

1-1-2022

## Asking the right questions to solve algebraic word problems


EGE YİĞİT ÇELİK

ZEYNEL ORULLUOĞLU

RIDVAN MERTOĞLU

SELMA TEKİR

Follow this and additional works at: <https://journals.tubitak.gov.tr/elektrik>

 Part of the [Computer Engineering Commons](#), [Computer Sciences Commons](#), and the [Electrical and Computer Engineering Commons](#)

---

### Recommended Citation

ÇELİK, EGE YİĞİT; ORULLUOĞLU, ZEYNEL; MERTOĞLU, RIDVAN; and TEKİR, SELMA (2022) "Asking the right questions to solve algebraic word problems," *Turkish Journal of Electrical Engineering and Computer Sciences*: Vol. 30: No. 7, Article 13. <https://doi.org/10.55730/1300-0632.3962>

Available at: <https://journals.tubitak.gov.tr/elektrik/vol30/iss7/13>

This Article is brought to you for free and open access by TÜBİTAK Academic Journals. It has been accepted for inclusion in Turkish Journal of Electrical Engineering and Computer Sciences by an authorized editor of TÜBİTAK Academic Journals. For more information, please contact [academic.publications@tubitak.gov.tr](mailto:academic.publications@tubitak.gov.tr).

## Asking the right questions to solve algebraic word problems

Ege Yiğit ÇELİK<sup>✉</sup>, Zeynel ORULLUOĞLU<sup>✉</sup>, Rıdvan MERTOĞLU<sup>✉</sup>, Selma TEKİR\*<sup>✉</sup>

Department of Computer Engineering, Izmir Institute of Technology, Izmir, Turkey

Received: 20.04.2022

Accepted/Published Online: 08.10.2022

Final Version: 28.11.2022

**Abstract:** Word algebra problems are among challenging AI tasks as they combine natural language understanding with a formal equation system. Traditional approaches to the problem work with equation templates and frame the task as a template selection and number assignment to the selected template. The recent deep learning-based solutions exploit contextual language models like BERT and encode the natural language text to decode the corresponding equation system. The proposed approach is similar to the template-based methods as it works with a template and fills in the number slots. Nevertheless, it has contextual understanding because it adopts a question generation and answering pipeline to create tuples of numbers, to finally perform the number assignment task by custom sets of rules. The inspiring idea is that by asking the right questions and answering them using a state-of-the-art language model-based system, one can learn the correct values for the number slots in an equation system. The empirical results show that the proposed approach outperforms the other methods significantly on the word algebra benchmark dataset alg514 and performs the second best on the AI2 corpus for arithmetic word problems. It also has superior performance on the challenging SVAMP dataset. Though it is a rule-based system, simple rule sets and relatively slight differences between rules for different templates indicate that it is highly probable to develop a system that can learn the patterns for the collection of all possible templates, and produce the correct equations for an example instance.

**Key words:** Math problem solver, question generation and answering, algebraic word problems

### 1. Introduction

A word algebra problem is a simple mathematical problem written out as a short scenario in real life. It describes a problem using natural language and requires people to solve it using math. Humans can solve easy algebraic word problems in a matter of seconds. However, they pose a challenge for a machine since the solution involves the formation of algebraic equations from the input text and then the solving of this linear system of equations.

Automatic solving of math word problems has been recognized as a fair challenge similar to the Turing test [1]. One can control the difficulty of questions, which allows a gradual assessment of the performance. At its core, the task requires language understanding, including world knowledge, implicit for humans but hard to realize for machines. World knowledge is scattered in different proportions to these problems, rendering the training experience insufficient for solving unseen problems.

Until recently, traditional machine learning approaches have been adopted to tackle word algebra problems, and the benchmark datasets have been relatively smaller in size and included a limited number of types. The advances in deep learning have led to new deep learning-based approaches in math problem solvers. Generally, deep neural solvers [2–5] adopt encoder-decoder architectures. The encoder phase serves as a candidate

\*Correspondence: selmatekir@iyte.edu.tr

equation generator, and the subsequent decoder ranks candidate equations and returns the highest-ranked one as a result. As for benchmarks, more challenging large datasets [6] started to be released.

Kushman et al.'s [7] baseline work uses a template-based solver method for word algebra problems. The authors also collect a new corpus (alg514) of 514 word algebra problems and their ground-truth equation systems. Their approach can solve these equation set problems with acceptable accuracies. Zhou et al. [8] introduce a more efficient solver by changing the objective function of Kushman et al.'s [7] and also achieve higher accuracy on the alg514 dataset.

In datasets, each problem has its associated math equation, provided as a template. Templates represent the types of algebra word problems. Template-based solvers involve a two-step process. Firstly, they classify a given problem into one of the predefined templates. This process is called template selection. In the number assignment stage, the solver should fill in the empty number slots in the equation template correctly by referring to the math problem's text. Then, the solving of fully-assigned equation templates is a trivial task.

Given the boosted performances in NLP tasks especially in question answering due to the contextual language models, e.g., BERT [9], we hypothesize that one can extract relevant numerical quantities out of a math problem by asking the right questions and answering them. Moreover, we argue that the level of performance indicates to what degree the state-of-the-art question answering systems address world knowledge.

This work presents a novel template-based solver. The proposed approach generates questions from the math problem's text using a rule-based question generation system [10] to form a basis to extract some useful information out of the math problem. Then, a state-of-the-art question answering model answers those questions. Afterward, we process the answers through custom sets of rules to fill in the slots in the corresponding equation templates. To the best of our knowledge, this is the first work that solves math problems using a combination of automatic question generation and answering and custom sets of rules for templates.

