Solvers for Mathematical Word Problems in Czech

Jan Kadlec, Daniel Průša

Czech Technical University, Faculty of Electrical Engineering
Karlovo nám. 13, 121 35 Prague 2, Czech Republic
{kadlej24,prusa}@fel.cvut.cz

Abstract: We study the task of an automatic evaluation of mathematical word problems, which belongs to the category of natural language processing and has become popular in recent years. Since all the so far published methods were developed for inputs in English, our goal is to review them and propose solutions able to cope with inputs in the Czech language. We face the question whether we can achieve a competitive accuracy for a natural language with flexible word order, and with the assumption that only a relatively small dataset of training and testing data is available.

We propose and evaluate two methods. One relies on a rule-based processing of dependency trees computed by UDPipe. The other method builds on machine learning. It transforms word problems into numeric vectors and trains SVM to classify them. We also show that it improves in a combination with a search for predefined sequences of words and word classes, achieving 75% accuracy on our dataset of 500 Czech word problems.

1 Introduction

The research in an automatic evaluation of mathematical word problems has nowadays a good practical motivation. This motivation comes from the fact that a natural language is the easiest way how many people can express themselves. Hence, it is useful to have methods that analyze assignments described by words, transform them into an internal representation, calculate results and present them in a descriptive form. This is exactly what is aimed by computational knowledge engines like WolframAlpha[1]. Mathematical word problems can be seen as one of the most complicated text assignments. Moreover, the complexity of translating a mathematical text into symbolic math scales with the difficulty of math involved in the problem solution. Hence, from a scientific point of view, word problems represent a very suitable domain for research.

The interest in automatic evaluation of word problems dates back to 1963. It was identified in [4] as one of the important challenges for artificial intelligence in upcoming decades. A first attempt to implement an automatic solver was done by Bobrow within his diploma thesis [2]. The next development was not so rapid since this initial study until 2014 when Kushman et al. published their solution based on semantic interpretation and information extraction [9]. Several other works then followed.

The survey [12] suggests that three different principles for solving word problems automatically can be distinguished. The simplest approach works with *patterns* which are sequences of words. Each pattern represents one type of word problems. Classification is done by detecting the sequences in inputs. The approach is used e.g. by WolframAlpha. It might be problematic to use it for languages with flexible word order.

The second approach relies on *syntactic analysis*. Syntactic dependencies among words are used to extract information from a given word problem. A disadvantage might be a dependence on linguistic tools. System ARIS [5] is an example of this type of solver.

The most efficient approach is to apply *machine learning*. Kushman et al. [9] use machine learning together with information extraction and semantic interpretation. The role of the machine learning part is to estimate parameters of a probabilistic model. Huang et al. [6] apply neural networks and reinforcement learning. Their method works without an explicit information extraction and is even able to solve types of word problems that were not present in the training dataset. On the other hand, the method fails sometimes because of generating numbers not involved in the input instance.

We can compare the accuracy of the methods since both were evaluated on dataset Alg514 [9] consisting of 514 word problems. The method from [9] achieved 70% accuracy while the method from [6] achieved 82.5% accuracy. However, it is a known fact that the accuracy of all solvers drops significantly down when they are applied to instances from a large and much more diverse dataset. One such a dataset, Dolphin18K consisting of 18.000 word problems, was introduced in [7]. None of the solvers evaluated in [7] and [6] was able to solve correctly more than 34% instances from this dataset. This shows that solving math word problems automatically is a really challenging task.

All the systems discussed above were developed for word problems in English. It is therefore natural to ask how the used principles would work for inputs in other languages, especially if the language is in some sense more difficult for analysis, for example, due to more flexible word order.

[1] https://wolframalpha.com

In this paper, we present a study of methods suitable for processing word problems in Czech, but we also present some new linguistically independent ideas. As no suitable annotated data has existed so far, it was necessary to build a new dataset. We have collected 500 word problems intended for the first three classes of elementary schools. These word problems can be solved by substituting numbers from the problem to an arithmetic expression. Hence, the problems are not so complex as some of those in Alg514 and Dolphin18K that also include problems whose solution requires to assemble and solve systems of equations. Nevertheless, we consider our dataset to be sufficient for an initial study, which can serve as a step towards methods processing more complex word problems. Furthermore, a solver working reliably on the considered domain might be appreciated by the youngest pupils, especially if there is a possibility to provide them a step by step explanation of how to solve a given problem. This should justify enough the focus of our research.

