DRAW: A Challenging and Diverse Algebra Word Problem Set

Shyam Upadhyay
University of Illinois at Urbana-Champaign
Urbana, IL, USA
upadhya3@illinois.edu

Ming-Wei Chang
Microsoft Research
Redmond, WA, USA
minchang@microsoft.com

Abstract

We present DRAW, a dataset consisting of 1000 algebra word problems, semi-automatically annotated for the evaluation of automatic solvers.[1] Details of the annotation process are described, which involves a novel template reconciliation procedure for reducing equivalent templates. DRAW also consists of richer annotations, including gold coefficient alignments and equation system templates, which were absent in existing benchmarks.

We present a quantitative comparison of DRAW to existing benchmarks, showing that DRAW consists a wide variety of problems, both in terms of narrative diversity and problem types. We provide a strong baseline for DRAW using a simple yet powerful solver. We also experimentally verify that the additional annotations indeed improves the performance for our automatic solver.

1 Introduction

Developing automatic solvers which can solve simple mathematical reasoning problems is a long pursued sub-goal of AI (Newell et al., 1959; Bobrow, 1964; Mukherjee and Garain, 2008). Recent work (Kushman et al., 2014; Hosseini et al., 2014; Shi et al., 2015) in the NLP community has been devoted to solving word problems, as they require synergy across several disciplines — the automatic systems should be able to parse natural language, perform reasoning and model world knowledge.

A *word problem* is a piece of text describing a real world scenario or a story, where mathematical relationships are expressed in the narrative instead of explicit algebraic equations. There are two steps for solving the algebra word problems: building mathematical expressions (*the semantic parsing component*) and answering the questions (*the question answering component*).

The following is an example of a word problem:

> Maria is now four times as old as Kate. Four years ago, Maria was six times as old as Kate. Find each of their actual ages now.

The input of the semantic parsing component is a word problem (like the one above), and the desired output is an *equation system* which expresses the same mathematical relationship symbolically: $m = 4 \times n$ and $m - 4 = 6 \times (n - 4)$. The question answering component then parses the question, like "*find their ages now*" or "*what is the age of the older sister*" and answers according to the equation system. Following previous work (Kushman et al., 2014; Zhou et al., 2015), we mainly focus on the semantic parsing component and leave the question answering component as future work.

While there exist a few benchmark datasets (Kushman et al., 2014; Hosseini et al., 2014; Shi et al., 2015) for this task, they suffer from several limitations. For example, although the data sets proposed in (Kushman et al., 2014; Hosseini et al., 2014) have good narrative variety, they have limited variations in the equation systems. On the other hand, the data set proposed in (Shi et

[1] The dataset and the official train-dev-test split can be downloaded at http://research.microsoft.com/en-US/downloads/5aeeacb2-8abd-4999-8ac4-9155d1f85109/default.aspx.

al., 2015) focuses on number word problems[2] and has many different types of equation systems, but it has limited textual variations and lacks narrative. We believe an ideal evaluation dataset for algebra word problems should have enough diversity on *both* vocabularies and equation systems, as it lends to broad-domain mathematical reasoning.

In this paper, we describe **D**ive**R**se **A**lgebra **W**ord problem set (DRAW), a new dataset with 1000 annotated algebra word problems over a wide range of narratives and equation systems, thereby closing the gap left by earlier datasets. DRAW is also the first dataset that provides the alignments between coefficients in equations and the numbers in the text and annotated templates. The alignment annotations uniquely identify the derivation of an equation system from its template, thereby providing more reliable features for learning.

To summarize, the contributions of our paper are:

- We release DRAW, a challenging algebra word problem dataset that has a rich diversity in vocabulary and the types of equation systems.
- Unlike existing datasets, DRAW includes gold coefficient alignments that are necessary uniquely identify the derivation of an equation system (Section 3). We also propose a novel template reconciliation procedure and use it in DRAW's annotation process.
- We constructed a simple yet powerful baseline system that achieve close to state-of-the-art results on existing datasets using only POS tagging and number extraction components. We then analyze its performance by re-training on DRAW, to show the challenges of DRAW and confirms the benefits of the additional annotations (Section 5).

2 Preliminaries

We describe the notations and definitions used throughout the paper. Table 1 concisely explains our notations using an example word problem.

Let x represent an algebra word problem. Each example in the annotated dataset provides us with a word problem x and an *equation system* expressing the algebraic relation described in x. An equation system consists of one or more *equations*.