We test our approach on the benchmark dataset alg514 for word algebra problems, and AI2 [11] for arithmetic word problems. The experimental results show that the proposed approach performs better on both problem types. This performance can imply that language use and world knowledge do not vary much in the template-specific problems. We also measure our solver's accuracy on the challenging SVAMP dataset against the leading state-of-the-art sequence to sequence, sequence to tree, and graph to tree solvers, where it is also better. This result reflects that our pipeline's question answerer module can capture problems' contexts and is robust against the types of variations in SVAMP. Although we work with template-specific custom rule sets for solving problems, once we have helpful answers due to our right questions, extending the solution set with a new template does not take much effort. It can mean that the problem text language and knowledge level variations among templates are also somehow limited.

In the remainder of the paper, we firstly describe related work (Section 2), the evolution of math problem solvers. In Section 3, we explain the proposed approach's pipeline along with its components (preprocessing, question generation and answering, and custom rule sets for number assignment). Afterward, we provide both quantitative and qualitative evaluation (Section 4). Finally, we conclude the paper with further remarks for future work.

## 2. Related work

There are many research papers on the subject of math problem solvers. The earlier work has been template-based learning frameworks, while the latest solvers rely on deep learning pipelines. Zhang et al. [12] provide a comprehensive survey on this subject and categorize the existing work from different perspectives.

The first strong baselines for math problem solvers are due to template-based methods [7, 8, 13, 14]. The pioneering work of Kushman et al. [7] uses a template-based solver for word algebra problems. A template-based classifier is expected to learn alignments between the textual form of the problem and its set of algebraic equations. An improved version of this method with more efficiency and higher accuracy can be found in the work of Zhou et al. [8].

Kushman et al. [7] also provide a benchmark dataset for word algebra problems (alg514), which is considered a strong foundation. This dataset includes many problems with equation sets as can be seen in the example in Figure 1.

```

"sQuestion": "You are selling tickets for a high school play. Student tickets cost 4 dollars and general
admission tickets cost 6 dollars. You sell 525 tickets and collect 2876 dollars. How many students tickets
and general admission tickets did you sell?",
"IEquations": [
"4.0*student_ticket+6.0*admission_ticket=2876.0",
"student_ticket+admission_ticket=525.0"
],
"ISolutions": [
137.0,
388.0]

```

**Figure 1.** An example math problem from the alg514 dataset.

The example problem has the numbers 4, 6, 525, and 2876 and nouns "student ticket," "admission ticket," which are placed inside the equation set template. To perform this substitution (alignment), the earlier work [7, 8] use some descriptive features from the problem text to pick a better alignment. However, this feature selection task is human-engineered.

The recent deep neural network-based solvers primarily adopt encoder-decoder architectures. The encoder phase serves as a candidate equation generator, and the subsequent decoder ranks candidate equations and returns the highest-ranked one as a result. The first deep neural solvers [2, 4, 5, 15] have used sequential neural representations, e.g., LSTM and GRU. The more recent studies, such as [3], utilize tree-based recursive neural networks to capture the intrinsic structure of the equations to solve math problems better. Moreover, their system inputs the operation embeddings, which helps distinguish between different operators, thus solves more complex arithmetic word problems.

Lee et al.'s solver [16] provides a reconciliation of template-based methods with deep-learning frameworks. The approach is called template-based multitask generation (TM-generation). This approach uses the state-of-the-art language model ELECTRA to encode the input question. It passes the question to the multitask environment normalized by replacing numbers with variables. The multitasking architecture consists of two classification tasks: template classification and operator classification. The former assigns a predefined template number to the given problem while the latter predicts operators for the operator fields in the template. The approach is tested on the MAWPS and Math23k datasets. The authors report the state-of-the-art accuracy scores of 85.2% and 85.3% for MAWPS and the Math23k datasets, respectively.

Another approach by Mandal and Naskar [17] classifies math problems according to their problem types like change, combine, and compare, and then a subtype and the operations are involved and continues with a rule-based system to solve them. They test the performance of their approach on the AddSub, and SingleOp subsets of the MAWPS corpus [18] and report that their solver produces new state-of-the-art results.

The recent deep learning approaches are primarily sequence to sequence, sequence to tree, or graph to tree solvers.

Lin et al. [19] improve seq2seq math word solvers by proposing a hierarchical word encoder and an expression decoder that is based on a point network to copy numbers to the expression and make the inference. The encoder part aims to understand the textual problem by leveraging intraclause structures through dependency parsing and interclause relations by using positional embeddings for clauses and a self-attention mechanism between clauses. The authors show that their approach produces better results than baselines on Math23K and MAWPS [18] datasets.

Hong et al. [20] propose a weakly-supervised solver for math word problems where the training samples include the problem text and the final answer excluding the solution expression. The approach uses a neural tree-based solver similar to the goal-driven tree-structured solver (GTS) of Xie and Sun's [21]. However, it introduces a fixing mechanism, which, relying on the correct final answer, propagates the error from the root node to the leaves to resolve the issue with an incorrect solution. Then, the corrected expression tree is added to the training set to make further the model learn. The solver is tested on the Math23K dataset, and the results confirm that it significantly outperforms the weakly supervised baselines. The approach supports the generation of diverse solutions for given problems as an advantage over the fully supervised models.

Wu et al. [22] adopt a sequence-to-tree network to better capture numerical values in math word problems. Their numerical attribute prediction module further helps understand the global relationship among the numerical values. Their solver improves the performance on Math23K and Ape210K datasets.

In their math word problem pipeline, Yu et al. [23] combine two encoders with a tree-structured decoder. Their first encoder is a pretrained language model, a source of external knowledge. As for the second encoder, its responsibility is to model the hierarchical context among words and sentences. The experimental results validate the effectiveness of their approach by the performance boost on Math23K and MAWPS benchmarks.