We touch all three principles discussed before and propose two solvers. The first one is based on a support vector machine (SVM) classifier and it is further extended by incorporating the patterns-based classification to handle some of the easier inputs. The advantage of this solver is that it does not require any knowledge of semantic (unlike the system from [5]). The accuracy achieved on the created dataset is 75%. The second solution builds on syntactic analyses. It first calls UDPipe [11] to create syntactic dependency trees for all sentences, then it traverses the trees, tries to retrieve information on logical entities encoded in the problem, processes the question part and computes the result. Unlike the first solution, the extracted information could be used to output details on how to solve the problem. On the other hand, the accuracy is much worse when compared to the first solution.

The collected dataset and implementation of the proposed methods are freely available[2].

The content of the paper is structured as follows. Section 2 describes our dataset of 500 word problems in Czech. The next three sections describe and evaluate the proposed principles. Section 3 discusses how to extract and use patterns, Section 4 explains the use of SVM, Section 5 focuses on retrieving information from dependency trees produced by UDPipe. Finally, Section 6 summarizes the achieved results, gives additional thoughts on the presented methods and outlines possible future work.

2 WP500CZ dataset

We have created a dataset called WP500CZ comprising of 500 word problems taken from the schoolbook [10]. We have further divided the whole dataset into training and testing subsets, denoted WP500CZ-train and WP500CZ-test, respectively, each comprising 250 word problems. The word problems were transformed into an electronic form partially by using OCR, but manual editing and several corrections of OCR outputs were also needed. Each word problem was annotated by the expected answer (a non-negative integer) and also the arithmetic expression used to derive the answer.

Let us list four samples from the dataset, together with their English translations:

W_1 *Jana má 4 pastelky nové a 3 pastelky staré. Kolik pastelek má Jana?*
(Jane has 4 new crayons and 3 old crayons. How many crayons does Jane have?)

W_2 *Pavel má 120 Kč. Ota má 80 Kč. Kolik korun mají dohromady?*
(Paul has 120 CZK. Otto has 80 CZK. How many Czech korunas do they have together?)

W_3 *Petr přečetl 9 knih. Eva přečetla 6 krát více knih. Kolik knih Eva přečetla?*
(Peter read 9 books. Eve read 6 times more books. How many books did Eve read?)

W_4 *Maminka koupila 20 tvarohových koláčků a 15 makových koláčků. Děti 8 koláčů snědly. Kolik koláčů zůstalo?*
(Mom bought 20 cheesecakes and 15 poppy seed cakes. The children ate 8 cakes. How many cakes are left?)

Word problem W_1 is represented in a text file as follows.

```
Jana má 4 pastelky nové a 3 pastelky staré.
Kolik pastelek má Jana? | 7 | NUM1 + NUM2
```

As it can be seen, there are three parts separated by the pipe characters. The first part is a word problem, the second part is the answer and the third part is the expression solving the problem. Variables in the expression are strings of the form *NUMx* where *x* is the order number of the referenced numeric value in the assignment.

Note that this approach has some limitations. For example, it does not allow to reference numbers expressed by words like *day* (has 24 hours) or *June* (has 30 days). However, our dataset does not contain word problems relaying on these translations of words to numbers.

A word problem itself consists of two parts – a *word problem assignment* (WPA-part) and a *question* (QUE-part). Each word problem in WP500CZ fulfills that its QUE-part is a simple sentence asking for one detail and the expression which gives the answer is exclusively composed of addition, subtraction, multiplication and division operations.

Task 1. *Given a word problem W, the goal of an automatic word problem evaluation is to find an expression E solving W.*