We define a *textual number* as span of text in x that indicates a numeric quantity. Some examples of textual numbers are "two" in "two numbers","14" in "14 kittens", "one" in "one day" etc. Note that some textual numbers need not be relevant to the problem. We use the quantities normalizer in Stanford CoreNLP (Manning et al., 2014) and a small lexicon that maps phrases to numbers (e.g. "half" to 0.5), to extract all the textual numbers. The set all textual numbers for a problem x is denoted by $\mathcal{Q}(x)$.

We denote an equation system using E. Note that E only contains variables and constants. The real-value solution(s) of the equation system can be obtained by directly applying a solver[3] to E.

Kushman et al. (2014) introduced the notion of an *equation system template*[4]. We define an equation system template as follows:

Definition 1 *An* equation system template T *is a family of equation systems parameterized by a set of coefficients* $\mathcal{C}(T) = \{c_i\}_{i=1}^{k}$*, where each coefficient* c_i *aligns to a textual number in a word problem.*

To identify an equation system belonging to the family represents by T, we need *alignments* to specify the value of each coefficient. More formally, an *alignment* A is a set of tuples $A = \{(q, c)\}$ s.t. $q \in \mathcal{Q}(x)$, and $c \in \mathcal{C}(T) \cup \{\epsilon\}$.[5]

We use $(a, b, c, \ldots)$ to represents coefficients and $(m, n, \ldots)$ to represent the variables.

A Universal Template? At first glance, it may seem that only a single equation system template should suffice for solving all algebra problems.

More specifically, consider the equation system template:

$$Am + Bn = C,$$
$$Dm + En = F.$$

Note that here $A, B, \ldots, F$ are *not* the coefficients. In our definition, the coefficients need to be able

[2]Number word problems describes algebraic manipulations explicitly and are not set in a narrative. Thus such problems can often be expressed with simple vocabularies

[3]We use a gaussian elimination based solver.

[4]We use the terms "template" and "equation system template" interchangeably.

[5]The symbol ϵ is a special null-slot. An alignment (q, ϵ) means the number q is not relevant for the final equation system.

Terms	Sym	Example
Problems	x	Maria is now four times as old as Kate. Four years ago, Maria was six times as old as Kate. Find each of their actual ages now.
Textual Number	$\mathcal{Q}(x)$	{four, Four, six}
Equation System	E	$m = 4 \times n, m-4 = 6 \times (n-4)$
Solution		$n = 10, m = 40$
Equation System Template	T	$m - a \times n = -1 \times a \times b + b, m - c \times n = 0$
Variables	v	m, n
Coefficients	$\mathcal{C}(T)$	a, b, c
Alignment	A	$a \rightarrow 6$, $b \rightarrow$ Four, $c \rightarrow$ four

Table 1: Notations used in this paper. Note the difference between E and T. T is more informative than E, as it is impossible to distinguish between "four" and "Four" just using E in the example above. *In contrast to existing datasets, which only provide E,* DRAW *also provides annotation for T and A.*

to be mapped directly to the numbers in the text. Therefore, $A, B, \ldots, F$ etc. can denote any arithmetic expressions consisting of coefficients, which themselves map to some textual number. For example, A could be a complicated arithmetic expression such as $(1/c_2) + c_1$. Moreover, A and B can share coefficients in their respective arithmetic expressions. Every combination of arithmetic expressions describes a new template.

3 Data Collections and Annotations

We first discuss the annotations schema used by DRAW. We restrict ourselves to linear equation systems.

Existing datasets (Kushman et al., 2014; Shi et al., 2015; Hosseini et al., 2014) only include the annotations of the equation systems and the solutions. Although (Kushman et al., 2014) introduce the concept of equation system template and use a template-based approach, their dataset does not have annotations for templates and alignments. Unfortunately, an equation system alone is not enough to correctly identify the correct alignments of textual numbers to coefficients. Indeed, the example in Table 1, shows that we cannot disambiguate which numbers (four or Four) aligns to which real-value coefficients in the equation system (given that the multiple values that are 4) using only the equation system. Therefore, DRAW's annotation schema includes template and alignment annotations.

```
x + y = 10 (total cars)
2x+3y = 1
2(10-y)+3y = 1
20-2y+3y = 1
```

Figure 1: Equations crawled from the explanation given by a user for a single problem. The equations were extracted automatically.

