe which feature groups were used for each language in the final shared task results. We describe in this section each feature group. We hereafter use symbol B_i to indicate the ıth element in the buffer. S_0 and S_1 stand for the top and the second elements of the stack. For every unit X in the stack or buffer, we denote X_w its word form, X_l its lemma and X_p its POS tag. The concatenation of two elements X and Y is noted XY.

Basic linguistic features

For each language, we used a precise set of stack or buffer elements, hereafter the *focused elements*, to derive unigram, bigram and trigram features. By default, the focused elements are S_0, S_1 and B_0. For some languages, B_1 was also used (code K in Table 1). If bigrams are on (code C in Table 1) features are generated for the element pairs S_1S_0, S_0B_0, S_1B_0, plus S_0B_1 if K is on, and plus S_0B_2 for a few

languages (code E). For trigrams, we only used the features of the $S_1S_0B_0$ triple (code D).

For any resulting unigram, bigram or trigram, we use by default the word form (e.g. S_0w). For languages whose datasets comprise morphological information, we further use the lemmas and POS tags (code A in Table 1), i.e. X_l and X_p. The precise features for a bigram XY are X_wY_w, X_pY_p, X_lY_l, X_pY_l and X_lY_p. Those for a trigram XYZ are $X_wY_wZ_w$, $X_lY_lZ_l$, $X_pY_pZ_p$, $X_lY_pZ_p$, $X_pY_lZ_p$, $X_pY_pZ_l$, $X_lY_lZ_p$, $X_lY_pZ_l$, $X_pY_lZ_l$.

For the languages lacking companion morphological information, we tried to mimic that information using suffixes (code A' in Table 1), more precisely the last two and last three letters, which we used for unigram elements only.

Syntax-based features

After integrating classical linguistic attributes, we investigated using more linguistically sophisticated features. First of all, syntactic structure is known to help MWE identification (Fazly et al. 2009; Seretan 2011; Nagy T. & Vincze 2014). So for datasets comprising syntactic information, we introduced features capturing the existence of syntactic dependencies between elements of the buffer and of the stack (code B in Table 1). More precisely, provided that S_0 is a single token, we generate (i) the features **RightDep (S_0, B_i) = True** and **RightDepLab (S_0, B_i) = l** for each buffer token B_i that is a syntactic dependent of S_0 with label l, and (ii) the features **LeftDep (S_0, B_i) = True** and **LeftDepLab (S_0, B_i) = l** when a buffer element B_i is S_0's syntactic governor.

Other syntax-based features aim at modeling the direction and label of a syntactic relation between the top two tokens of the stack (feature **syntacticRelation (S_0, S_1) = ± l** is used for S_0 governing/governed by S_1, provided S_0 and S_1 are single tokens).[10] All these syntactic features try to capture syntactic regularities between the tokens composing a VMWE.

Miscellaneous features

We found that other traditional features, used in transition-based systems, were sometimes useful like (local) transition history of the system. We thus added **History-based features** to represent the sequence of previous transitions (of length one, two or three, cf. codes F, G and H in Table 1).

[10]For ATILF-LLF 1, we used gold syntactic features for the languages accompanied with gold dependency companion files, as authorized in the closed track. Performance when using predicted syntax will be evaluated in future work.

We also added **Distance-based features**, known to help transition-based dependency parsing (Zhang & Nivre 2011), more precisely the distance between S_0 and S_1 and between S_0 and B_0 (respectively codes I and J in Table 1). We also extracted **Stack-length-based features** (code M in Table 1).

The VMWE identification task is highly lexical so we found it useful to use **dictionary-based features**, which use "dictionaries" extracted from the training set, both for multi-token VMWEs and MWTs. The dictionaries are lemma-based when lemmas are available, and form-based otherwise. These dictionary-based features include (i) a boolean feature set to true when S_0 belongs to the MWT dictionary (code N in Table 1), and (ii) boolean features firing when S_0, S_1, B_0, B_1 or B_2 belong to an entry of the extracted VMWE dictionary (code L in Table 1).

