Annotating Derivations: A New Evaluation Strategy and Dataset for Algebra Word Problems

Shyam Upadhyay[1] **and Ming-Wei Chang**[2]

[1]University of Illinois at Urbana-Champaign, IL, USA
[2]Microsoft Research, Redmond, WA, USA
upadhya3@illinois.edu
minchang@microsoft.com

Abstract

We propose a new evaluation for automatic solvers for algebra word problems, which can identify mistakes that existing evaluations overlook. Our proposal is to evaluate such solvers using *derivations*, which reflect how an equation system was constructed from the word problem. To accomplish this, we develop an algorithm for checking the equivalence between two derivations, and show how derivation annotations can be semi-automatically added to existing datasets. To make our experiments more comprehensive, we include the derivation annotation for DRAW-1K, a new dataset containing 1000 general algebra word problems. In our experiments, we found that the annotated derivations enable a more accurate evaluation of automatic solvers than previously used metrics. We release derivation annotations for over 2300 algebra word problems for future evaluations.

1 Introduction

Automatically solving math reasoning problems is a long-pursued goal of AI (Newell et al., 1959; Bobrow, 1964). Recent work (Kushman et al., 2014; Shi et al., 2015; Koncel-Kedziorski et al., 2015) has focused on developing solvers for *algebra word problems*, such as the one shown in Figure 1. Developing a solver for word problems can open several new avenues, especially for online education and intelligent tutoring systems (Kang et al., 2016). In addition, as solving word problems requires the ability to understand and analyze natural language, it serves as a good test-bed for evaluating progress towards goals of artificial intelligence (Clark and Etzioni, 2016).

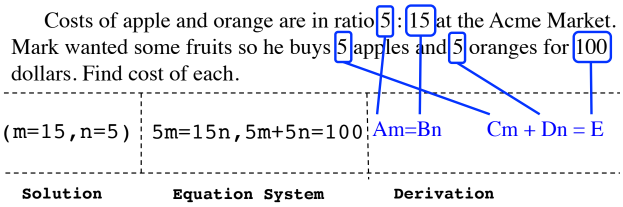
Costs of apple and orange are in ratio 5 : 15 at the Acme Market. Mark wanted some fruits so he buys 5 apples and 5 oranges for 100 dollars. Find cost of each.
(m=15,n=5)
5m=15n,5m+5n=100
Am=Bn
Cm + Dn = E
Solution
Equation System
Derivation

Figure 1: An algebra word problem with its *solution*, *equation system* and *derivation*. Evaluating solvers on derivation is more reliable than evaluating on solution or equation system, as it reveals errors that other metric overlook.

An automatic solver finds the solution of a given word problem by constructing a *derivation*, consisting of an un-grounded equation system[1] ($\{\mathtt{Am} = \mathtt{Bn}, \mathtt{Cm} + \mathtt{Dn} = \mathtt{E}\}$ in Figure 1) and *alignments* of numbers in the text to its coefficients (blue edges). The derivation identifies a grounded equation system $\{\mathtt{5m} = \mathtt{15n}, \mathtt{5m} + \mathtt{5n} = \mathtt{100}\}$, whose *solution* can then be generated to answer the problem. A derivation precisely describes *how* the grounded equation system was constructed from the word problem by the automatic solver. On the other hand, the grounded equation systems and the solutions are less informative, as they do not explain which span of text aligns to the coefficients in the equations.

While the derivation is clearly the most informative structure, surprisingly, no prior work evaluates automatic solvers using derivations directly. To the best of our knowledge, none of the current datasets contain human-annotated derivations, possibly due to the belief that the current evaluation metrics are sufficient and the benefit of evaluating on derivations is minor. Currently, the most popular evaluation strategy is to use *solution accuracy* (Kushman et al., 2014; Hosseini et al., 2014; Shi et al., 2015; Koncel-Kedziorski et

[1] Also referred to as a *template*. We use these two terms interchangeably.

Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers, pages 494–504,
Valencia, Spain, April 3-7, 2017. ©2017 Association for Computational Linguistics

al., 2015; Zhou et al., 2015; Huang et al., 2016), which computes whether the solution was correct or not, as this is an easy-to-implement metric. Another evaluation strategy was proposed in (Kushman et al., 2014), which finds an approximate derivation from the gold equation system and uses it to compare against a predicted derivation. We follow (Kushman et al., 2014) and call this evaluation strategy the *equation accuracy.* [2]

In this work, we argue that evaluating solvers against human labeled derivation is important. Existing evaluation metrics, like solution accuracy are often quite generous — for example, an incorrect equation system, such as,

$$\{m + 5 = n + 15, \quad m + n = 15 + 5\}, \qquad (1)$$

can generate the correct solution of the word problem in Figure 1. While equation accuracy appears to be a stricter metric than solution accuracy, our experiments show that the approximation can mislead evaluation, by assigning higher scores to an inferior solver. Indeed, a correct equation system, ($5m = 15n, 5m+5n = 100$), can be generated by using a wrong template, $\mathtt{A}m = \mathtt{B}n, \mathtt{A}m + \mathtt{A}n = \mathtt{C}$, and aligning numbers in the text to coefficients incorrectly. We show that without knowing the correct derivation at evaluation time, a solver can be awarded for the wrong reasons.