Liang et al. [24] propose the MWP-BERT solver that includes a BERT-based encoder and a tree-based decoder to solve math word problems. They intend to inject world knowledge using pretrained BERT. To meet this objective, they propose a prompt-based triggering into BERT. As a result, their model achieves higher accuracy on the benchmark datasets. Zhang et al. [25], on the other hand, combines a graph-based encoder with a tree-based decoder and is one of the state-of-the-art solvers.

In the evolution of math problem solvers, introducing benchmark corpora is crucial and leads to the development of upcoming methods. Such a work belongs to Miao et al. [26], where they suggest a new dataset named ASdiv, which contains 2305 problems in 6 different difficulty levels. Many approaches, including [7], have been tested on this dataset. The average accuracy of all the tested methods has been found as 36%. Thus, it presents a new challenging dataset in this area.

Patel et al. [6] give their reflections on the status of math word problem datasets and the state-of-the-art solvers. They argue that a shallow-level understanding of the problem text is sufficient to perform well on the existing benchmarks. They further claim that even the question text/word order is unnecessary to solve a large part of the math word problems. Accordingly, they propose a new challenge dataset called SVAMP enriched with question sensitivity, reasoning ability, and structural invariance variations. The authors also report state-of-the-art performances on their dataset using three solvers (Seq2Seq [15], GTS [21], Graph2Tree [25]) when they are trained with RoBERTa [27] embeddings.

The proposed approach is similar to the template-based methods as it works with a template and fills in the number slots. Nevertheless, it has contextual understanding because it adopts a question generation and

answering pipeline to create tuples of numbers, to finally perform the number assignment task by custom sets of rules.

We generate the questions using the rule-based system of Keklik et al. [10] since rule-based systems generally work with template-based rules to develop questions, and this fits the circumstances of the math word problem datasets. On the other hand, there is less control over the generated questions in neural question generators. In the question-answering phase, we use a BERT-based [9] system to answer the generated questions from the previous stage.

The question generator and the question answering system are combined to extract helpful information for math problem solvers. As far as we know, no math problem solver uses a similar approach.

### 3. Approach

The proposed approach is a rule-based math problem solver. As we define template-specific rules to instantiate correct equations from templates, a dataset of problem texts and their associated templates is required. Our point is that an intelligent machine can ask the right questions and, by answering them, return crucial information such as numbers (for some template slots) to be used in the equation. Math word problems include world knowledge, and it is implicit and in varying degrees in those problems. Consequently, the state-of-the-art question generation and answering systems may fall short in this aspect.

The proposed approach asks multiple specific questions given a math word problem. Pieces of evidence show that those questions could provide all the necessary information to solve a math problem if they focus on the correct aspects. We ask the right questions using a rule-based question generation system [10] and answer those custom questions by a state-of-the-art question answering system, thanks to its contextual understanding. The source code with dependencies is provided in GitHub<sup>1</sup>.

In the remainder of the paper, the word “problem” refers to a math problem inside the dataset. On the other hand, the word “question” refers to the generated questions used to acquire necessary information from the math problem. We generate these questions using the rule-based question generation system and treat them as the “right” questions for the problem at hand.

#### 3.1. Template-based solvers

Template-based methods [7, 8, 13, 14] split the process of math problem solving into two tasks: Template selection and number assignment. In literature, both these tasks are generalized under the name “alignment problem.”

##### 3.1.1. Template selection task

Each math problem in the dataset has a specific equation to place its numbers into for its solution. Choosing the correct equation type for a math problem is the primary purpose of this task. While the variables in these equations could be values like “x” and “y,” the numbers in an empty template are denoted as “n1, n2, n3, ..., etc.” so that they can be filled in during the number assignment task.

##### 3.1.2. Number assignment task

When a math problem’s template type is selected, this task must fill empty number slots such as “n1” with numbers from the corresponding problem text. The aim is to place all the relevant numbers in a math problem

<sup>1</sup><https://github.com/zalio/word-problem-solver-thesis>.

text into their correct empty slots in that problem’s template type. Firstly, the aligner should extract all crucial numbers in a math problem text. Then, it should put these numbers into the number slots in the correct order. For example, three slots named  $n_1$ ,  $n_2$ , and  $n_3$  must be filled with three specific numbers from a problem text in the correct order.

### 3.2. Dataset

One of the most commonly used datasets for the equation set problems that involve multiple unknown variables is the “alg514” dataset, which is released by Kushman et al. [7]. Zhou et al. [8] report the performance of their approach on this benchmark as well.

In this dataset, each math problem contains four pieces of information: “iIndex,” “IEquations,” “sQuestion,” and “lSolutions.” “iIndex” is simply a unique number to represent each math problem. “IEquations” shows a problem’s solution equation with all of its values placed correctly. “sQuestions” is the text of the math problem. Finally, “lSolutions” contains the actual answer to the math problem after the solution equation is solved.

The dataset consists of 514 problems, 28 templates, 1.62K sentences, and 19.3K words with a vocabulary size of 2352. We preprocess the “alg514” dataset to make it ready for the proposed approach.

#### 3.2.1. Data preprocessing

To solve math problems using the proposed approach, we apply some steps of data preprocessing.

Firstly, we preprocess the dataset so that all of its problems have the necessary information about their template types. As a preprocessing step, we add “template” and “templateType” information to each entry in the dataset. The variables in the solution equation have been replaced with “x” and “y.” Moreover, the numbers are replaced with different  $n$  values to represent the empty number slots. The resulting equation becomes the problem’s blank equation template. This blank template is named as a specific template type. Thus, every entry receives the “template” and “templateType” fields next to themselves in the preprocessed dataset.

After this preprocessing step, all 514 problems in the dataset are organized into 22 different template types and extended with template type and empty template information accordingly.