3.2 Feature tuning

We first divided the data sets into three groups, based on the availability of CoNLL-U files: (a) for **Bulgarian**, **Hebrew** and **Lithuanian** neither morphological nor syntactic information is available on top of tokens and VMWE annotation; (b) **Czech**, **Spanish**, **Farsi**, **Maltese** and **Romanian** are accompanied by CoNLL-U files with morphological information only, and (c) **the other languages**[11] are accompanied by a fully annotated CoNLL-U file.

In the first tuning phase, we used one "pilot" language for each group (Bulgarian, Czech and French). Then, German was added as pilot language to investigate features for languages with high percentage of MWTs and embedded VMWEs. We tuned the features using both development sets extracted from the provided training sets, and using cross-validation.

Finally, we used the discovered feature groups as a source of inspiration for producing specialized feature groups for all other languages. Note that given the combinatorial explosion of feature combinations, we could not apply a full grid-search for the pilot languages, and a fortiori for all languages.

4 Results

We provide the identification results in Table 2, in which the performance of ATILF-LLF 2, both in the shared task test sets and in cross-validation, can be compared with (i) a baseline system, (ii) the best performing system of the shared

[11] These languages are German, Greek, French, Hungarian, Italian, Polish, Portuguese, Slovene, Swedish, and Turkish.

Table 2: **VMWE identification**: The first column provides the language, it is shown whether the companion file contains morpho and syntax, morpho only (*) or nothing (**). The last column lists the feature groups used for that language (using the codes of Table 1). Columns 2, 3, 4: VMWE-based F-scores on test sets, for the **baseline** system, **ATILF-LLF 2**, and the best performing Shared task systems (**Best of ST**). Columns 5, 6, 7: same as 2, 3, 4 but token-based. Columns 8, 9, 10: VMWE-based results in 5-fold cross-validation over training sets, for the baseline system, ATILF-LLF 1, and ATILF-LLF 2. The last row (Avg) provides the average results weighted by the size of the test sets (or train sets for cross-validation results). The stars in columns Best of ST are those for which ATILF-LLF 1 did not rank first.

	Test dataset						Cross validation			
	VMWE-based F_1			Token-based F_1			VMWE-based F_1			
Language	Baseline	ATILF-LLF 2	Best of ST	Baseline	ATILF-LLF 2	Best of ST	Baseline	ATILF-LLF 1	ATILF-LLF 2	Feature Settings
BG**	47.6	55.8	**61.3**	50.7	60.2	**66.2**	48.3	**57.1**	53.0	A' C D F G I L
CS*	61.6	70.9	**71.7**	66.7	**73.9**	73.7	60.1	**71.4**	68.9	A C F G H I J K L M
DE	37.9	**45.8**	41,1	33.3	44.8	***45.5**	39.9	27.9	**47.6**	A B C D E J L N
EL	35.6	**42.8**	40.1	39.9	46.3	**46.9**	48.0	56.2	**57.3**	A B C E J K L
ES*	56.9	**58.9**	57.4	56.7	**60.3**	58.4	61.2	63.5	**66.0**	A C D F G H I J K L
FA*	72.2	84.3	**86.6**	72.5	84.8	**90.2**	67.3	**87.7**	81.1	A C I J K
FR	44.6	**60.6**	57.7	49.3	**62.6**	*61.5	66.0	71.1	**73.8**	A B C E I J K L
HE**	**33.4**	29.9	**33.4**	29.6	30.5	**31.3**	**30.0**	17.0	26.8	A' C E F G H K L
HU	68.3	**74.8**	*74.0	64.9	**72.1**	*70.8	73.7	23.5	**83.7**	A B C D F G H K L N
IT	39.2	28.2	**39,9**	39.5	29.8	**43.6**	**33.7**	27.4	27.2	A B C H J L
LT**	30.5	**34.1**	28.4	27.3	**31.7**	25.3	20.7	8.6	**21.5**	A' C D E F G H I J K L M
MT*	8.2	6.9	**14.4**	12.3	9.4	**16.3**	7.7	**8.1**	7.2	A C F G H J L M
PL	72.6	**75.1**	69.1	71.8	**75.5**	*72.7	70.0	70.4	**73.6**	A B C H L
PT	65.5	**69.6**	67.3	67.4	**71.4**	70.9	65.2	64.7	**67.5**	All features
RO*	55.0	**86.3**	*77.8	65.4	**87.0**	*83.6	61.8	**86.3**	86.0	A C D E F G H I J K
SL	13.9	42.9	**43.2**	17.6	45.7	**46.6**	17.3	**47.7**	40.8	A B C F G H I K N
SV	10.4	30.1	**30.4**	10.1	**34.3**	*31.5	6.9	**25.0**	24.7	All features except N
TR	11.3	53.8	**55.4**	18.1	**53.9**	55.3	19.3	58.1	**60.1**	A B C F G H I K
AVG	46.2	56.5	**56.7**	48.5	58.1	**59.2**	52.0	60.3	**64.5**	