The lack of annotated derivations for word problems and no clear definition for comparing derivations present technical difficulties in using derivation for evaluation. In this paper, we address these difficulties and for the first time propose to evaluate the solvers using *derivation accuracy*. To summarize, the contributions of this paper are:

- We point out that evaluating using derivations is more precise compared to existing metrics. Moreover, contrary to popular belief, there is a meaningful gap between the derivation accuracy and existing metrics, as it can discover crucial errors not captured previously.

- We formally define when two derivations are equivalent, and develop an algorithm that can determine the same. The algorithm is simple to implement, and can accurately detect the equivalence even if two derivations have very different syntactic forms.

- We annotated over 2300 word algebra problems[3] with detailed derivation annotations, providing high quality labeled semantic parses for evaluating word problems.

Word Problem	$\boldsymbol{x}$	We are mixing a solution of 32% sodium and another solution of 12% sodium. How many liters of 32% and 12% solution will produce 50 liters of a 20% sodium solution?
Textual Numbers	$\mathcal{Q}(\boldsymbol{x})$	$\{32_1, 12_1, 32_2, 12_2, 50, 20\}$
Equation System	$\boldsymbol{y}$	$32m + 12n = 20 * 50,$ $m + n = 50$
Solution		$m = 20, n = 30$
Template	T	$\mathtt{A}m + \mathtt{B}n = \mathtt{C} * \mathtt{D},$ $\mathtt{m} + \mathtt{n} = \mathtt{C}$
Coefficients	$\mathcal{C}(T)$	$\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{D}$
Alignments	A	$\{32_1 \rightarrow \mathtt{A},\ 12_1 \rightarrow \mathtt{B},$ $50 \rightarrow \mathtt{C}, 20 \rightarrow \mathtt{D}\}$
EquivTNum		$\{[32_1, 32_2], [12_1, 12_2]\}$
Derivation	$\boldsymbol{z}$	(T, A)

Table 1: The symbols we used in the paper. Our proposed annotations are shown in **bold**. Equivalent textual numbers, described in EquivTNum, are distinguished with subscripts.

2 Evaluating Derivations

We describe our notation and revisit the notion of derivation introduced in (Kushman et al., 2014). We then formalize the notion of derivation equivalence and provide an algorithm to determine it.

Structure of Derivation The word problem in Table 1 shows our notation, where our proposed annotations are shown in **bold**. We denote a word problem by $\boldsymbol{x}$ and an *equation system* by $\boldsymbol{y}$.

An un-grounded equation system (or *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* (e.g., *four*) in the word problem. We also refer to the coefficients as *slots* of the template. We use $(\mathtt{A}, \mathtt{B}, \mathtt{C}, \ldots)$ to represents coefficients and $(\mathtt{m}, \mathtt{n}, \ldots)$ to represent the unknown variables in the templates.

Let $\mathcal{Q}(\boldsymbol{x})$ be the set of all the textual numbers in the problem $\boldsymbol{x}$, and $\mathcal{C}(T)$ be the coefficients to be determined in the template T. An *alignment* is a set of tuples $A = \{(q, c) \mid q \in \mathcal{Q}(\boldsymbol{x}), c \in \mathcal{C}(T) \cup \{\epsilon\}\}$ aligning textual numbers to coefficient slots,

[2] Note that an approximation of the derivation is necessary, as there is no annotated derivation. From the brief description in their paper and the code released by Kushman et al. (2014), we found that their implementation assumes that the first derivation that matches the equations and generates the correct solution is the correct reference derivation against which predicted derivations are then evaluated.

[3] available at `https://aka.ms/datadraw`

where a tuple (q, ϵ) indicates that the number q is not relevant to the final equation system.

Note that there may be multiple semantically equivalent textual numbers. e.g., in Figure 1, either of the 32 can be aligned to coefficient slot $\mathtt{A}$ in the template. These equivalent textual numbers are marked in the EquivTNum field in the annotation. If two textual numbers $q, q' \in$ EquivTNum, then we can align a coefficient slot to either q or q', and generate a equivalent alignment.

An alignment A and a template T together identify a *derivation* $\boldsymbol{z} = (T, A)$ of an equation system. Note that there may be multiple valid derivations, using one of the equivalent alignments. We assume there exists a routine $\mathtt{Solve}(\boldsymbol{y})$ that find the solution of an equation system. We use a Gaussian elimination solver for our $\mathtt{Solve}$ routine. We use hand-written rules and the quantity normalizer in Stanford CoreNLP (Manning et al., 2014) to identify textual numbers.

Derivation Equivalence We define two derivations (T_1, A_1) and (T_2, A_2) to be equivalent *iff* the corresponding templates T_1, T_2 and alignments A_1, A_2 are equivalent.

Intuitively, two templates T_1, T_2 are equivalent if they can generate the same space of equation systems – i.e., for every assignment of values to slots of T_1, there exists an assignment of values to slots of T_2 such that they generate the same equation systems. For instance, template (2) and (3) below are equivalent

$$m = \mathtt{A} + \mathtt{B}n \quad m = \mathtt{C} - n \tag{2}$$

$$m + n = \mathtt{A} \quad m - \mathtt{C}n = \mathtt{B}. \tag{3}$$

because after renaming $(\mathtt{A}, \mathtt{B}, \mathtt{C})$ to $(\mathtt{B}, \mathtt{C}, \mathtt{A})$ respectively in template (2), and algebraic manipulations, it is identical to template (3). We can see that any assignment of values to *corresponding* slots will result in the same equation system.