We now discuss the procedure of creating DRAW in detail.

3.1 Crawling Word Problems

We crawl over 100k problems from `http://algebra.com`. The 100k word problems include some problems which require solving non-linear equations (e.g. finding roots of quadratic equations). As our focus in this work is to solve algebra word problems involving a system of linear equations, we filter out these problems using keyword matching. We also filter problems whose explanation do not contain a variable named "x". This leaves us with 12k word problems.

Extracting Equations A word problem on `algebra.com` is accompanied by a detailed explanation provided by instructors, which often includes a derivation of the relevant equations. Ideally, one would extract the equations from the accompanying explanation, thereby crawling both the word problems and their equations.

In our crawler, we use simple pattern matching rules to extract all the equations in the explanation. In Figure 1, we shown an example of the equations we extracted from the explanation of one problem.

The example demonstrates the difficulty of using crawled equations directly. Some of the equations are just derivations of other equations. Moreover, the equations often have explanatory annotations (`(total cars)` above), which are sometimes difficult to distinguish from variable (e.g. `10*(total_cars)`). Therefore, we did not use the extracted equations directly. Instead, we use these extracted equations to aid manual annotations, which we describe in detail in the next section.

3.2 Annotating Equation Systems

To aid manual annotation, we created an annotation tool, that presents a human annotator with a semi-clean version of a word problem and a list all the equations extracted from the explanation. The annotator can choose the correct equations from the list of extracted equations, and clean them if necessary. When the target equations are not in the list, the annotator can type the equation systems manually. We found that the list of extracted equations helps reduce the annotation effort considerably.

The problems often have sentences which are irrelevant to solving the word problem (e.g. "Please help me, I am stuck."). During cleaning, the annotator removes such sentences from the final word problem and performs some minor editing if necessary.[6]

3.3 Template and Alignment Derivation

After collecting word problems and their corresponding equation system, we now describe the procedure of generating the template and alignment annotations.

We generate the template and alignment annotation simultaneously. We first use an approximate procedure to generate a template and alignments from a word problem x and its equation system E. This is followed by a template reconciliation step, which we describe later. Finally, a post processing and cleaning is done to correct the alignments and templates.

In our automatic procedure for generating templates, for each real-value number appearing in E, we replace it by a coefficient c if we can find a textual number $q \in \mathcal{Q}(x)$ with the same value. At the same time, the alignment (q, c) is added to the final alignment set. When there are multiple valid choices of q, we randomly pick one. It is also possible that there are multiple real-value numbers in E with the same value. In such cases, we assume that they all map to the same coefficient. Note that this is an approximation procedure, and the generated template and alignment might not be correct.

We canonicalize the generated templates so that the terms with variables always appear on the left-hand side of the equation. Using this procedure, we generate 332 templates.

Template Reconciliation The above approximation over-generates templates as it is possible for two generated templates to be equivalent to each other. Consider the following two templates,

$$m - b \times n = a, m + n = c \quad (1)$$

$$m + n = a, m - c \times n = b. \quad (2)$$

Note that the template (1) and the template (2) are the same equation system template after renaming coefficients. To remove the extraneous templates, we perform a template reconciliation step. We reduce the number of template by seeking pairs of templates that are equivalent to each other, and only keep one of them. To identify such pairs, we first formalize the notion of template equivalence.

Given two template T_1 and T_2 with the same number of coefficients, let γ to represent a one-to-one mapping function that maps the coefficients of T_1 to the coefficients in T_2. That is,

$$\gamma(c) = c', c \in \mathcal{C}(T_1), c' \in \mathcal{C}(T_2)$$

such that $c_i \neq c_j \implies \gamma(c_i) \neq \gamma(c_j)$. In other words, γ represents a choice of renaming the coefficients. Then, we have the following definition:

Definition 2 *Let $\mathcal{C}_\gamma = \{c' | \gamma(c) = c', c \in \mathcal{C}(T_1)\}$. Two template T_1 and T_2 are said to be equivalent if and only if*

1. $|\mathcal{C}(T_1)| = |\mathcal{C}(T_2)|$
2. *There exist a mapping γ such that the equations in the template T_1 can be derived from the equation template T_2 using algebraic manipulations.*