Remark 1. *We say that W is of* type *E.*

[2] https://github.com/hkad98/problem_solver

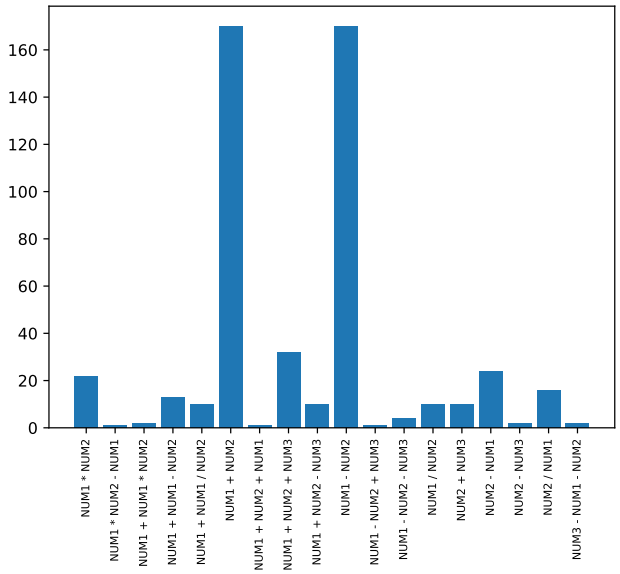

Figure 1: Histogram of word problem types in WP500CZ.

Types of the word problem in WP500CZ and their frequencies are depicted in Figure 1.

As it can be seen, approximately one-third of the instances are of type NUM1 + NUM2, one-third are of type NUM1 − NUM2, and the remaining instances are of 11 different types (note that these types include NUM2 − NUM1, which is considered as a type different to NUM1 − NUM2). This corresponds to the distribution of word problem types in [10].

3 Pattern-based classification

In this section, we describe a classification of word problems based on *patterns* extracted from a training dataset. This approach is able to handle reliably a subset of instances (we will show that the patters extracted from WP500CZ-train apply to 54% instances of WP500CZ-test). For this reason, we later combine the method with machine learning.

A pattern is a sequence of word lemmas and classes. Patterns are extracted from WPA-parts and QUE-parts separately and we call them *WPA-patterns* and *QUE-patterns*, respectively. Given a word problem W, it matches a WPA-pattern (QUE-pattern) P if the WPA-part (QUE-part) of W contains a subsequence of words whose lemmas/word classes are in the one-to-one correspondence with the elements of P, preserving the order of the elements.

To illustrate the form of patterns, we give two examples of WPA-patterns followed by two examples of QUE-patterns.

```
NOUN a NUM krát málo NOUN | NUM1 / NUM2
(NOUN and NUM times little NOUN)
být o NUM NOUN vysoký | NUM1 + NUM2
(to be NUM NOUN high)

kolik NOUN zbýt NOUN v NOUN | NUM1 - NUM2
(how many NOUN to left NOUN in NOUN)
kolik NOUN NOUN zbýt | NUM1 - NUM2
(how many NOUN NOUN to left)
```

Note that each line contains a pattern separated by the pipe character from the expression used for classification when the pattern is matched. In addition, NOUN and NUM in the pattern part represent word classes (noun and numeral).

We propose an algorithm which obtains the patterns automatically. Let us describe how it extracts e.g. WPA-patterns. Let $\mathscr{S}_1, \ldots, \mathscr{S}_k$ be the partition of training dataset by word problem types (i.e., each $\mathscr{S}_i$ consists of word problems of the same type). Maps $\mathscr{P}_1, \ldots, \mathscr{P}_k$ are allocated to collect the found patterns (stored as keys) and their number of occurrences (stored as values). All these maps are initially empty. In the end, each $\mathscr{P}_i$ contains patterns extracted from $\mathscr{S}_i$.

The algorithm has three phases. First, it process the subsets $\mathscr{S}_i$ one by one. For each $\mathscr{S}_i$, it iterates through all pairs $U, V \in \mathscr{S}_i$, $U \neq V$. Let u and v be the WPA-part of U and V, respectively. Moreover, for a sequence of words $\mathbf{x} = (x_1, \ldots, x_s)$, let $\ell(\mathbf{x})$ be the sequence $(x'_1, \ldots, x'_s)$ where

$$x'_i = \begin{cases} \text{NOUN} & \text{if } x_i \text{ is a noun,} \\ \text{ADJ} & \text{if } x_i \text{ is an adjective,} \\ \text{NUM} & \text{if } x_i \text{ is a numeral,} \\ \text{the lemma of } x_i & \text{otherwise.} \end{cases}$$

A longest common subsequence of $\ell(u)$ and $\ell(v)$, denoted $P_{u,v}$, is found. If $\mathscr{P}_i$ does not yet contain $P_{u,v}$, the sequence is inserted to $\mathscr{P}_i$ and its number of occurrences is set to 1. If $P_{u,v}$ is already in $\mathscr{P}_i$, the number of occurrences of $P_{u,v}$ is incremented.