The main focus of our math problem solver is the number assignment task. Within the scope of our math problem solver, we assume each math problem’s template type is already known. Afterward, our math problem solver will assign the selected numbers to the already known empty equation template from the problem text. Because of this assumption, grouping by template types eases figuring out a problem’s pattern according to its template type.

Our workflow diagram (Figure 2) depicts the pipeline of the proposed approach once the dataset is preprocessed as required. In the following subsections, we explain each such step.

### 3.3. Question generation and answering

The main idea behind the proposed math problem solver is that one could pose natural language questions to ask for the numbers in the solution equation template. Humans can easily ask the right questions to get the required numbers as a result. To make a machine to ask the right questions given the problem text, we relied on a rule-based automatic question generation system [10] to have some control over the types of questions (constrain them to focus on numerical aspects) since rule-based systems generally work with template-based rules to develop questions. We generated multiple questions for all 514 problems in the dataset. Thanks to this

question generator, it is possible to create questions that ask the crucial numerical details of a math problem simply by providing its short text as a context paragraph input.

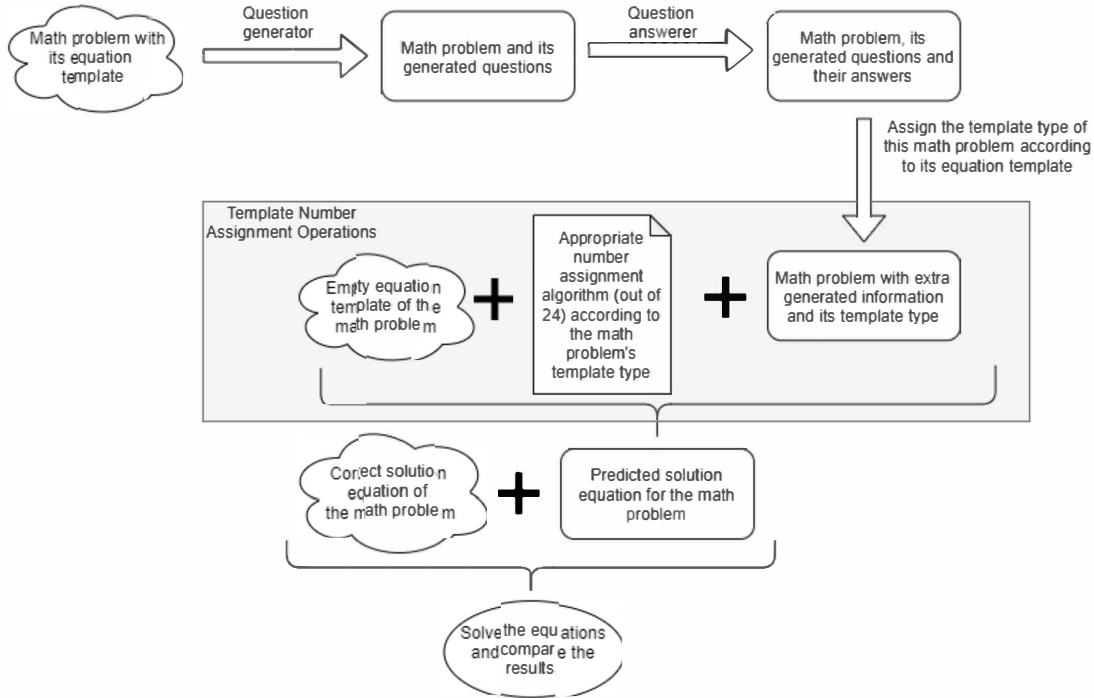


Figure 2. Workflow diagram.

The rationale behind the selection of this question generation approach is based on the additional information required for the number assignment steps. Our number assignment rule sets mainly focus on how often a number or a number group appears in the answers to reach a solution. For this reason, the answers to generated questions should contain many numbers or number groups (two or more numbers that always appear together in the generated questions and answers) from a math problem. Our question generation-answering pipeline fulfills the requirement to a large extent. As an ablation study, we experimented with another question generator called Questgen.ai<sup>2</sup> to test the question generation method’s effect on the overall performance. However, Questgen.ai did not meet our requirements as the generated questions do not contain numbers most of the time. Furthermore, the total number of generated questions for each math problem was around 4–5, which is insufficient for having reliable number assignment decisions for our rules.

Afterwards, we answer all the generated questions by a state-of-the-art question answering system. State-of-the-art question-answering systems rely on contextual language models and define the task as a text span prediction given an input paragraph. We conducted our experiments with the BERT-base-uncased pretrained model [9] which has 24 layers, 1024 hidden units, 16 attention heads, and 336M parameters. The fine-tuning details on the SQuAD v1.1 dataset for the pretrained model are as follows: 2 training epochs, Adam optimizer with a learning rate of  $3e-5$ , max sequence length as 384, doc stride as 128, and a batch size of 3. Furthermore, we implemented our code in Python. We performed all experiments on a workstation with Intel Core i7 11800H CPU, NVIDIA RTX 3070 GPU, 16 GB of RAM, Windows 10 OS.

<sup>2</sup>Questgen.ai (2021). Questgen.ai [online]. Website <https://questgen.ai> [accessed 28 March 2022]



Figure 3 shows an example math problem, its generated questions, and their answers using our pipeline. The upper part contains the original math problem. In the bottom part, on the left side of each arrow, resides a generated question. On the right side, its answer is given.

A writing workshop enrolls novelists and poets in a ratio of 5 to 3. There are 24 people at the workshop. How many novelists are there? How many poets are there?