Similarly, two alignments A_1 and A_2 are equivalent if corresponding slots from each template align to the same textual number. For the above example, the alignment $\{1 \to \mathtt{A}, 3 \to \mathtt{B}, 4 \to \mathtt{C}\}$ in template (2), and alignment $\{1 \to \mathtt{B}, 3 \to \mathtt{C}, 4 \to \mathtt{A}\}$ in template (3) are equivalent. Note that the alignment $\{1 \to \mathtt{A}, 3 \to \mathtt{B}, 4 \to \mathtt{C}\}$ for (2) is not equivalent to $\{1 \to \mathtt{A}, 3 \to \mathtt{B}, 4 \to \mathtt{C}\}$ in (3), because it does not respect variable renaming. Our definition also allows two alignments to be equivalent, if they use textual numbers in equivalent positions for corresponding slots (as described by EquivTNum field).

Algorithm 1 Evaluating Derivation

```
Input: Predicted $(T_p, A_p)$ and gold $(T_g, A_g)$ derivation
Output: 1 if predicted derivation is correct, 0 otherwise
 1: if $|\mathcal{C}(T_p)| \neq |\mathcal{C}(T_g)|$ then        ▷ different # of coeff. slots
 2:     return 0
 3: end if
 4: $\Gamma \leftarrow$ TEMPLEQUIV($T_p$,$T_g$)
 5: if $\Gamma = \emptyset$ then                      ▷ not equivalent templates
 6:     return 0
 7: end if
 8: if ALIGNEQUIV($\Gamma, A_p, A_g$) then              ▷ Check alignments
 9:     return 1
10: end if
11: return 0
12: ------------------------------------------------------------
13: procedure TEMPLEQUIV($T_1$, $T_2$)
14:                 ▷ Note that here $|\mathcal{C}(T_1)| = |\mathcal{C}(T_2)|$ holds
15:     $\Gamma \leftarrow \emptyset$
16:     for each 1-to-1 mapping $\gamma : \mathcal{C}(T_1) \to \mathcal{C}(T_2)$ do
17:         match ← True
18:         for $t = 1 \cdots R$ do                        ▷ $R$ : Rounds
19:             Generate random vector $\mathbf{v}$
20:             $A_1 \leftarrow \{(\mathbf{v}_i \to c_i)\}$,$A_2 \leftarrow \{(\mathbf{v}_i \to \gamma(c_i))\}$
21:             if $\mathtt{Solve}(T_1, A_1) \neq \mathtt{Solve}(T_2, A_2)$ then
22:                 match ← False; break
23:             end if
24:         end for
25:         if match then $\Gamma \leftarrow \Gamma \cup \{\gamma\}$
26:     end for
27:     return $\Gamma$      ▷ $\Gamma \neq \emptyset$ iff the templates are equivalent
28: end procedure
29: ------------------------------------------------------------
30: procedure ALIGNEQUIV($\Gamma, A_1, A_2$)
31:     for mapping $\gamma \in \Gamma$ do
32:         if following holds true,

        $(q, c) \in A_1 \iff \{(q, \gamma(c))$ or $(q', \gamma(c))\} \in A_2$

33:         where $(q', q) \in$ EquivTNum
34:         then return 1
35:         end if
36:     end for
37:     return 0
38: end procedure
```

In the following, we carefully explain how template and alignment equivalence are determined algorithmically. Algorithm 1 shows the complete algorithm for comparing two derivations.

Template Equivalence We propose an approximate procedure TEMPLEQUIV (line 13) that detects equivalence between two templates. The procedure relies on the fact that under appropriate renaming of coefficients, two equivalent templates will generate equations which have the same solutions, for all possible coefficient assignments.

For two templates T_1 and T_2, with the same number of coefficients $|\mathcal{C}(T_1)| = |\mathcal{C}(T_2)|$, we represent a choice of renaming coefficients by γ, a

1-to-1 mapping from $\mathcal{C}(T_1)$ to $\mathcal{C}(T_2)$. The two templates are equivalent if there exists a γ such that solutions of the equations identified by T_1 and T_2 are same, for all possible coefficient assignments. The TEMPLEQUIV procedure exhaustively tries all possible renaming of coefficients (line 16), checking if the solutions of the equation systems generated from a random assignment (line 19) match exactly. It declares equivalence if for a renaming γ, the solutions match for $R = 10$ such random assignments.[4] The procedure returns all renamings Γ of coefficients between two templates under which they are equivalent (line 27). We discuss its effectiveness in §3.

Alignment Equivalence The TEMPLEQUIV procedure returns every mapping γ in Γ under which the templates were equivalent (line 4). Recall that γ identifies corresponding slots, c and $\gamma(c)$, in T_1 and T_2 respectively. We describe alignment equivalence using these mappings.

Two alignments A_1 and A_2 are equivalent if corresponding slots (according to γ) align to the same textual number. More formally, if we find a mapping γ such that for each tuple (q, c) in A_1 there is $(q, \gamma(c))$ in A_2, then the alignments are equivalent (line 33). We allow for equivalent textual numbers (as identified by EquivTNum field) to match when comparing tuples in alignments.