task, and with (iii) ATILF-LLF 1.[12] The table shows that results are very heterogeneous across languages. We can hypothesize that multiple factors come into play, such as the size of corpora, the availability and the quality of annotations, the most common VMWE categories in train and test sets, the percentage of unknown VMWEs in test sets. For example, Figure 2 illustrates the impact of this last trait, showing an approximative linear negative correlation between VMWE-based F-score and the proportion of unknown VMWE occurrences in test sets.[13]

Because the datasets have very varying sizes across languages, we provide in the last row of the table the weighted average F-scores, with each language F-score weighted by the size of the test set (or of the training set in cross-validation).

Comparison with the best results at the shared task: Although the ATILF-LLF 2 benefited from more design time, it is interesting to compare its results to the best results obtained at the shared task for each language. When considering the weighted average results (last row of Table 2), it can be seen that the VMWE-based results are almost as high for ATILF-LLF 2 as for the Best of ST (56.5 versus 56.7), and are ahead for 9 languages out of 18. For token-based results, our system is a bit less effective: while still ahead for 10 languages out of 18, it is on average 1.1 point lower (58.1 versus 59.2). This can be viewed as a particularity of our system: while the token-based results are generally higher than the VMWE-based ones (for the baseline, or for other participating systems, cf. Savary et al. 2017), the gap is less pronounced in our case.

Comparison with the baseline: The baseline system is a string matching-based system that uses a lemma-based VMWE dictionary extracted from the training set and identifies as VMWEs all matching strings in the test set.

The matching procedure is very simple: a VMWE is identified inside a sentence if all of its components (lemmas if available, otherwise word forms) occur in the sentence, provided that the order of the components corresponds to an order observed in the dictionary and that the distances between them do not exceed the maximal observed distances in the training dataset.

[12] More precisely, for the results on the test sets (columns 2 to 7), the **Best of ST** columns reflect the performance of ATILF-LLF 1 for the non starred values (cf. no star means we ranked first). F-scores of ATILF-LLF 1 for starred values are as following: Hungarian=70%, Romanian=75% for VMWE-based and German=41%, French=60%, Hungarian=67.4%, Polish=70.5%, Romanian=79.1% and Swedish=30% for token-based.

[13] We also checked for the correlation between the F-score and the training set size, and obtained a positive correlation, but less marked, in particular some languages like Czech and Turkish reach relatively low scores given the size of training data, which is better explained considering the unknown VMWE rate.

Regarding VMWE-based evaluation, ATILF-LLF 2 outperforms the baseline in all experimental settings but four (VMWE-based evaluation on test set for Hebrew and VMWE-based cross-validation for Hebrew and Italian): on average, we obtain about 10- and 12.5-point F-score difference when evaluating on the test set and in cross-validation respectively. Yet, the baseline consistently beats our system on Hebrew. This might be explained by several characteristics of this language preventing the system to generalize well to morpho-syntactic variants: (i) small training set and (ii) no companion linguistic information (no POS, no lemmas, no syntactic parses).