In practice, we use an approximate method to detect if two templates are equivalent. We generate all possible mappings γ between the coefficient sets of two templates, T_1 and T_2. Given a γ, the template T_1 parameterized by $\mathcal{C}_\gamma$ and the equation template T_2 parameterized by $\mathcal{C}(T_2)$ share exactly the same set of the coefficients. We then generate a random assignment vector for these coefficients, where the value of each coefficient ranges from -1 to 1. This

[6] In some cases, some of the numbers in the text are rephrased ("10ml" to "10 ml") in order to allow NLP pipeline work properly.

assignment describes two equation systems. If T_1 and T_2 are equivalent, then the solutions of these equation systems should agree. For a given γ, we repeat this procedure 10 times and if the solutions agree for all 10 random assignments, we consider these two templates to be equivalent. We checked all possible mappings γ and see if there exists one mapping that make two templates equivalent.

We can find some interesting reductions using the above procedure. For example, the following reduction was found by our system:

$$\frac{a}{b}m + \frac{a}{c}m - a = \frac{a^2}{b} \rightarrow \frac{1}{b}m + \frac{1}{c}m - 1 = \frac{a}{b}$$

After applying our procedure, we reduced the total number of templates to 244 from 332 (a reduction of over 25% in size).

Post-Processing and Cleaning After the reduction is done, we manually go over the template and fix the template and alignment mistakes. After fixing the template, the annotators might introduce an unseen template that is equivalent to existing ones. Therefore, we re-run the template reconciliation procedure one more time to finalize the dataset. Finally, the total number of template is 238.

4 Comparisons to Existing Datasets

In this section, we compare DRAW to existing word problem datasets and highlight its unique properties.

Existing Datasets We compare DRAW to ALG-514 (Kushman et al., 2014) and VERB-395 (Hosseini et al., 2014). We briefly describe ALG-514 and VERB-395 first, and discuss the relatively new dataset DOLPHIN (Shi et al., 2015) later.

Alg-514 consists of 514 problems, ranging over a variety of narrative scenarios, including distance-speed, computing cost, object counting, simple interest etc.[7] Evaluation on ALG-514 is performed using manually chosen splits for cross-validation, where the splits are chosen so as to avoid unseen templates.

[7]We also augment this dataset with our annotation by running our template reconciliation procedure and adding the gold alignments. The new annotations are released in the same URL as well.

Dataset	DRAW	ALG-514	VERB-395
# of templates	232	24	3
# of problems	1000	514	395
Vocab[8]	2.48k	1.97k	1.10k
$\frac{\text{\# of problems}}{\text{\# of templates}}$	4.31	21.41	131.67
# of words	35.3k	19.3k	12.4k
# of sentences	2.67k	1.61k	1.13k

Table 2: Comparison of statistics of DRAW, ALG-514 (Kushman et al., 2014),VERB-395 (Hosseini et al., 2014). See text for details and the comparisons to DOLPHIN (Shi et al., 2015).

We also compare to **Verb-395**, which is a dataset aimed at simpler mathematical relationships expressed through arithmetic word problems. Arithmetic word problems can be viewed as a special case of algebra word problems, involving only one unknown variable. Evaluation on VERB-395 is also done using manually created splits, where the splits are chosen to test robustness to different dataset types.

Some statistics for all the datasets are shown in Table 2. We now analyze these along the following dimensions:

Template Diversity It is desirable to have a diverse set of equation system templates in a dataset. Otherwise, with a small template vocabulary, an automatic solver can simply memorize template specific rules for each template. We use size of the template vocabulary as a measure of diversity in the equation systems templates (i.e. equation-side variation) in the dataset.

In Table 2, it is clear that DRAW has the most number of templates. Surprisingly, the number of templates in DRAW is around 10 times that of the number of templates in ALG-514, whereas the number of problems is only around twice as large. The reason behind this is that in DRAW, we do not remove the problems with rare templates, as we believe that it is closer to the real-word settings.[9]

We also estimate the template diversity by the average number of problems which share the same template. This metric for DRAW is only 4.31, while

[8]Vocabulary was computed after removing all numbers in the text, as we are only looking for textual variation

[9]Kushman et al. (2014) removed problems whose template appeared less than 6 times in their dataset.

for ALG-514 and VERB-395, this metric is 21.41 and 131.67 respectively.

Narrative Diversity Another desirable property of an evaluation dataset is that it should be textually diverse. In Table 2, we can see that DRAW uses the largest non-numerical vocabulary. Given that DRAW consists of problems with a wide array of narrative scenarios, an automatic solver needs to learn how to ground various natural language expressions that lead to the same mathematical expression.