The proof of correctness of Algorithm 1 is sketched in the appendix. Using Algorithm 1, we can define *derivation accuracy*, to be 1 if the predicted derivation (T_p, A_p) and the reference derivation (T_g, A_g) are equivalent, and 0 otherwise.

Properties of Derivation Accuracy By comparing derivations, we can ensure that the following errors are detected by the evaluation.

Firstly, correct solutions found using incorrect equations will be penalized, as the template used will not be equivalent to reference template. Secondly, correct equation system obtained by an incorrect template will also be penalized for the same reason. Lastly, if the solver uses the correct template to get the correct equation system, but aligns the wrong number to a slot, the alignment will not be equivalent to the reference alignment, and the solver will be penalized too.

We will see some illustrative examples of above errors in §5.3. Note that the currently popular evaluation metric of solution accuracy will not detect *any* of these error types.

3 Annotating Derivations

As none of the existing benchmarks contain derivation annotations, we decided to augment existing datasets with these annotations. We also annotated DRAW-1K, a new dataset of 1000 general algebra word problems to make our study more comprehensive. Below, we describe how we reduced annotation effort by semi-automatic generated some annotations.

Annotating gold derivations from scratch for all problems is time consuming. However, not all word problems require manual annotation – sometimes all numbers appearing in the equation system can be uniquely aligned to a textual number without ambiguity. For such problems, the annotations are generated automatically.[5] We identify word problems which have at least one *alignment ambiguity* – multiple textual numbers with the same value, which appears in the equation system. A example of such a problem is shown in Figure 1, where there are three textual numbers with value 5, which appears in the equation system. Statistics for the number of word problems with such ambiguity is shown in Table 2.

We only ask annotators to resolve such alignment ambiguities, instead of annotating the entire derivation. If more than one alignments are genuinely correct (as in word problem of Table 1), we ask the annotators to mark both (using the *EquivTNum* field). This ensures our derivation annotations are *exhaustive* – all correct derivations are marked. With the correct alignment annotations, templates for all problems can be easily induced.

Annotation Effort To estimate the effort required to annotate derivations, we timed our annotators when annotating 50 word problems (all involved alignment ambiguities). As a control, we also asked annotators to annotate the entire derivation from scratch (i.e., only provided with the word problem and equations), instead of only fixing alignment ambiguities. When annotating from scratch, annotators took an average of 4 minute per word problem, while when fixing alignment ambiguities this time dropped to average of 1 minute

[4] Note that this procedure is a Monte-Carlo algorithm, and can be made more precise by increasing R. We found making R larger than 10 did not have an impact on the empirical results.

[5] Annotations for *all* problems are manually verified later.

Dataset	Draw-1k	Alg-514	Dolphin-L
# problems	1000	514	832
w/ ambiguity	21%	23%	35%
vocab.	2.21k	1.83k	0.33k
	Number of Templates		
before	329	30	273
after	224	24	203
% reduction	32%	20%	25%

Table 2: Statistics of the datasets. At least 20% of problems in each dataset had alignment ambiguities that required human annotations. The number of templates before and after annotation is also shown (reduction $> 20\%$).

per word problem. We attained a inter-annotator agreement of 92% (raw percentage agreement), with most disagreements arising on EquivTNum field.[6]

Reconciling Equivalent Templates The number of templates has been used as a measure of dataset diversity (Shi et al., 2015; Huang et al., 2016), however prior work did not reconcile the equivalent templates in the dataset. Indeed, if two templates are equivalent, we can replace one with the other and still generate the correct equations. Therefore, after getting human judgements on alignments, we reconcile all the templates using TemplEquiv as the final step of annotation.

TemplEquiv is quite effective (despite being approximate), reducing the number of templates by at least 20% for all datasets (Table 2). We did not find any false positives generated by the TemplEquiv in our manual examination. The reduction in Table 2 clearly indicates that equivalent templates are quite common in all datasets, and number of templates (and hence, dataset diversity) can be significantly overestimated without proper reconciliation.

4 Experimental Setup

We describe the three datasets used in our experiments. Statistics comparing the datasets is shown in Table 2. In total, our experiments involve over 2300 word problems.

Alg-514 The dataset Alg-514 was introduced in (Kushman et al., 2014). It consists of 514 general algebra word problems ranging over a variety of narrative scenarios (distance-speed, object counting, simple interest, etc.).

Dolphin-L Dolphin-L is the linear-T2 subset of the Dolphin dataset (Shi et al., 2015), which focuses on *number word problems* – algebra word problems which describe mathematical relationships directly in the text. All word problems in the linear-T2 subset of the Dolphin dataset can be solved using linear equations.

Draw-1k *D*ive*r*se *A*lgebra *W*ord (Draw-1k), consists of 1000 word problems crawled from `algebra.com`. Details on the dataset creation can be found in the appendix. As Alg-514 was also crawled from `algebra.com`, we ensured that there is little overlap between the datasets.

We randomly split Draw-1k into train, development and test splits with 600, 200, 200 problems respectively. We use 5-fold cross validation splits provided by the authors for Dolphin-L and Alg-514.