Comparison between ATILF-LLF 2 and ATILF-LLF 1: The ATILF-LLF 1 system participated in the shared task and reached the best VMWE-based scores for almost all languages (cf. the two starred results out of 18 in column Best of ST, for Hungarian and Romanian). It can be seen that ATILF-LLF 2 shows comparable performance on the same test sets (see in particular the weighted average performance shown in the last row: 56.5 versus 56.7). It is worth noticing though that there is great variation between results on test sets and results in cross-validation. As the latter are more representative, let us focus on them (columns 8 to 10). Despite a few languages showing a drop in performance (in particular Bulgarian, Farsi and Slovene), ATILF-LLF 2 beats ATILF-LLF 1 for 10 languages out of 18, and the average result (last row of Table 2) has improved (4.2-point gain). Again, even though ATILF-LLF 2 benefited from more design time, this is a good result considering that (i) ATILF-LLF 1 did identification only, and the introduction of the categorization task led us to multiply the number of transitions (e.g. 5 MERGE AS C transitions instead of 1), (ii) the expressive power was increased to some cases of embeddings and (iii) the overall architecture is more elegant since hard-coded procedures included in the rush of the shared task have been replaced by features.[14]

Categorization results: Table 3 details the categorization results for the basic categories over all languages but Farsi (cf. the Farsi dataset does not comprise VMWE category information).[15] The table allows us to compare the performance of our system with best-performing shared task systems (for the systems having the optional categorization predictions, note that our former system ATILF-LLF 1 is excluded given that it does not categorize VMWEs). It can be seen

[14]It is worth noting that feature groups for each language were very close for both systems (ATILF-LLF 1, 2). However, we transformed the dictionary-based hard-coded feature groups into dynamic ones.

[15]We do not include the category OTH because of its marginal presence in test and train datasets for all languages but Turkish (for which our F-score is 51.5 and the Shared task best F-score is 54.6).

Table 3: **VMWE categorization**: detailed results for the four basic categories over all the languages except Farsi. For each category, we display the proportion of the given category **in test set**, the F-scores $\mathbf{F_1}$ for **ATILF-LLF 2**, and the best performing shared task systems (**Best of ST**), among systems having provided categorization information.

	LVC			IReflV			VPC			ID		
	%	F_1	F_1	%	F_1	F_1	%	F_1	F_1	%	F_1	F_1
Languages	In test set	ATILF-LLF 2	Best of ST	In test set	ATILF-LLF 2	Best of ST	In test set	ATILF-LLF 2	Best of ST	In test set	ATILF-LLF 2	Best of ST
BG	16	**28.6**		63	**68.3**	46.6				21	**19.4**	
CS	20	**53.3**	40.1	68	**80.7**	73.3				11	**33.6**	22.0
DE	8	**4.7**	2.3	4	8.9	**16.0**	45	**58.3**	43.3	43	**29.1**	16.4
EL	67	**44.4**	33.2				3	**52.2**	36.4	25	**27.1**	15.4
ES	21	**49.0**	35.1	44	**72.0**	40.4				33	**39.0**	13.8
FR	54	40.0	**42.8**	21	**78.6**	68.3				24	**75.0**	60.8
HE	25	**31.4**					37	**27.2**		6	**16.2**	
HU	29	**50.9**	41.5				71	**80.3**	77.2			
IT	17	**17.5**	12.9	30	**30.3**	9.3	2	**50.0**	14.3	50	**25.7**	20.3
LT	42	**56.3**								58	**14.9**	
MT	52	**6.6**	5.8							52	**7.7**	2.1
PL	34	**62.9**	39.1	53	**87.3**	80.2				13	**51.5**	
PT	66	**68.0**		16	**68.9**					18	**65.3**	
RO	27	**87.4**	86.3	58	**86.3**	79.1				15	**76.7**	65.6
SL	9	7.4	**8.3**	51	**45.7**	40.8	22	**47.5**	34.5	18	**09.1**	3.9
SV	6	16.7	**21.1**	6	**08.7**		66	**33.6**	30.2	21	**13.3**	3.8
TR	40	57.6	**59.1**							11	**49.3**	49.8

that our system reaches high performance on categorization too, although performance varies greatly across categories. Although the general trend is higher performance for IReflV, then LVC, then ID, Figure 3 shows that this pattern is not systematic. For instance, results are relatively low for Czech given its high IReflV proportion. On the contrary, results for Portuguese are high despite a high LVC ratio.