Comparison to Dolphin DOLPHIN (Shi et al., 2015) is a new dataset that has a different goal compared to DRAW. Unlike ALG-514, DOLPHIN focuses on *number word problems*. An example of a number word problem is:

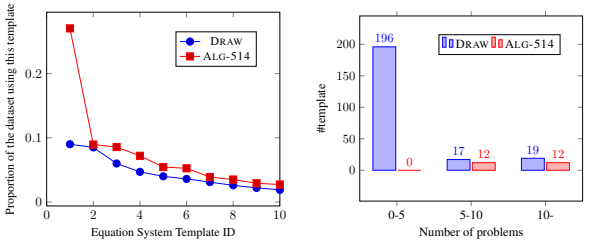
> The difference between two numbers is 8. Five times the larger number is 4 more than 8 time the other. Find the numbers ?

As we can see, number word problems express relationships between numbers, but solving them requires very little background knowledge. The nature of the number word problems restrict the size of the vocabulary, which explains why DOLPHIN has 1233 linear problems but only 0.69k words in its vocabulary. Therefore, although Dolphin also has impressive amount of variety in the equation systems, its vocabulary is the smallest of all of the datasets. This restricts its ability to evaluate the generalization of an automatic solver to broad domain mathematical reasoning problems.

4.1 Comparison with ALG-514

Among all the existing datasets, DRAW is most similar to ALG-514 as we both focus general algebra word problems. Moreover, several works have used ALG-514 as the benchmark dataset, so it is worthwhile to analyze the differences between ALG-514 and DRAW in detail.

We draw the template distribution for both ALG-514 and DRAW in Figure 2. In the Figure 2(a), we plot the proportion of problems that can be solved by a given template, for the top 10 templates. ALG-514 suffers from a skewed distribution of templates favoring one template, which appears in more than 25% of the problems. For DRAW, the top template only accounts for less than 10% of the problems.



(a) (b)

Figure 2: Comparison of the template distributions of DRAW and ALG-514. **(a)** We list the top 10 popular templates and show the proportion of the entire dataset which can be solved using these templates. **(b)** We list the number of templates appeared in less than 5 problems, 5-10 problems, and more than 10 problems, respectively.

In Figure 2(b), we bin the templates by the number of problems they appear in. It is clear that ALG-514 lacks rare templates (frequency < 5). On the other hand, a large proportion of templates in DRAW were seen in less than 5 problems. As rare templates appear naturally in a broad-domain problem solving setting, it is desirable to have a benchmark dataset which contains a long tail of rare templates.

We believe DRAW evaluates ability to generalize to rare templates and model textual variation in a more realistic setting than existing datasets.

5 Experiments

We first describe an automatic solver which achieves near state-of-the-art performance on existing benchmarks. Then we re-train the same solver on DRAW to provide a strong baseline. Our experimental results show that there are still a lot of room to improve performance on DRAW.

Moreover, we empirically show that template reconciliation and the additional alignment annotations, which are unique to the annotation schema of DRAW, prove valuable in learning better solvers.

Baseline Solver We describe a simple automatic solving system that we use in our experiments.

We train our model using the structured perceptron algorithm (Collins, 2002). In our inference algorithm, we first use our model to select the top 10 templates. Note that the templates are chosen from the set of training templates only, and hence this model does not have access to the templates in the

test data at all. After this step, all of the derivation for the chosen templates are generated and ranked by the model.

Following (Zhou et al., 2015), we do not model the alignment of nouns phrases to variables and hence our search space is small enough for the inference to be tractable.

Evaluation We use **solution accuracy** as the evaluation metric in all our experiments. Given a reference solution $G = (u_1, u_2, ...)$, and a system output $P = (v_1, v_2, ...)$, we execute the following:

1. If $|G| \neq |P|$, the solution is incorrect. Otherwise,
2. Sort P and G. If the maximum difference between corresponding positions in G and P is smaller than 0.001, the solution is correct.

Note that our evaluation is stricter than the standard evaluation adopted by (Kushman et al., 2014; Zhou et al., 2015).[10] To ensure fair comparisons, we use the relaxed version when conducting experiments on ALG-514.

Results on ALG-514 and VERB-395 We first discuss results on ALG-514 and VERB-395. We evaluate our system under the standard setting of 5-fold cross validation and 3-fold cross-validation respectively, in order to ensure fair comparison with other systems. Recall that the folds for ALG-514 and VERB-395 were manually created, and we use the folds provided by the authors. The results are reported in Table 3.