In the second phase, the algorithm discards sequences that were not detected frequently. It iterates through sets $\mathscr{P}_i$ and deletes from $\mathscr{P}_i$ each sequence P for which the number of occurrences does not exceed $\tau \cdot |\mathscr{P}_i|$ where $\tau \in (0,1)$ is a suitable constant.

Finally, the third phase of the algorithm discards every sequence P which is shared by two distinct sets $\mathscr{P}_i$, $\mathscr{P}_j$.

In terms of the time complexity, the first phase is the most demanding. Let us assume that the number of words in the WPA-part as well as QUE-part of considered word problems is at most N. Moreover, let $\mathscr{S} = \bigcup_{i=1}^{k} \mathscr{S}_i$. Since the basic dynamic programming algorithm finding the longest common subsequence for two sequences of length at most N works in $O(N^2)$ time [3], the time complexity of the first phase, denoted $T_1(\mathscr{S})$, satisfies

$$\begin{aligned} T_1(\mathscr{S}) &= O\left(\sum_{i=1}^{k} \binom{|\mathscr{S}_i|}{2} N^2\right) = O\left(N^2 \sum_{i=1}^{k} |\mathscr{S}_i|^2\right) \\ &= O\left(N^2 |\mathscr{S}|^2\right). \end{aligned}$$

When WPA-patterns are available, they are ordered into a sequence $\mathscr{P}_{\text{WPA}} = P_1, \ldots, P_n$ which fulfills that a pattern P_j is a proper subsequence of a pattern P_i if and only if $j > i$. Analogously, the extracted QUE-patters are ordered into a sequence $\mathscr{P}_{\text{QUE}}$ with the same property. If a word problem W with a WPA-part u and QUE-part v is given, the pattern-based classification iterates through elements of $\mathscr{P}_{\text{QUE}}$ until it finds a pattern P_{QUE} that matches u, or the end of $\mathscr{P}_{\text{QUE}}$ is reached. If P_{QUE} is found, the classification result is the expression assigned to P_{QUE}. Otherwise the algorithm iterates through $\mathscr{P}_{\text{WPA}}$ and tries to find a pattern P_{WPA} matching v, which again determines the classification result. No result is returned if P_{QUE} neither P_{WPA} is found.

3.1 Experiments

The described procedures work with lemmatized inputs. We thus used corpus SYN2015 [8] to build a vocabulary where keys are words extracted from the corpus and values are pairs consisting of a word class and lemma. The ambiguity of some words is resolved by taking into account their frequency. The built vocabulary has about 1 million items.

The proposed pattern extraction algorithm was implemented in Python 3. It returned 28 WPA-patterns and 8 QUE-patterns for instances in WP500CZ-train. The running time was 4.9 [s] on a notebook equipped with Intel(R) Core(TM) i5-5250U CPU @ 1.60GHz, 8GB RAM and MacOS.

Table 1 shows the efficiency of the subsequent pattern-based classification. Three possible classification outcomes are distinguished – a correct classification, incorrect classification and no result when the classification algorithm does not return any expression.

	correct	incorrect	no result
WP500CZ-train	85	10	155
WP500CZ-test	107	8	135

Table 1: The accuracy of pattern matching on WP500CZ.

The collected patterns were detected in nearly one-half of the word problems in WP500CZ-test (115 out of 250). In those cases, the reliability of classification was quite high, achieving 93% accuracy (107 correct classifications). Interestingly, the accuracy was better than in the case of training data, however, there were fewer instances matching a pattern.

4 Machine learning

This section proposes a solver based on machine learning. The main idea is to come up with a suitable transformation of a word problem into a numeric vector of fixed dimension (a so-called *feature vector*) and train a multi-class support vector machine (SVM) [1] to classify the inputs by their types (expressions). In the experimental part, we also show that it is worth to combine results produced by SVM and the pattern matching from Section 3.

We decided to use SVM because it is a classifier able to cope with high-dimensional data and it is not prone to overfitting. A flexibility is achieved through a kernel choice, which gives a possibility to linearly separate non-linearly separable data by mapping them into a higher dimension. It also does not require a large dataset for training, unlike neural networks.