4.1 Evaluation

We compare derivation accuracy against the following evaluation metrics.

Solution Accuracy We compute solution accuracy by checking if each number in the reference solution appears in the generated solution (disregarding order), following previous work (Kushman et al., 2014; Shi et al., 2015).

Equation Accuracy An approximation of derivation accuracy that is similar to the one used in Kushman et al. (2014). We approximate the reference derivation $\tilde{\boldsymbol{z}}$ by randomly chosen from the (several possible) derivations which lead to the gold $\boldsymbol{y}$ from $\boldsymbol{x}$. Derivation accuracy is computed against this (possibly incorrect) reference derivation. Note that in equation accuracy, the approximation is used instead of annotated derivation. We include the metric of equation accuracy in our evaluations to show that human annotated derivation is necessary, as approximation made by equation accuracy might be problematic.

4.2 Our Solver

We train a solver using a simple modeling approach inspired by Kushman et al. (2014) and Zhou et al. (2015). The solver operates as follows. Given a word problem, the solver ranks all templates seen during training, Γ_{train}, and selects the set of the top-k (we use $k = 10$) templates $\Pi \subset \Gamma_{train}$. Next, all possible derivations $\mathcal{D}(\Pi)$ that use a template from Π are generated

[6] These were adjudicated on by the first author.

Setting	Soln. Acc.	Eqn. Acc.	Deriv. Acc.
Alg-514			
TE	76.2	72.7	75.5
TD	78.4	73.9	77.8
TD - TE	2.2	1.2	**2.3**
Draw-1k			
TE	52.0	48.0	48.0
TD	55.0	48.0	53.0
TD - TE	3.0	0	**5.0**
Dolphin			
TE	55.1	50.1	44.2
TD	57.5	36.8	54.9
TD - TE	2.4	-13.3	**10.7**

Table 3: TE and TD compared using different evaluation metrics. Note that while TD is clearly superior to TE due to extra supervision using the annotations, only derivation accuracy is able to correctly reflect the differences.

and scored. The equation system $\hat{\boldsymbol{y}}$ identified by highest scoring derivation $\hat{\boldsymbol{z}}$ is output as the prediction. Following (Zhou et al., 2015), we do not model the alignment of nouns phrases to variables, allowing for tractable inference when scoring the generated derivations. The solver is trained using a structured perceptron (Collins, 2002). We extract the following features for a $(\boldsymbol{x}, \boldsymbol{z})$ pair,

Template Features. Unigrams and bigrams of lemmas and POS tags from the word problem $\boldsymbol{x}$, conjoined with $|\mathcal{Q}(\boldsymbol{x})|$ and $|\mathcal{C}(T)|$.

Alignment Tuple Features. For two alignment tuples, $(q_1, c_1), (q_2, c_2)$, we add features indicating whether c_1 and c_2 belong to the same equation in the template or share the same variable. If they belong to the same sentence, we also add lemmas of the nouns and verbs between q_1 and q_2 in $\boldsymbol{x}$.

Solution Features. Features indicating if the solution of the system identified by the derivation are integer, negative, non-negative or fractional.

5 Experiments

Are solution and equation accuracy equally capable as derivation accuracy at distinguishing between good and bad models? To answer this question, we train the solver under two settings such that one of the settings has clear advantage over the other, and see if the evaluation metrics reflect this advantage. The two settings are,

Setting	Soln. Acc.	Eqn. Acc.	Deriv. Acc.
Draw-1k + Alg-514			
TE	32.5	31.5	29.5
TE*	60.5	56.0	54.0
TD	62.0	53.0	59.5
TD - TE*	1.5	-3.0	**5.5**
Draw-1k + Dolphin			
TE	41.0	37.5	37.5
TE*	58.5	55.5	51.5
TD	60.0	53.0	58.0
TD - TE*	1.5	-2.5	**6.5**

Table 4: When combining two datasets, it is essential to reconcile templates across datasets. Here TE* denotes training on equations after reconciling the templates, while TE simply combines datasets naively. As TE* represents a more appropriate setting, we compare TE* and TD in this experiment.

TE (Train on Equation) Only the $(\boldsymbol{x}, \boldsymbol{y})$ pairs are provided as supervision. Similar to (Kushman et al., 2014; Zhou et al., 2015), the solver finds a derivation which agrees with the equation system and the solution, and trains on it. Note that the derivation found by the solver may be incorrect.

TD (Train on Derivation) $(\boldsymbol{x}, \boldsymbol{z})$ pairs obtained by the derivation annotation are used as supervision. This setting trains the solver on human-labeled derivations. Clearly, the TD setting is a more informative supervision strategy than the TE setting. TD provides the correct template and correct alignment (i.e. labeled derivation) as supervision and is expected to perform better than TE, which only provides the question-equation pair.

We first present the main results comparing different evaluation metrics on solvers trained using the two settings.

5.1 Main Results

We compare the evaluation metrics in Table 3. We want to determine to what degree each evaluation metric reflects the superiority of TD over TE.