Dataset	Ours	KAZB[11]	ZDC[12]	ARIS [13]
ALG-514	78.2	68.7	79.7	N/A
VERB-395	78.3	64.0	N/A	77.7

Table 3: Comparison of our baseline system and other state-of-the-art systems. Results are averaged accuracy using 5-fold cross-validation on the ALG-514 dataset and 3-fold cross-validation for the VERB-395 dataset.

[10]The standard evaluation for ALG-514 only checks if there exists an element in P that is close enough for each element in G. In other words, the requirement $|G| = |P|$ is relaxed. It is important to notice that solution accuracy only evaluates the performance of the semantic parsing components rather than the question answer component given that the ordering of the answers is disregarded.

Setting	Dev	Test
DRAW	53.5	55.0
DRAW + ALG-514	53.0	53.5

Table 4: Results on DRAW. Note that adding ALG-514 does not improve performance.

Although our system does not use features from a dependency parser and a coreference system, the performance of our system is competitive to the state-of-the-art system of (Zhou et al., 2015) on ALG-514. On VERB-395, our baseline solver is also competitive to previous state-of-the-art system of (Hosseini et al., 2014), despite using very few resources.

Results on DRAW We randomly split[14] DRAW into train (600), dev (200) and test (200) examples, and re-train the baseline solver on the train split. Table 4 shows the results of the re-trained baseline solver on DRAW.

The baseline solver, which was achieving near state-of-the-art accuracy on ALG-514, achieves only 55 % on our dev and test split, *which is about 25% lower than the performance on* ALG-514 *despite the fact that the size of training data is larger.* It is evident that there is a lot of room for improvement on DRAW.

The result may appear surprising, given that both ALG-514 and DRAW are crawled from `Algebra.com`. But, as we analyzed in Section 4, DRAW has a wide variety of problems and textual variation, making it a harder evaluation setting than ALG-514.

A natural question to ask is what happens if we include ALG-514 in our training split. Interestingly, adding examples ALG-514 from does not improve the solution accuracy (Table 4). A possible explanation is that the template distribution of ALG-514 is quite different from that of DRAW.

Value of Annotations and Reconciliation To show the value of the additional alignment annotations and the template reconciliation, we first run experiments with the approximate procedure of finding alignments and templates automatically from the

[11]The system of (Kushman et al., 2014)

[12]The system of (Zhou et al., 2015)

[13]The system of (Hosseini et al., 2014)

[14]We release these splits with the dataset.

DRAW	Dev	Test
(1): auto template alignment deduction	47.5	54.0
(2): (1) + template reconciliation	51.0	54.5
(3): (2) + gold alignment	53.5	55.0

Table 5: Contributions of our additional annotations.

word problem and the equation system. In the second experiment, we add the template reconciliation module, which allows the solver to operate with a smaller template space. Finally, we run the system with both reduced template space and gold alignments. From the results in Table 5 we can see that both template reconciliation and alignment annotations improve accuracy consistently, justifying the need for these annotations.

6 Conclusion

We have described DRAW, a new dataset for algebra word problems which provide richer annotations and wider variety than existing benchmarks. We hope that DRAW enables a more realistic and extensive evaluation of automatic solvers for word algebra problems.

References

Daniel G. Bobrow. 1964. A question-answering system for high school algebra word problems. In *Proceedings of the October 27-29, 1964, Fall Joint Computer Conference, Part I*, AFIPS '64 (Fall, part I), pages 591–614, New York, NY, USA. ACM.

M. Collins. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms.

Javad Mohammad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. 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. Association for Computational Linguistics.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. 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. Association for Computational Linguistics.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60.

Anirban Mukherjee and Utpal Garain. 2008. A review of methods for automatic understanding of natural language mathematical problems. *Artif. Intell. Rev.*, 29(2):93–122.

Allen Newell, John C Shaw, and Herbert A Simon. 1959. Report on a general problem-solving program. In *IFIP Congress*, pages 256–264.

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1132–1142, Lisbon, Portugal, September. Association for Computational Linguistics.

Lipu Zhou, Shuaixiang Dai, and Liwei Chen. 2015. Learn to solve algebra word problems using quadratic programming. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 817–822, Lisbon, Portugal, September. Association for Computational Linguistics.