How many people at the workshop are? -> 24  
 Does a writing workshop enroll novelists and poets? -> 5 to 3  
 How does a writing workshop enroll novelists and poets? -> 5 to 3  
 How many in a ratio of 5 to 3 does enroll in a ratio of 5 to novelists and poets? -> 24

**Figure 3.** An example math problem with the generated questions and their answers.

While all rule sets require the generated questions and answers for the number assignment step, most of them need the original math problem as well. The primary usage of the original problem text is to determine the order of the selected number pairs' assignments to the empty pair slots. Most problem types have slots that can be grouped into pairs, and the rule sets try to pick number pairs with the highest occurrences for them. This logic creates the need to assign two selected numbers into the empty pair slots in the correct order. Generally, the number that appears before the other one on the original problem gets the first empty pair slot. Then, the remaining number gets assigned to the remaining empty pair slot. Thus, in a sense, original math problem texts are prompts for the number assignment rule sets.

### 3.4. Custom rule sets for number assignment

Once the relevant numerical answers are taken from a question generation and answering pipeline, humans can perform the placement of numbers to the slots of the equation templates using their background knowledge. However, it is a nontrivial task for a machine. There can be different numbers of slots in different templates, different sets of questions are generated for problem texts, templates require a set of ordered numbers, etc. Thus, we wrote custom rule sets for each template in the collection to achieve the number assignment step automatically. Although we could not develop an all-encompassing rule set for the whole set of templates, the template-specific number assignment rules perform well within each template. We have 24 different template types and rule sets (i.e. number assignment algorithm) for alg514 and AI2 datasets.

Figure 4 depicts the pseudocode of the algorithm for the template type 10. Therein, the set  $S$  represents all the generated question-answer pairs of a math problem, where  $q$  is a generated question, and  $a$  is its answer. A set of conditions is applied in the first loop to filter the question-answer pairs in  $S$ . Then, we create a dictionary  $D$  to store the count of number pairs seen in the answer. In the second *while* loop, we update the frequency of such pairs using two conditional structures. We next select the most frequent tuple in the dictionary and place the tuple elements to the  $n1$  and  $n2$  slots using those elements' positions in the problem text. Finally, if all the slots are filled in, the equation set is established. Otherwise, the problem is labeled as unsolvable. One can find the trace of this algorithm on an example problem in Appendices<sup>3</sup>.

Using similar pseudocodes for other template types, we perform the number assignment task for all the 514 math problems in the dataset. Furthermore, the logic behind the rule sets in the form of modular subrules is given in Appendices. When these assignments are complete, solving the resulting equation set is trivial. We

<sup>3</sup><https://github.com/zalio/word-problem-solver-thesis/blob/main/appendices.pdf>

```

Data: a math problem P,  $S = \{(q_1, a_1), \dots, (q_k, a_k)\}$ , ["n1*x = n2*y", "x+y = n3"]
while for all (q, a)  $\in S$  do
    if q contains no number or number related words then
        | set this (q, a) pair to be ignored;
    end
    if q starts with the word "Where" then
        | set this (q, a) pair to be ignored;
    end
    if q contains more than 30 words then
        | set this (q, a) pair to be ignored;
    end
end
create a dictionary D to hold the appearance count of potential number pairs;
while for all (q, a)  $\in S$  such that its q is not set to be ignored do
    if q starts with the words "How many" And a is only a number by itself And "n3" slot
    is empty then
        | n3 slot  $\leftarrow$  the number in a;
    end
    if q and a have at least 2 number values in total then
        | find all potential number pairs for each number in the q and a group;
        | increase the count of each number pair appearance in the dictionary D;
    end
end
assign (m1, m2) as the number pair with the max appearance count;
if m1 appears before m2 in P then
    | n2 slot  $\leftarrow$  the number in m1;
    | n1 slot  $\leftarrow$  the number in m2;
else
    | n1 slot  $\leftarrow$  the number in m1;
    | n2 slot  $\leftarrow$  the number in m2;
end
if n1 Or n2 Or n3 is empty then
    | return a flag that shows this math problem as unsolvable;
else
    | return n1 And n2 And n3
end

```

**Figure 4.** Solution algorithm for the template type 10 problems.

used the Python library “sympy”<sup>4</sup> to get the final set of solutions.

#### 4. Experimental evaluation

As part of the evaluation, we provide the performance of the proposed approach on the alg514 dataset. We test the accuracy of the proposed approach also on the AI2 corpus, a collection of simpler arithmetic word problems. The corpus consists of 396 problems, 1483 sentences, and 13,632 words.

<sup>4</sup>SymPy Development Team (2021). SymPy Documentation [online]. Website <https://docs.sympy.org/latest/index.html> [accessed: 22.05.2021]

Table 1 reports the detailed accuracy results, where the overall accuracy is 85.2. Table 2 shows the accuracy rates for template groups based on their frequency bins. As the most frequent template type is 15, it fills up an individual bin. We see a balanced performance over all different template groups, which account for almost the same percentage of data.

We also compare the performance against template-based baselines and state-of-the-art deep neural solvers (Table 3). The results show that our solver outperforms them by a significant margin.

**Table 1.** Detailed accuracy of the proposed approach for the alg514 dataset.

The total number of problems	514
The total number of correct solutions out of all problems	438
The total number of incorrect solutions out of all problems	45
The total number of unsolvable problems out of all questions	31
Final accuracy score for all problems	85.2

**Table 2.** Performance on different template groups based on their frequencies on the alg514 dataset.

Template types	Avg. accuracy	Corresponding number of problems	% of data
$\leq 8$	83.85%	132	0.26
9 – 14	90.23%	112	0.22
15	81.72%	186	0.36
16 – 21	98.55%	84	0.16

**Table 3.** Comparison with the state-of-the-art approaches on the alg514 dataset.

Approach	Accuracy for alg514 dataset
Kushman et al. (2014) [7]	68.7
Zhou et al. (2015) [8]	79.7
Upadhyay et al. (2016) [14]	83.0
Wang et al. (2017) [2]	70.1
Huang et al. (2018) [29]	82.5
The Proposed Approach	<b>85.2</b>

It should be noted that the proposed approach only focuses on the number assignment task, while the compared approaches perform both template selection and number assignment tasks. In our case, the equation template must be acknowledged to solve a given math problem. The main intuition is by framing the right questions and answering them, you can get good clues regarding the organization of numbers to form an equation instance. All this can be accomplished through an automated question generation and answering pipeline. We add simple rule sets on top of this pipeline to dispatch the solving of equation systems. Although we could not induce a single rule set for all the template types, template-specific rule sets resemble each other, where some part of the dissimilarity can be accounted to the template differences such as the number of slot variables etc. To conclude, the proposed workflow highlights that a question generation and answering pipeline has a promising potential to solve word algebra problems.