Our method of representing word problems by numeric vectors is semantic-independent. It is based on histograms of words occurring in word problems of the same type. For this purpose we define

1. a frequency function $\omega(w, p, E)$ which for a word w in its canonical form, $p \in \{\text{WPA}, \text{QUE}\}$ and a word problem type E is equal to the number of word problems of type E in the training dataset that contain a word of the canonical form w in the p-part, and

2. a weight function $\alpha(c, p)$ which for a word class c and $p \in \{\text{WPA}, \text{QUE}\}$ is equal to an integer determining the importance of word class c in the classification process.

The values of the frequency function are calculated based on the training dataset. For example, we obtained the following values for verb *mít* (*to have*) and adverb *dohromady* (*together*) from WP500CZ-train:

$$\omega(\textit{mít}, \text{WPA}, \text{NUM1} + \text{NUM2}) = 27, \quad (1)$$
$$\omega(\textit{mít}, \text{QUE}, \text{NUM1} + \text{NUM2}) = 28, \quad (2)$$
$$\omega(\textit{mít}, \text{WPA}, \text{NUM1} - \text{NUM2}) = 25, \quad (3)$$
$$\omega(\textit{mít}, \text{QUE}, \text{NUM1} - \text{NUM2}) = 11, \quad (4)$$
$$\omega(\textit{dohromady}, \text{WPA}, \text{NUM1} + \text{NUM2}) = 0, \quad (5)$$
$$\omega(\textit{dohromady}, \text{QUE}, \text{NUM1} + \text{NUM2}) = 6, \quad (6)$$
$$\omega(\textit{dohromady}, \text{WPA}, \text{NUM1} - \text{NUM2}) = 0, \quad (7)$$
$$\omega(\textit{dohromady}, \text{QUE}, \text{NUM1} - \text{NUM2}) = 0. \quad (8)$$

Values of the weight function were learned by a genetic algorithm, taking SVM accuracy as the objective function.

When the functions ω and α are known, we transform a word problem W to a numeric vector $\mathbf{x}$ using the following algorithm.

1. Components of $\mathbf{x}$ are indexed by types of word problems in the training dataset (thus the vector dimension coincides with the number of SVM classes) and all of them are initially set to 0.

2. Words of W are taken one by one. Let the current word be of a lemma w and word class c. Moreover, let it be from a p-part of W. For each word problem type E, it is checked if $\omega(w, p, E)$ is defined and the expression E evaluates to a non-negative integer after

substituting the numeric values from W to E (i.e., E is feasible to W). If true, then the E-component of $\mathbf{x}$ is increased by $\alpha(c,p)\cdot\omega(w,p,E)$.

Let us demonstrate how the frequencies (1)-(8) are used to calculate the numeric vector for word problem W_1 in Section 2. For simplicity, assume that word problems in the training dataset are only of types NUM1 + NUM2 and NUM1 − NUM2, and that $\alpha(c,p)=1$ for all c and p (i.e., we will ignore the weigh function).

First, we observe that both considered expressions evaluate to a positive integer when the numbers 4 and 3 from W_1 are substituted for NUM1 and NUM2, respectively, hence both expressions are feasible to W_1. Since the verb *mít* is in the WPA-part as well as QUE-part of W_1, it contributes by frequencies (1) and (2) to vector component NUM1 + NUM2, and, analogously, by frequencies (3) and (4) to the component NUM1 − NUM2. The adverb *dohromady* appears only in the QUE-part of W_1, hence it contributes by frequency (6) to component NUM1 + NUM2 (note that frequency (7) is zero, hence it does not contribute to component NUM1 − NUM2). If we sum the listed contributions, we obtain the vector

$$\{\text{NUM1}+\text{NUM2}:61,\ \text{NUM1}-\text{NUM2}:36\}.$$

Remark 2. *It is important to notice that the proposed form of vectors suggests to use a simple classifier which only finds a component of maximum value and returns the expression assigned to it. We will consider this strategy in the experimental part where it will be shown that it performs considerably worse than SVM.*

As the testing data might have a different distribution than the training data, we normalize the vectors before passing them to SVM. For the SVM training stage, the normalization modifies the training set vectors so that the mean value is 0 and the variance is 1. The same transformation is applied to each input vector before it is classified.

Figure 2 shows a visualization of an SVM trained for word problems of types NUM1 + NUM2 and NUM1 − NUM2.