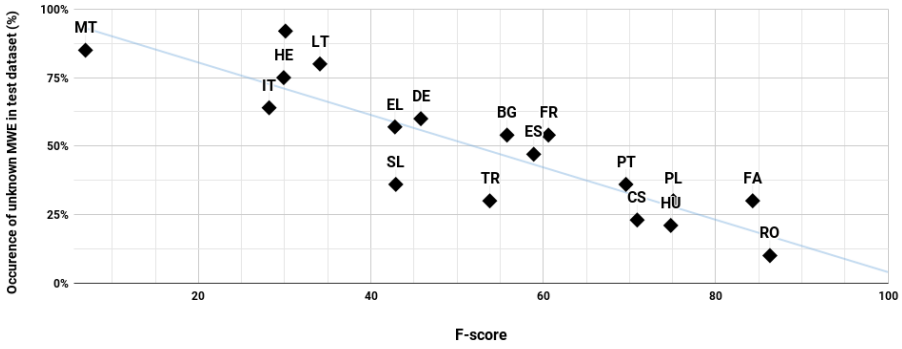

Figure 2: Correlation between the ATILF-LLF 2 identification results for each language (F-score, on the x axis) and the percentage of **occurrences of test VMWEs unknown in the train set** (y axis).

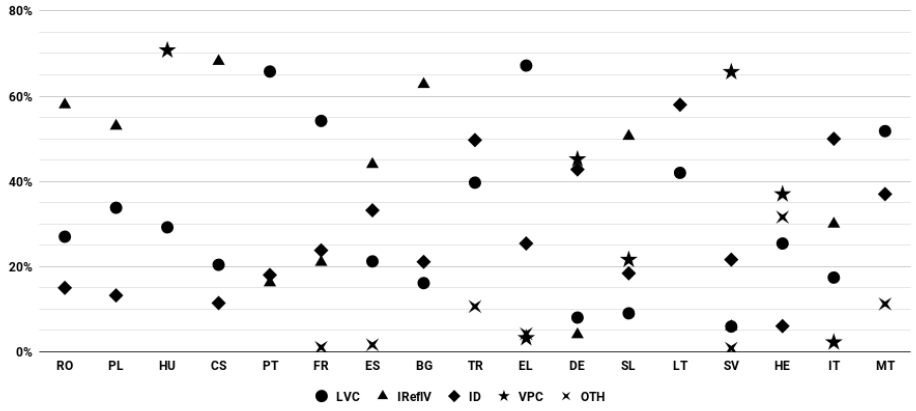

Figure 3: A graph ranking all languages except Farsi according to their F-scores. In each bar, the proportions of VMWE categories in the test set are shown using symbols.

5 Related work

A popular VMWE identification method is to use a sequence labeling approach, with IOB-based tagsets. For instance, Diab & Bhutada (2009) apply a sequential SVM to identify verb-noun idiomatic combinations in English. Note also that three (out of seven) systems participating in the PARSEME shared task used such approach (Boroş et al. 2017; Maldonado et al. 2017; Klyueva et al. 2017). Such an approach was also investigated for MWE identification in general (including verbal expressions) ranging from continuous expressions (Blunsom & Baldwin 2006) to gappy ones (Schneider et al. 2014). Recently, neural networks have been successfully integrated into this framework (Legrand & Collobert 2016; Klyueva et al. 2017).