Additionally, we report detailed accuracy results, average accuracies based on the template groups, and comparison with the state-of-the-art for the AI2 arithmetic word problems dataset in Tables 4–6 respectively. Here, though Sundaram et al.'s [28] is the best performing approach, ours' performance is close to it and the second best.

**Table 4.** Detailed accuracy of the proposed approach for the AI2 dataset.

The total number of problems	396
The total number of correct solutions out of all problems	347
The total number of incorrect solutions out of all problems	12
The total number of unsolvable problems out of all questions	37
Final accuracy score for all problems	87.6

**Table 5.** Performance on different template groups based on their frequencies on the AI2 dataset. (Template types that are not explicitly shown here do not exist in the AI2 dataset.)

Template types	Avg. accuracy	Corresponding number of problems	% of data
3	97.39%	115	0.290
4, 5, 7	71.43%	77	0.195
22	94.49%	127	0.320
23	77.92%	77	0.195

**Table 6.** Comparison with the state-of-the-art approaches on the AI2 dataset.

Approach	Accuracy for AI2 dataset
Kushman et al. (2014) [7]	64.0
Hosseini et al. (2014) [11]	77.7
Sundaram and Khemani (2015) [28]	<b>88.64</b>
Mitra and Baral (2016) [30]	86.07
Wang et al. (2017) [2]	52.96
MathDQN (Wang et al. 2018) [31]	78.5
Zaporojets et al. (2021) [3]	75.47±0.62
The proposed approach	87.6

We also perform experiments on the SVAMP dataset [6]. SVAMP contains 1000 problems, 26 templates, and 1.24 average number of operators. Table 7 includes the comparative results against leading state-of-the-art approaches. All of them are sequence-to-sequence neural models and trained with RoBERTa [27] embeddings. We take their scores from Patel et al. [6]. SVAMP contains two main types of problems: One-Op and Two-Op. One-Op problems contain only a single operand out of  $\{+, -, *, .\}$  while Two-Op problems contain two operands that can be different. The problems in the SVAMP dataset are classified according to their main operands: ADD, SUB, MUL, and DIV. For Two-Op problems, the operand category is determined based on the placement of parentheses on the solution equations.

We observe from the results in Table 7 that the proposed approach has significantly higher accuracy on the One-Op problems because our already existing solution templates fit these problems. The performance on the Two-Op problems suffers as they contain many combinations of two different operands, e.g.,  $(+, -)$ ,  $(*, -)$ ,  $(/, +)$ , etc. Thus, there is a need for extending template types to cover different operand combinations.

**Table 7.** Comparison with the state-of-the-art approaches on the SVAMP dataset.

-	Seq2Seq [15]	GTS [16]	Graph2Tree [25]	The proposed approach
Full set (1000 problems)	40.3	41.0	43.8	<b>45.5</b>
One-Op (762 problems)	42.6	44.6	51.9	<b>59.0</b>
Two-Op (238 problems)	<b>33.1</b>	29.7	17.8	2.5
ADD (193 problems)	<b>41.9</b>	36.3	36.8	34.2
SUB (532 problems)	35.1	36.9	41.3	<b>45.7</b>
MUL (108 problems)	<b>38.7</b>	<b>38.7</b>	35.8	34.3
DIV (167 problems)	56.3	61.1	65.3	<b>65.7</b>

Another observation is that the accuracy values belonging to SUB and DIV types are higher than those of ADD and MUL. One defining characteristic of the SVAMP dataset is that problems generally have multiple extra numbers not used in the solution equations. The abundance of these numbers in the ADD and MUL problem types mainly causes the difference in accuracy because they mislead the number assignment process.

Table 8 shows the accuracy of the proposed approach on the SVAMP dataset with only One-Op problems. The relatively higher accuracy values in this table prove the real potential of the proposed method as long as the solution equations of the problems fit our template types. These results reflect that our pipeline’s question answerer module can capture problems’ contexts, and thus our solver is robust against the types of variations in SVAMP.

**Table 8.** Detailed results of the proposed approach on the SVAMP dataset with only One-Op problems.

-	Accuracy of The Proposed Approach
Full Set (1000 problems)	45.5
One-Op (762 problems)	59.0
ADD (139 problems)	47.5
SUB (403 problems)	58.8
MUL (74 problems)	50.0
DIV (146 problems)	75.2

#### 4.1. Qualitative analysis

While the quantitative results prove the usefulness of the proposed approach, it has shortcomings as well. To give an idea about the reasons why the proposed method fails in solving the problems, we provide three sample unsolved problems in Figure 5.

The first example problem poses a difficulty due to some implicit information not stated in the problem text. There are many cases similar to this. In this problem, the text does not explicitly include the number

of legs an animal possesses for apparent reasons. However, this causes a failure because the model lacks the knowledge of the mentioned animals' number of legs. It is possible to overcome this issue by training the machine using language contexts enriched with world knowledge.