We note that solution accuracy always exceeds derivation accuracy, as a solver can sometimes get the right solutions even with the wrong derivation. Also, solution accuracy is not as sensitive as derivation accuracy to improvements in the solver. For instance, solution accuracy only changes by 2.4 on Dolphin-L when comparing TE and TD, whereas derivation accuracy changes by 10.7 points. We found that the large gap on Dolphin-L was due to several alignment errors in the predicted derivations, which were detected by derivation accuracy. Recall that over 35% of the problems in Dolphin-L have alignment ambiguities (Table 2). In the TD setting, many of these errors made by our solver were corrected as the gold alignment was part of supervision.

Equation accuracy too has several limitations. For DRAW-1K, it cannot determine which solver is better and assigns them the same score. Furthermore, it often (incorrectly) considers TD to be a worse setting than TE, as evident from decrease in the scores (for instance, on DOLPHIN-L). Recall that equation accuracy attempts to approximate derivation accuracy by choosing a random derivation agreeing with the equations, which might be incorrect.

Study with Combining Datasets With several ongoing annotation efforts, it is a natural question to ask is whether we can leverage multiple datasets in training to generalize better. In Table 4, we combine DRAW-1K's train split with other datasets, and test on DRAW-1K's test split. DRAW-1K's test split was chosen as it is the largest test split with general algebra problems (recall Dolphin-L contains only number word problems).

We found that in this setting, it was important to reconcile the templates *across* datasets. Indeed, when we simply combine the two datasets in the TE setting, we notice a sharp drop in performance (compared to Table 3). However, if we reconciled all templates and then used the new equations for training (called TE* setting in Table 4), we were able to see improvements from training on more data. We suspect difference in annotation style led to several equivalent templates in the combined dataset, which got resolved in TE*. Therefore, in Table 4, we compare TE* and TD settings.[7]

In Table 4, a trend similar to Table 3 can be observed – solution accuracy assigns a small improvement to TD over TE*. Derivation accuracy clearly reflects the fact that TD is superior to TE*, with a larger improvement compared to solution accuracy (eg., 5.5 vs 1.5). Equation accuracy, as before, considers TD to be worse than TE*.

Note that this experiment also shows that differences in annotation styles across different algebra problem datasets can lead to poor performance when combining these datasets naively. Our findings suggest that derivation annotation and template reconciliation are crucial for such multi-data supervision scenarios.

Dataset	Ours	KAZB	Best Result
ALG-514	76.2	68.7	**79.7** (ZDC)
DOLPHIN-L	55.1	37.5	46.3‡ (SWLLR)
DRAW-1K	52.0	43.2	–

Table 5: Comparison of our solver and other state-of-the-art systems, when trained under TE setting. All numbers are solution accuracy. See footnote for details on the comparison to SWLLR.

5.2 Comparing Solvers

To ensure that the results in the previous section were not an artifact of any limitations of our solver, we show here that our solver is competitive to other state-of-the-art solvers, and therefore it is reasonable to assume that similar results can be obtained with other automatic solvers.

In Table 5, we compare our solver to KAZB, the system of Kushman et al. (2014), when trained under the existing supervision paradigm, TE (i.e., training on equations) and evaluated using solution accuracy. We also report the best scores on each dataset, using ZDC and SWLLR to denote the systems of Zhou et al. (2015) and Shi et al. (2015) respectively. Note that our system and KAZB are the only systems that can process all three datasets without significant modification, with our solver being clearly superior to KAZB.

5.3 Case Study

We discuss some interesting examples from the datasets, to show the limitations of existing metrics, which derivation accuracy overcomes.

Correct Solution, Incorrect Equation In the following example from the DOLPHIN-L dataset, by choosing the correct template and the wrong alignments, the solver arrived at the correct solutions, and gets rewarded by solution accuracy.

> The sum of 2(q_1) numbers is 25(q_2). 12(q_3) less than 4(q_4) times one(q_5) of the numbers is 16(q_6) more than twice(q_7) the other number. Find the numbers.

[7] In TE*, the model still trains only using equations, without access to derivations. So TD is still better than TE*.

‡SWLLR also had a solver which achieves 68.0, using over 9000 semi-automatically generated rules tailored to number word problems. We compare to their similarity based solver instead, which does not use any such rules, given that the rule-based system cannot be applied to general word problems.

Note that there are seven textual numbers ($q_1, \ldots, q_7$) in the word problem. We can arrive at the correct equations ($\{m + n = 25, 4m - 2n = 16 + 12\}$), by the correct derivation,

$$m + n = q_2 \quad q_4 m - q_7 n = q_6 + q_3.$$

However, the solver found the following derivation, which produces the incorrect equations ($\{m + n = 25, 2m - n = 2 + 12\}$),

$$m + n = q_2 \quad \mathbf{q_1} m - \mathbf{q_5} n = \mathbf{q_7} + q_3.$$

Both the equations have the same solutions ($m = 13, n = 12$), but the second derivation is clearly using incorrect reasoning.

Correct Equation, Incorrect Alignment In such cases, the solver gets the right equation system, but derived it using wrong alignment. Solution accuracy still rewards the solver. Consider the problem from the DOLPHIN-L dataset,

> The larger of two(q_1) numbers is 2(q_2) more than 4(q_3) times the smaller. Their sum is 67(q_4). Find the numbers.