VMWE identification can naturally take advantage of previously predicted syntactic parses. Some systems use them as soft constraints. For instance, the sequence labeling systems of the shared task and our system use them as source of features in their statistical tagging models. There also exist approaches using syntactic parses as hard constraints. For example, Baptista et al. (2015) apply hand-crafted identification rules on them. Fazly et al. (2009) and Nagy T. & Vincze (2014) propose a two-pass identification process consisting of candidate extraction followed by binary classification. In particular, candidate extraction takes advantage of predicted syntactic parses, through the use of linguistic patterns.

A joint syntactic analysis and VMWE identification approach using off-the-shelf parsers is another interesting alternative that has shown to help VMWE identification such as light-verb constructions (Eryiğit et al. 2011; Vincze & Csirik 2010). Some parsers integrate mechanisms into the parsing algorithm to identify MWEs on top of predicting the syntactic structure, like in Wehrli (2014) and Constant & Nivre (2016), our system being a simplified version of the latter.

6 Conclusion and future work

This article presents a simple transition-based system devoted to VMWE identification and categorization. In particular, it offers a simple mechanism to handle discontinuity and embedding, which is a crucial point for VMWEs. Results on the PARSEME Shared Task datasets show that our system is quite robust across languages, and achieves very competitive results. Its linear time complexity is also an asset.

As future work, we would like to apply more sophisticated syntax-based features, as well as more advanced machine-learning techniques like neural networks. We also plan to investigate the use of a dynamic oracle (Goldberg & Nivre 2012).

Acknowledgements

This work was partially funded by the French National Research Agency (PARSEME-FR ANR-14-CERA-0001).

Abbreviations

ID	Idiom	MWT	multiword token
IOB	inside outside beginning	OTH	other verbal MWE
IREFLV	inherently reflexive verb	VMWE	verbal multiword expression
LVC	light-verb construction	VPC	verb-particle construction
MWE	multiword expression		

References

Al Saied, Hazem, Matthieu Constant & Marie Candito. 2017. The ATILF-LLF system for parseme shared task: A transition-based verbal multiword expression tagger. In *Proceedings of the 13th Workshop on Multiword Expressions* (MWE '17), 127–132. Association for Computational Linguistics. DOI:10.18653/v1/W17-1717

Baptista, Jorge, Graça Fernandes, Rui Talhadas, Francisco Dias & Nuno Mamede. 2015. Implementing European Portuguese verbal idioms in a natural language processing system. In Gloria Corpas Pastor (ed.), *Proceedings of europhras 2015*, 102–115.

Blunsom, Phil & Timothy Baldwin. 2006. Multilingual deep lexical acquisition for HPSGs via supertagging. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, 164–171. Sydney, Australia: Association for Computational Linguistics. DOI:10.3115/1610075.1610101

Boroş, Tiberiu, Sonia Pipa, Verginica Barbu Mititelu & Dan Tufiş. 2017. A data-driven approach to verbal multiword expression detection. PARSEME shared task system description paper. In *Proceedings of the 13th Workshop on Multiword Expressions* (MWE '17), 121–126. Association for Computational Linguistics. DOI:10.18653/v1/W17-1716

Constant, Matthieu & Joakim Nivre. 2016. A transition-based system for joint lexical and syntactic analysis. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, 161–171. Association for Computational Linguistics. http://www.aclweb.org/anthology/P16-1016.

Diab, Mona & Pravin Bhutada. 2009. Verb noun construction MWE token classification. In *Proceedings of the Workshop on Multiword Expressions: Identification, Interpretation, Disambiguation and Applications*, 17–22. Association for Computational Linguistics. http://www.aclweb.org/anthology/W/W09/W09-2903.

Eryiğit, Gülşen, Tugay İlbay & Ozan Arkan Can. 2011. Multiword expressions in statistical dependency parsing. In *Proceedings of IWPT Workshop on Statistical Parsing of Morphologically-Rich Languages* (SPMRL 2011), 45–55. http://dl.acm.org/citation.cfm?id=2206359.2206365. October 6, 2011.