4.1 Experiments

In the next paragraphs we use the following abbreviations to denote the proposed solvers:

- sSVM is SVM-based solver,
- sMAX is the classifier suggested by Remark 2,
- sSVM+PAT is sSVM combined with the pattern matching, and
- sMAX+PAT is sMAX combined with the pattern matching.

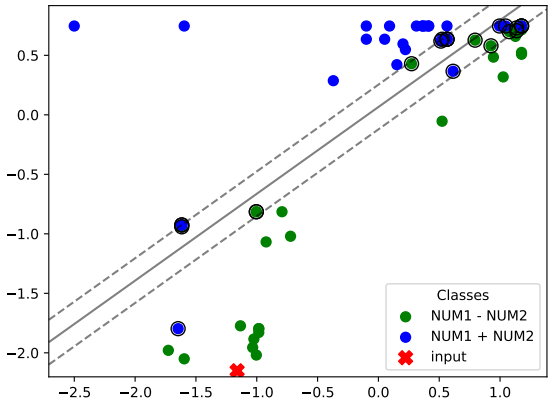

Figure 2: A SVM trained for normalized 2-dimensional vectors obtained for several word problems of NUM1 + NUM2 and NUM1 − NUM2 types. The circled samples are the support vectors. The input displayed in red represents the normalized vector of an input word problem to be classified.

The combined solvers sSVM+PAT and sMAX+PAT try to apply the pattern matching first. If it does not return a result, then sSVM and sMAX is called to complete the classification.

We used *scikit-learn*[3] library to implement sSVM. Radial basis function kernel was chosen as the best performing one among tested kernels.

The accuracy of the solvers can be compared in Table 2 and Table 3. The combined solver sSVM+PAT was the best solver on the testing dataset. In addition, the strategy of sMAX solver is inferior to the SVM classification.

	sSVM	sMAX
WP500CZ-train	87.6%	58.0%
WP500CZ-test	61.1%	51.0%

Table 2: The accuracy of sSVM and sMAX.

	sSVM+PAT	sMAX+PAT
WP500CZ-train	86.8%	69.2%
WP500CZ-test	74.9%	66.1%

Table 3: The accuracy of the combined solvers.

All instances of WP500CZ were evaluated in 3.66 [s] by sSVM+PAT and 3.15 [s] by sMAX+PAT (the running environment was identical to that one described in Subsection 3.1).

Figures 3 and 4 show distributions of errors per word problem type for sSVM+PAT (note that only word problem types present in the training dataset are considered).

[3] `https://scikit-learn.org/stable/`

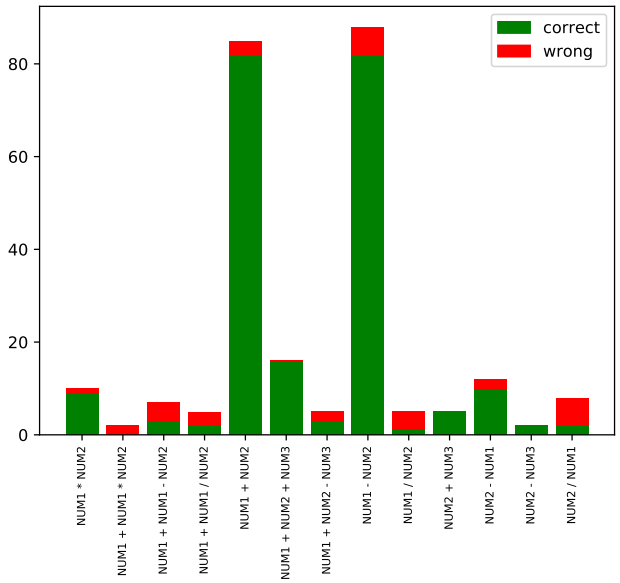

Figure 3: The accuracy of sSVM+PAT on WP500CZ-train.