The correct derivation for this problem is,

$$m - q_3 n = q_2 \quad m + n = q_4.$$

However, our system generated the following derivation, which although results in the exact same equation system (and thus same solutions), is clearly incorrect due incorrect choice of ”two”,

$$m - q_3 n = \mathbf{q_1} \quad m + n = q_4.$$

Note that derivation accuracy will penalize the solver, as the alignment is not equivalent to the reference alignment (q_1 and q_2 are not semantically equivalent textual numbers).

Bad Approx. in Equation Accuracy The following word problem is from the ALG-514 dataset:

> Mrs. Martin bought 3(q_1) cups of coffee and 2(q_2) bagels and spent 12.75(q_3) dollars. Mr. Martin bought 2(q_4) cups of coffee and 5(q_5) bagels and spent 14.00(q_6) dollars. Find the cost of one(q_7) cup of coffee and that of one(q_8) bagel.

The correct derivation is,

$$q_1 m + q_2 n = q_3 \quad q_4 m + q_5 n = q_6.$$

However, we found that equation accuracy used the following incorrect derivation for evaluation,

$$q_1 m + \mathbf{q_2} n = q_3 \quad \mathbf{q_2} m + q_5 n = q_6.$$

Note while this derivation does generate the correct equation system and solutions, the derivation utilizes the wrong numbers and misunderstood the word problem. This example demonstrates the needs to evaluate the quality of the word problem solvers using the annotated derivations.

6 Related Work

We discuss several aspects of previous work in the literature, and how it relates to our study.

Existing Solvers Current solvers for this task can be divided into two broad categories based on their inference approach – *template-first* and *bottom-up*. Template-first approaches like (Kushman et al., 2014; Zhou et al., 2015) infer the derivation $\boldsymbol{z} = (T, A)$ *sequentially*. They first predict the template T and then predict alignments A from textual numbers to coefficients. In contrast, bottom-up approaches (Hosseini et al., 2014; Shi et al., 2015; Koncel-Kedziorski et al., 2015) *jointly* infer the derivation $\boldsymbol{z} = (T, A)$. Inference proceeds by identifying parts of the template (eg. $\mathtt{A}m + \mathtt{B}n$) and aligning numbers to it ($\{2 \rightarrow \mathtt{A}, 3 \rightarrow \mathtt{B}\}$). At any intermediate state during inference, we have a partial derivation, describing a fragment of the final equation system ($\mathtt{2m} + \mathtt{3n}$). While our experiments used a solver employing the template-first approach, it is evident that performing inference in all such solvers requires constructing a derivation $\boldsymbol{z} = (T, A)$. Therefore, annotated derivations will be useful for evaluating *all* such solvers, and may also aid in debugging errors.

Other reconciliation procedures are also discussed (though briefly) in earlier work. Kushman et al. (2014) reconciled templates by using a symbolic solver and removing pairs with the same canonicalized form. Zhou et al. (2015) also reconciled templates, but do not describe how it was performed. We showed that reconciliation is important for correct evaluation, for reporting dataset complexity, and also when combining multiple datasets.

Labeling Semantic Parses Similar to our work, efforts have been made to annotate semantic parses for other tasks, although primarily for providing supervision. Prior to the works of Liang et al. (2009) and Clarke et al. (2010), semantic parsers were trained using annotated logical forms (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Wong and Mooney, 2007, *inter alia*), which were expensive to annotate. Recently, Yih et al. (2016) showed that labeled semantic parses for the knowledge based question answering task can be obtained at a cost comparable to obtaining answers. They showed significant improvements in performance of a question-answering system using the labeled parses instead of answers for training. More recently, by treating word problems as a semantic parsing task, Upadhyay et al. (2016) found that joint learning using both explicit (derivation as labeled semantic parses) and implicit supervision signals (solution as responses) can significantly outperform models trained using only one type of supervision signal.

Other Semantic Parsing Tasks We demonstrated that response-based evaluation, which is quite popular for most semantic parsing problems (Zelle and Mooney, 1996; Berant et al., 2013; Liang et al., 2011, *inter alia*) can overlook reasoning errors for algebra problems. A reason for this is that in algebra word problems there can be several semantic parses (i.e., derivations, both correct and incorrect) that can lead to the correct solution using the input (i.e., textual number in word problem). This is not the case for semantic parsing problems like knowledge based question answering, as correct semantic parse can often be identified given the question and the answer. For instance, paths in the knowledge base (KB), that connect the answer and the entities in the question can be interpreted as legitimate semantic parses. The KB therefore acts as a constraint which helps prune out possible semantic parses, given only the problem and the answer. However, such KB-based constraints are unavailable for algebra word problems.

7 Conclusion and Discussion

We proposed an algorithm for evaluating derivations for word problems. We also showed how derivation annotations can be easily obtained by only involving annotators for ambiguous cases. We augmented several existing benchmarks with derivation annotations to facilitate future comparisons. Our experiments with multiple datasets also provided insights into the right approach to combine datasets – a natural step in future work. Our main finding indicates that derivation accuracy leads to a more accurate assessment of algebra word problem solvers, finding errors which other metrics overlook. While we should strive to build such solvers using as little supervision as possible for training, having high quality annotated data is essential for correct evaluation.

The value of such annotations for evaluation becomes more immediate for online education scenarios, where such word solvers are likely to be used. Indeed, in these cases, merely arriving at the correct solution, by using incorrect reasoning may prove detrimental for teaching purposes. We believe derivation based evaluation closely mirrors how humans are evaluated in schools (by forcing solvers to show "their work").