The second example problem is incorrectly solved because it refers to multiplication operations as percentages of numbers, e.g., "0.5 of first number" or "0.3333 of the second number". This unusual way of referencing multiplication operations causes the question generator to ask incorrect questions on this problem. Answers to those wrong questions bring about the grouping of irrelevant numbers. Thus, the number assignments are negatively affected by incorrectly paired numbers, and the result becomes erroneous. One can overcome this issue by recognizing "of a number" phrases so that the irrelevant information in the generated questions does not lead to any confusion while assigning numbers.

The third example problem is unsolvable because the problem text contains the phrase "82 out of 85", which confuses the assignment of the number slot  $n_4$ . The correct number to be substituted into that slot is 82. However, besides textual understanding, one requires a simple inference to reach this conclusion. Here the total number of questions (85) does not make a difference because unanswered questions do not affect the calculation. The machine should make this inference to form the correct equation instance.

As seen from these examples, there are some issues to be resolved for math problem solvers. Contextual understanding, basic inference capabilities, and world knowledge are crucial aspects to integrate into those systems.

**Example Problem 1:** There are 41 animals on a farm , each of which is either a pig or a chicken. There are 100 legs altogether. How many pigs are there? How many chickens?  
 Correct number slot values:  $n_1 = 4.0$  ;  $n_2 = 2.0$  ;  $n_3 = 100.0$  ;  $n_4 = 41.0$   
 Correct solution equation: " $4.0*x+2.0*y=100.0$ ", " $x+y=41.0$ "  
 Correct answer:  $x = 9, y = 32$

**Example Problem 2:** The sum of 2 numbers is 27. 0.5 of the first number plus 0.3333 of the second number is 11. Find the smaller and the larger number.  
 Correct number slot values:  $n_1 = 0.3333$  ;  $n_2 = 11.0$  ;  $n_3 = 0.5$  ;  $n_4 = 27.0$   
 Correct solution equation: " $0.5*x+0.3333*y=11.0$ ", " $x+y=27.0$ "  
 Correct answer:  $x = 12.0029994001200, y = 14.9970005998800$

**Example Problem 3:** On a college entrance exam , each correct answer adds 1 point to your raw score , each unanswered question adds nothing and each incorrect answer subtracts 0.25 point. You answer 82 out of 85 questions for a raw score of 67. How many questions did you answer correctly?  
 Correct number slot values:  $n_1 = 1.0$  ;  $n_2 = 0.25$  ;  $n_3 = 67.0$  ;  $n_4 = 82.0$   
 Correct solution equation: " $1.0*x-0.25*y=67.0$ ", " $x+y=82.0$ "  
 Correct answer:  $x = 70, y = 12$

**Figure 5.** Example unsolvable problems from the alg514 dataset.

## 5. Conclusion

In this paper, we provided an alternative viewpoint to word algebra problems. We took the inspiration from human solvers in that they frame questions out of the problem text and then solve the problem by answering them. Thus, in our solution, a question generation and answering pipeline leads to the formation of the equation instances by substituting appropriate numbers for the coefficient slots through custom rule sets.

Contextual understanding is crucial to solving word algebra problems by considering the relevant parts from the problem text. BERT-based question answering meets this requirement. Another aspect is world knowledge. In word problems datasets, we encounter scattered world knowledge, which is implicit most of

the time. As seen in the error analysis part, the lack of this background knowledge is a source of failure. There is a growing interest in injecting world knowledge into the existing contextual language models [32]. The developments in this research direction will directly contribute to the solution of word algebra problems.

As seen from the quantitative results, the proposed approach has superior performance on two benchmark datasets, one for word algebra problems and one for arithmetic word problems, compared to state-of-the-art systems. It also performs better against state-of-the-art deep learning-based solvers on the challenging SVAMP dataset. Its success can be attributed to asking the right questions and answering them through a state-of-the-art question answerer component. We must acknowledge that our solver performs only the number assignment step, excluding the template selection, where we assume that the correct equation template is given.

Our proposed solver begins with the generation of questions from the problem text. Once we get all the answers to the generated questions, we apply custom rule sets for template types to form the equation system. One downside is whenever a new template type arrives, a need for a new rule set arises. However, the transition was smooth when we extended our rule-based solver for the arithmetic word problems in a second dataset. Minor revisions to the previous template-specific algorithms were sufficient to solve the problems.

A future improvement can be in the generalization of the proposed approach. We plan to formulate the problem to induce a general rule set for different types of templates. This formulation will liberate the process from the already known template type requirement.

## References

- [1] Clark P, Etzioni O. My Computer Is an Honor Student - but How Intelligent Is It? Standardized Tests as a Measure of AI. *AI Mag.* 2016; 37 (1): 5-12. doi: 10.1609/aimag.v37i1.2636
- [2] Wang Y, Liu X, Shi S. Deep Neural Solver for Math Word Problems. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*; Copenhagen, Denmark; 2017: 845-854. doi: 10.18653/v1/D17-1088.
- [3] Zaporojets K, Bekoulis G, Deleu J, Demeester T, Develder C. Solving arithmetic word problems by scoring equations with recursive neural networks. *Expert Systems with Applications* 2021; 174 (1): 114704. doi: 10.1016/j.eswa.2021.114704
- [4] Wang L, Wang Y, Cai D, Zhang D, Liu X. Translating a Math Word Problem to a Expression Tree. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*; Brussels, Belgium; 2018: 1064-1069. doi: 10.18653/v1/D18-1132
- [5] Chiang T, Chen Y. Semantically-Aligned Equation Generation for Solving and Reasoning Math Word Problems. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*; Minneapolis, Minnesota; 2019: 2656-2668. doi: 10.18653/v1/N19-1272
- [6] Patel, A, Bhattamishra, S, Goyal, N. Are NLP Models really able to Solve Simple Math Word Problems? In: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*; Virtual Event; 2021: 2080-2094.
- [7] Kushman N, Artzi Y, Zettlemoyer L, Barzilay R. Learning to Automatically Solve Algebra Word Problems. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*; Baltimore, Maryland 2014: 271-281. doi: 10.3115/v1/P14-1026
- [8] Zhou L, Dai S, Chen L. Learn to Solve Algebra Word Problems Using Quadratic Programming. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*; Lisbon, Portugal 2015: 817-822. doi: 10.18653/v1/D15-1096