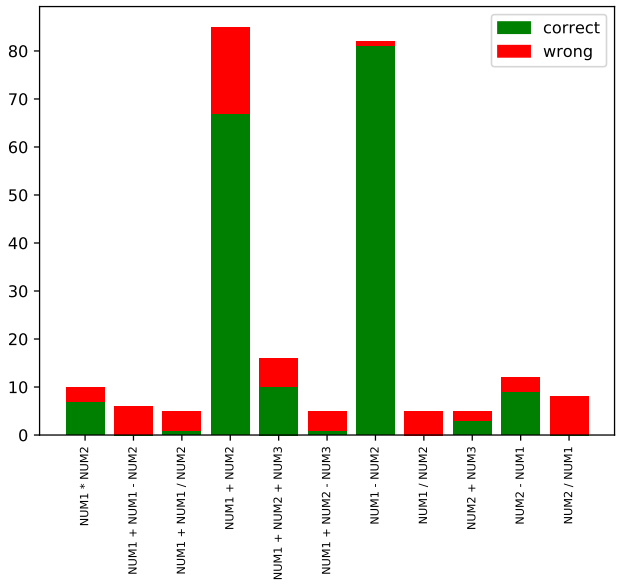

Figure 4: The accuracy of sSVM+PAT on WP500CZ-test.

5 Syntactic analysis

The so far presented solvers worked as classifiers but they were not suitable to be used for a step by step explanation of how to derive a solution to a given word problem. Here we give a high-level description of a method based on syntactic analysis, which creates an internal representation of the input problem. Although our current implementation of this method lags behind the SVM-based solver, we think that there is a room for improvements and it is thus worth to analyze causes of errors.

The method uses UDPipe (with Universal Dependencies 2.5 Models 2019-12-06) to process sentences of a given word problem. UDPipe returns information on word classes, sentence elements and their dependencies in CONNL-U format[4]. This information allows to construct a dependency tree.

We apply several algorithms working over the tree to retrieve structured information on entities encoded in the assignment. For each number N, it is extracted which entity E_1 the number refers to and which entity E_2 is the owner of E_1.

Let us demonstrate it for the sentence:

Alenka si koupila 5 čokoládových bonbónů.
(Alice bought 5 chocolate candies.)

Figure 5 shows the dependency tree constructed from the data returned by UDPipe.

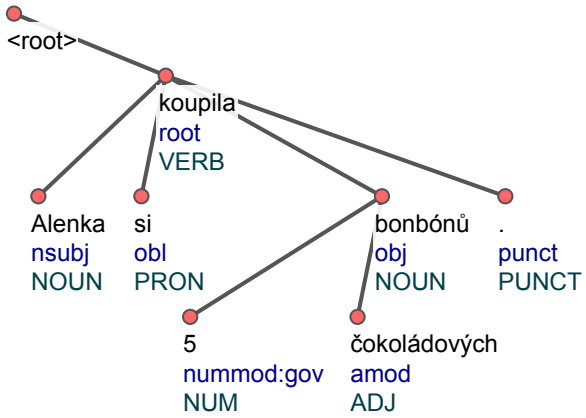

Figure 5: A dependency tree example.

The structure which is supposed to be extracted from the tree is visualized in Figure 6.

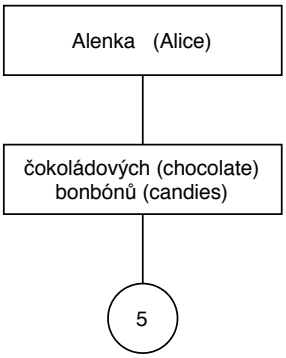

Figure 6: Extracted information.

A given word problem is solved in three phases designed to handle a variety of assignment formulations:

1. *Preprocessing* phase splits compound sentences and evaluates each part separately. It also replaces

[4] https://universaldependencies.org/format.html

phrases like *3 times more than* by numbers, i.e., if applicable, it computes intermediate results and passes them to the next phases.

2. *Structure creation* phase connects numbers and entities. This is done by traversing the dependency tree from nodes representing numbers. When a structure is extracted, a vocabulary of verbs that represent a subtraction is used to adjust sign of the number (i.e., unlike in the previous methods, verbs semantic is taken into account here).

3. *Evaluation* phase checks the question part and guesses which entity E the word problem queries for. The answer is derived based on the extracted structures that relate to E.

5.1 Experiments

To tune the method and to analyze errors encountered during testing, we considered only a subset of WP500CZ consisting of 150 word problem instances, denoted here as WP150CZ. The reason is that to analyze error types required a manual approach, it would, therefore, be too laborious to work with the entire dataset.

The method achieved 76% accuracy on the training part of WP150CZ (formed of 75 instances) and 67.1% accuracy on the testing part. However, the performance was much worse when the method was evaluated against all instances of WP500CZ (which took 23.5 [s]), solving correctly 41% of word problems. This indicates that WP500CZ is more diverse than WP150CZ and the rules designed based on WP150CZ are not general enough.