Fazly, Afsaneh, Paul Cook & Suzanne Stevenson. 2009. Unsupervised type and token identification of idiomatic expressions. *Computational Linguistics* 35(1). 61–103. http://aclweb.org/anthology/J09-1005.

Goldberg, Yoav & Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING)*, 959–976.

Goldberg, Yoav & Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the Association for Computational Linguistics* 1. 403–414.

Klyueva, Natalia, Antoine Doucet & Milan Straka. 2017. Neural networks for multi-word expression detection. In *Proceedings of the 13th Workshop on Multiword Expressions* (MWE '17), 60–65. Association for Computational Linguistics. April 4, 2017. DOI:10.18653/v1/W17-1707

Legrand, Joël & Ronan Collobert. 2016. Phrase representations for multiword expressions. In *Proceedings of the 12th Workshop on Multiword Expressions* (MWE '16), 67–71. Association for Computational Linguistics. http://anthology.aclweb.org/W16-1810.

Maldonado, Alfredo, Lifeng Han, Erwan Moreau, Ashjan Alsulaimani, Koel Dutta Chowdhury, Carl Vogel & Qun Liu. 2017. Detection of verbal multi-word expressions via conditional random fields with syntactic dependency features and semantic re-ranking. In *Proceedings of the 13th Workshop on Multiword Expressions* (MWE '17), 114–120. Association for Computational Linguistics. DOI:10.18653/v1/W17-1715

Nagy T., István & Veronika Vincze. 2014. VPCTagger: Detecting verb-particle constructions with syntax-based methods. In *Proceedings of the 10th Workshop*

on Multiword Expressions (MWE '14), 17–25. Association for Computational Linguistics. http://www.aclweb.org/anthology/W14-0803.

Nivre, Joakim. 2004. Incrementality in deterministic dependency parsing. In Frank Keller, Stephen Clark, Matthew Crocker & Mark Steedman (eds.), *Proceedings of the ACL Workshop on Incremental Parsing: Bringing Engineering and Cognition together*, 50–57. Association for Computational Linguistics.

Nivre, Joakim. 2014. *Transition-based parsing with multiword expressions*. IC1207 COST PARSEME 2nd general meeting. Athens, Greece. https://typo.uni-konstanz.de/PARSEME/images/Meeting/2014-03-11-Athens-meeting/PosterAbstracts/WG3-Nivre-athens-poster.pdf.

Savary, Agata, Carlos Ramisch, Silvio Cordeiro, Federico Sangati, Veronika Vincze, Behrang QasemiZadeh, Marie Candito, Fabienne Cap, Voula Giouli, Ivelina Stoyanova & Antoine Doucet. 2017. The PARSEME Shared Task on automatic identification of verbal multiword expressions. In *Proceedings of the 13th Workshop on Multiword Expressions* (MWE '17), 31–47. Association for Computational Linguistics. DOI:10.18653/v1/W17-1704

Schneider, Nathan, Emily Danchik, Chris Dyer & Noah A. Smith. 2014. Discriminative lexical semantic segmentation with gaps: Running the MWE gamut. *Transactions of the Association for Computational Linguistics* 2(1). 193–206. https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/281.

Seretan, Violeta. 2011. *Syntax-based collocation extraction* (Text, Speech and Language Technology). Dordrecht, Heidelberg, London, New York: Springer.

Vincze, Veronika & János Csirik. 2010. Hungarian corpus of light verb constructions. In *Proceedings of the 23rd International Conference on Computational Linguistics* (COLING '10), 1110–1118. Association for Computational Linguistics. http://www.aclweb.org/anthology/C10-1125.

Wehrli, Eric. 2014. The relevance of collocations for parsing. In *Proceedings of the 10th Workshop on Multiword Expressions* (MWE '14), 26–32. Association for Computational Linguistics. 26-27 April, 2014.

Zhang, Yue & Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, 188–193. Association for Computational Linguistics. http://www.aclweb.org/anthology/P11-2033.