- [9] Devlin J, Chang M, Lee K, Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers); Minneapolis, Minnesota; 2019: 4171–4186. doi: 10.18653/v1/N19-1423
- [10] Keklik O, Tuğlular T, Tekir S. Rule-Based Automatic Question Generation Using Semantic Role Labeling. IEICE TRANSACTIONS on Information and Systems 2019; E102-D (7): 1362-1373. doi: 10.1587/transinf.2018EDP7199
- [11] Hosseini MJ, Hajishirzi H, Etzioni O, Kushman N. Learning to Solve Arithmetic Word Problems with Verb Categorization. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing; Doha, Qatar; 2014.pp. 523-533. doi: 10.3115/v1/d14-1058
- [12] Zhang D, Wang L, Zhang L, Dai BT, Shen HT. The Gap of Semantic Parsing: A Survey on Automatic Math Word Problem Solvers. IEEE Trans. Pattern Anal. Mach. Intell. 2020; 42 (9): 2287-2305. doi: 10.1109/TPAMI.2019.2914054
- [13] Huang D, Shi S, Lin C, Yin J. Learning Fine-Grained Expressions to Solve Math Word Problems. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017; Copenhagen, Denmark; 2017: 805–814.https://doi.org/10.18653/v1/d17-1084
- [14] Upadhyay S, Chang M, Chang K, Yih W. Learning from Explicit and Implicit Supervision Jointly For Algebra Word Problems. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing; Austin, Texas; 2016. pp. 297-306. doi: 10.18653/v1/D16-1029.
- [15] Luong T, Pham H, Manning C.D. Effective approaches to attention-based neural machine translation. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing; Lisbon, Portugal 2015: 1412–1421.
- [16] Lee D, Ki K, Kim B, Gweon G. TM-generation model: a template-based method for automatically solving mathematical word problems. The Journal of Supercomputing 2021. doi: 10.1007/s11227-021-03855-9
- [17] Mandal S, Naskar S. Classifying and Solving Arithmetic Math Word Problems—An Intelligent Math Solver. IEEE Transactions on Learning Technologies 2021. pp. 1-1. doi: 10.1109/TLT.2021.3057805
- [18] Koncel-Kedziorski R, Roy S, Amini A, Kushman N, Hajishirzi H. MAWPS: A Math Word Problem Repository. In: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies; San Diego, California; 2016. pp. 1152-1157. doi: 10.18653/v1/N16-1136
- [19] Lin X, Huang Z, Zhao H, Chen E, Liu Q, et al. HMS: A Hierarchical Solver with Dependency-Enhanced Understanding for Math Word Problem. Proceedings of the AAAI Conference on Artificial Intelligence 2021; 35 (1): 4232-4240.
- [20] Hong Y, Li Q, Ciao D, Huang S, Zhu S. Learning by Fixing: Solving Math Word Problems with Weak Supervision. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event; 2021. pp. 4959-4967.
- [21] Xie Z, Sun S. A Goal-Driven Tree-Structured Neural Model for Math Word Problems. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence; Macao, China; 2019. pp. 5299-5305. doi: 10.24963/ijcai.2019/736.
- [22] Wu Q, Zhang Q, Wei Z, Huang X. Math Word Problem Solving with Explicit Numerical Values. In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021; Virtual Event; 2021. pp. 5859-5869.
- [23] Yu, W, Wen, Y, Zheng, F, Xiao, N. Improving Math Word Problems with Pre-trained Knowledge and Hierarchical Reasoning. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing; Virtual Event; 2021. pp. 3384-3394.
- [24] Liang, Z, Zhang, J, Shao, J, Zhang, X. MWP-BERT: A Strong Baseline for Math Word Problems. 2021.



- [25] Zhang J, Wang L, Lee RK, Bin Y, Wang Y et al. Graph-to-tree learning for solving math word problems. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics; Online; 2020: 3928–3937.
- [26] Miao S, Liang C, Su K. A Diverse Corpus for Evaluating and Developing English Math Word Problem Solvers. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics; 2020:975-984. doi: 10.18653/v1/2020.acl-main.92
- [27] Liu Y, Ott M, Goyal N, Du J, Joshi M et al. Roberta: A robustly optimized bert pretraining approach. 2019. doi: 10.48550/ARXIV.1907.11692.
- [28] Sundaram SS, Khemani D. Natural Language Processing for Solving Simple Word Problems. In: Proceedings of the 12th International Conference on Natural Language Processing; Trivandrum, India; 2015. pp. 394-402.
- [29] Huang D, Liu J, Lin C, Yin J. Neural Math Word Problem Solver with Reinforcement Learning. In: Proceedings of the 27th International Conference on Computational Linguistics; Santa Fe, New Mexico, USA; 2018. pp. 213-223.
- [30] Mitra A, Baral C. Learning To Use Formulas To Solve Simple Arithmetic Problems. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) 2016; 1 (1): 2144-2153. doi: 10.18653/v1/P16-1202
- [31] Wang L, Zhang D, Gao L, Song J, Guo L et al. MathDQN: Solving Arithmetic Word Problems via Deep Reinforcement Learning. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18); New Orleans, Louisiana, USA; 2018. pp. 5545-5552.
- [32] Colon-Hernandez P, Havasi C, Alonso JB, Huggins M, Breazeal C. Combining pre-trained language models and structured knowledge. CoRR 2021; Volume abs/2101.12294.