Table 4 shows detailed analysis of errors made by the method on WP150CZ. The analysis reveals that UDPipe itself is a non-negligible cause of errors since it often returns partially incorrect outputs and our method is not able to recover from these errors.

	UDPipe	preproc.	creation	eval.	total
train	7	3	5	3	18
test	9	5	9	1	24

Table 4: Errors by type made by the solver on WP150CZ.

6 Conclusion

Our paper is an initial study of an automatic evaluation of math word problems in Czech. We have reviewed the state-of-the-art methods for inputs in English and proposed three solvers, based on pattern matching, SVM and syntactic analysis. We also contributed by a dataset of 500 annotated word problems in Czech.

The main findings and new ideas of our work can be summarized as follows.

- The pattern extraction is a less efficient approach when used as a standalone solver for word problems entered in a language with flexible word order. It is applicable only to a portion of inputs. However, we showed that it can be well used in a combination with other solvers since a matched pattern usually results in a correct classification. Another success is that we managed to implement a fully automatic pattern extraction.

- A semantic-independent representation of word problems, considering only frequencies of words, turned out to be sufficient to obtain a relatively accurate classifier. The trained SVM performed better than the straightforward sMAX classifier. This might be interpreted as meaning that the SVM was able to find nontrivial relationships in the used features vectors.

- We have tested the suitability of existing linguistic tools for the studied task. The method based on syntactic analysis processed dependency trees produced by UDPipe. The method was not able to recover from errors occasionally done by UDPipe. Improving the accuracy of UDPipe is thus supposed to result in a higher accuracy of the method.

The accuracy of sSVM+PAT on WP500CZ (achieving 74.9%) was not too far from the accuracy of the state-of-the-art solvers on Alg514 (achieving 70% or 82.5% by methods from [9] or [6], respectively), it must, however, be said that the complexity of some instances in Alg514 is out of the scope of our methods. To make a fairer comparison with the state-of-the-art would require to create an English version of WP500CZ, or to adapt the state-of-the-art solvers to inputs in Czech.

It must also be noted that WP500CZ is a small dataset (however, as we explained in the introduction, the problem has not been studied for non-English languages by others and there are no publicly available datasets of word problems for e.g. Slavic languages). A worse performance can be expected if sSVM+PAT is applied to larger and more diverse datasets. A broad expansion of the dataset and a creation of multilingual versions can, therefore, be identified as the goal of our further work. Except much more thorough testing, it would allow us to apply deep learning which is expected to improve the quality of the results.

In addition to our future plans, we also hope that the presented research can inspire others to consider non-English inputs for the studied problem.

Acknowledgment

We thank anonymous reviewers whose suggestions helped improve this paper. Our work was supported by the Czech Science Foundation grant no. 19-21198S.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[2] Daniel G. Bobrow. Natural language input for a computer problem solving system. Technical report, USA, 1964.

[3] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[4] Edward A. Feigenbaum and Julian Feldman. *Computers and Thought*. McGraw-Hill, Inc., USA, 1963.

[5] Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533, Doha, Qatar, October 2014. Association for Computational Linguistics.

[6] Danqing Huang, Jing Liu, Chin-Yew Lin, and Jian Yin. Neural math word problem solver with reinforcement learning. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 213–223, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics.

[7] Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. How well do computers solve math word problems? Large-scale dataset construction and evaluation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 887–896, Berlin, Germany, August 2016. Association for Computational Linguistics.

[8] Michal Křen, Václav Cvrček, Tomáš Čapka, Anna Čermáková, Milena Hnátková, Lucie Chlumská, Dominika Kováříková, Tomáš Jelínek, Vladimír Petkevič, Pavel Procházka, Hana Skoumalová, Michal Škrabal, Petr Truneček, Pavel Vondřička, and Adrian Zasina. SYN2015: representative corpus of written Czech, 2015. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

[9] Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 271–281, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[10] Marie Reischigová. *Matematika na základní a obecné škole ve slovních úlohách*. Pansofia, 1996. In Czech.

[11] Milan Straka and Jana Straková. Tokenizing, POS tagging, lemmatizing and parsing UD 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[12] Dongxiang Zhang, Lei Wang, Luming Zhang, Bing Dai, and Heng Shen. The gap of semantic parsing: A survey on automatic math word problem solvers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP:1–1, 04 2019.