Our datasets with the derivation annotations have applications beyond accurate evaluation. For instance, certain solvers, like the one in (Roy and Roth, 2015), train a *relevance classifier* to identify which textual numbers are relevant to solving the word problem. As we only annotate relevant numbers in our annotations, our datasets can provide high quality supervision for such classifiers. The datasets can also be used in evaluation test-beds, like the one proposed in (Koncel-Kedziorski et al., 2016).

We hope our datasets will open new possibilities for the community to simulate new ideas and applications for automatic problem solvers.

Acknowledgments

The first author was supported on a grant sponsored by DARPA under agreement number FA8750-13-2-0008. We would also like to thank Subhro Roy, Stephen Mayhew and Christos Christodoulopoulos for useful discussions and comments on earlier versions of the paper.

References

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA, October. Association for Computational Linguistics.

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*, pages 591–614. ACM.

Peter Clark and Oren Etzioni. 2016. My computer is an honor student but how intelligent is it? standardized tests as a measure of ai. *AI Magazine*, 37(1):5–12.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from the world's response. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*, pages 18–27, Uppsala, Sweden, July. Association for Computational Linguistics.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8. Association for Computational Linguistics, July.

Mohammad Javad 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, Doha, Qatar, October. Association for Computational Linguistics.

Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. 2016. 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. Association for Computational Linguistics.

Bo Kang, Arun Kulshreshth, and Joseph J. LaViola Jr. 2016. Analyticalink: An interactive learning environment for math word problem solving. In *Proceedings of the 21st International Conference on Intelligent User Interfaces*, pages 419–430. ACM.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. 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*, pages 1152–1157, San Diego, California, June. 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, Baltimore, Maryland, June. Association for Computational Linguistics.

Percy Liang, Michael Jordan, and Dan Klein. 2009. Learning semantic correspondences with less supervision. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 91–99, Suntec, Singapore, August. Association for Computational Linguistics.

Percy Liang, Michael Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 590–599, Portland, Oregon, USA, June. 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 *Proc. of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Allen Newell, John C. Shaw, and Herbert A. Simon. 1959. Report on a general problem-solving program. In *IFIP Congress*, volume 256, page 64.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752, Lisbon, Portugal, September. Association for Computational Linguistics.

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.

Shyam Upadhyay, Ming-Wei Chang, Kai-Wei Chang, and Wen-tau Yih. 2016. Learning from Explicit and Implicit Supervision Jointly For Algebra Word Problems. In *Proceedings of EMNLP*, pages 297–306.

Yuk Wah Wong and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967, Prague, Czech Republic, June. Association for Computational Linguistics.

Wen-tau Yih, Matthew Richardson, Chris Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual*

Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 201–206, Berlin, Germany, August. Association for Computational Linguistics.

J. M. Zelle and R. J. Mooney. 1996. Learning to Parse Database Queries using Inductive Logic Programming. In *AAAI*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005*, pages 658–666.

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.

A Creating DRAW-1K

We crawl over 100k problems from `http://algebra.com`. The 100k word problems include some problems which require solving nonlinear equations (e.g. finding roots of quadratic 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. In our crawler, we use simple pattern matching rules to extract all the equations in the explanation. 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.[8]

1000 problems were randomly chosen from these pool of 12k problems, which were then shown to annotators as described earlier to get the derivation annotations.

B Proof of Correctness (Sketch)

For simplicity, we will assume that EquivTNum is empty. The proof can easily be extended to handle the more general situation.

Lemma 1. The procedure TEMPLEQUIV returns $\Gamma \neq \emptyset$ iff templates T_1, T_2 are equivalent (w.h.p.).

Proof First we prove that with high probability we are correct in claiming that a γ found by the algorithm leads to equivalence. Let probability of getting the same solution even when the template are not equivalent be $\epsilon(T_1, T_2, \gamma) < 1$. The probability that solution is same for R rounds for T_1, T_2 which are not equivalent is $\leq \epsilon^R$, which can be made arbitrarily small by choosing large R. Therefore, with a large enough R, obtaining $\Gamma \neq \emptyset$ from TEMPLEQUIV implies there is a γ under which templates generate equations with the same solution, and by definition, are equivalent.

Conversely, if templates are equivalent, it implies $\exists \gamma^*$ such that under that mapping for any assignment, the generated equations have the same solution. As we iterate over all possible 1-1 mappings γ between the two templates, we will find γ^* eventually.

Proposition Algorithm 1 returning 1 implies derivations (T_p, A_p) and (T_g, A_g) are equivalent.

Proof Algorithm returns 1 only if TEMPLEQUIV found a $\Gamma \neq \emptyset$, and $\exists \gamma \in \Gamma$, following holds

$$(q, c) \in A_g \iff (q, \gamma(c)) \in A_p$$

i.e., the corresponding slots aligned to the same textual number. TEMPLEQUIV found a $\Gamma \neq \emptyset$ implies templates are equivalent (w.h.p). Therefore, $\exists \gamma \in \Gamma$ such that the corresponding slots aligned to the same textual number implies the alignments are equivalent under mapping γ. Together they imply that the derivation was equivalent (w.h.p.).

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