Learning Fine-Grained Expressions to Solve Math Word Problems

Danqing Huang[1]*, **Shuming Shi**[2], **Jian Yin**[1], **and Chin-Yew Lin**[3]
{huangdq2@mail2,issjyin@mail}.sysu.edu.cn
shumingshi@tencent.com
cyl@microsoft.com
[1] Guangdong Key Laboratory of Big Data Analysis and Processing, Sun Yat-sen University
[2]Tencent AI Lab [3] Microsoft Research

Abstract

This paper presents a novel template-based method to solve math word problems. This method learns the mappings between math concept phrases in math word problems and their math expressions from training data. For each equation template, we automatically construct a rich template sketch by aggregating information from various problems with the same template. Our approach is implemented in a two-stage system. It first retrieves a few relevant equation system templates and aligns numbers in math word problems to those templates for candidate equation generation. It then does a fine-grained inference to obtain the final answer. Experiment results show that our method achieves an accuracy of 28.4% on the linear Dolphin18K benchmark, which is 10% (54% relative) higher than previous state-of-the-art systems while achieving an accuracy increase of 12% (59% relative) on the TS6 benchmark subset.

1 Introduction

The research topic of automatically solving math word problems dates back to the 1960s (Bobrow, 1964a,b; Charniak, 1968). Recently many systems have been proposed to these types of problems (Kushman et al., 2014; Hosseini et al., 2014; Koncel-Kedziorski et al., 2015; Zhou et al., 2015; Roy and Roth, 2015; Shi et al., 2015; Upadhyay et al., 2016; Mitra and Baral, 2016). On a recent evaluation conducted by Huang et al. (2016), current state-of-the-art systems only achieved an 18.3% accuracy on their published dataset Dolphin18K. Their results indicate that math word problem solving is a very challenging task.

To solve a math word problem, a system needs to understand natural language text to extract information from the problem as local context. Also, it should provide an external knowledge base, including commonsense knowledge (e.g. *"a chicken has two legs"*) and mathematical knowledge (e.g. *"the perimeter of a rectangle = 2 * length + 2 * width"*). The system can then perform reasoning based on the above two resources to generate an answer.

P1: *What's 25% off $139.99?*
Equation: (1-0.25)*139.99 = x

P2: *How much will the ipod now be if the original price is $260 and I get 10% discount?*
Equation: (1-0.1)*260 = x

Template: $(1-n_1)*n_2 = x$

P3: *I bought something for $306.00 dollars. I got a 20% discount. What was the original price?*
Equation: (1-0.2)*x = 306

Template: $(1-n_1)*x = n_2$

Figure 1: Math Word Problem Examples.

In this paper, we focus on the acquisition of mathematical knowledge, or deriving math concepts from natural language. Consider the first two problems $P1$ and $P2$ in Figure 1. The math concept in the problems tells you to take away a percentage from one and get the resulting percentage of a total. Using mathematical language, it can be formulated as $(1-n_1)*n_2$, where n_1, n_2 are quantities. In this example, we can derive the concept of subtraction from the text "*[NUM] % off*" and "*[NUM] % discount*".

*Work done while this author was an intern at Microsoft Research.

Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 805–814
Copenhagen, Denmark, September 7–11, 2017.

Acquisition of mathematical knowledge is non-trivial. Initial statistical approaches (Hosseini et al., 2014; Roy and Roth, 2015; Koncel-Kedziorski et al., 2015) derive math concepts based on observations from their dataset of specific types of problems, e.g. problems with one single equation. For example, Hosseini et al. (2014) assumes verbs and only verbs embed math concepts and map them to addition/subtraction. Roy and Roth (2015); Koncel-Kedziorski et al. (2015) assume there is only one unknown variable in the problem and cannot derive math concepts involving constants or more than one unknown variables, such as "*the product of two unknown numbers*".

Template-based approaches (Kushman et al., 2014; Zhou et al., 2015; Upadhyay et al., 2016), on the other hand, leverage the built-in composition structure of equation system templates to formulate all types of math concepts seen in training data, such as $(1 - n_1) * n_2 = x$ in Figure 1. However, they suffer from two major shortcomings. First, the math concepts they learned, which is expressed as an entire template, fails to capture a lot of useful information with sparse training instances. We argue that it would be more expressive if the math concept is learned in a finer granularity. Second, their learning processes rely heavily on lexical and syntactic features, such as the dependency path between two slots in a template. When applied to a large-scale dataset, they create a huge and sparse feature space and it is unclear how these template-related features would contribute.

To alleviate the sparseness problem of math concept learning and better utilize templates, we propose a novel approach to capture rich information contained in templates, including textual expressions that imply math concepts. We parse the template into a tree structure and define "template fragment" as any subtree with at least one operator and two operands. We learn fine-grained mappings between textual expressions and template fragments, based on longest common substring. For example, given the three problems in Figure 1, we can map "*[NUM] %* off" and "*[NUM] %* discount" to $1 - n_1$, and "*[NUM] %* off *[NUM]*" to $(1 - n_1) * n_2 = x$. In this way, we can decompose the templates and learn math concepts in a finer grain. Furthermore, we observe that problems of the same template share some common properties. By aggregating problems of the same template and capturing these properties, we automatically construct a sketch for each template in the training data.

Our approach is implemented in a two-stage system. We first retrieve a few relevant templates in the training data. This narrows our search space to focus only on those templates that are likely to be relevant. Then we align numbers in the problem to those few returned templates, and do fine-grained inference to obtain the final answer. We show that the textural expressions and template sketch we propose are effective for both stages. In addition, our system significantly reduces the hypothesis space of candidate equations compared to previous systems, which benefits the learning process and inference at scale.

We evaluate our system on the benchmark dataset provided by Huang et al. (2016). Experiments show that our system outperforms two state-of-the-art baselines with a more than 10% absolute (54% relative) accuracy increase in the linear benchmark and a more than 20% absolute (71% relative) accuracy increase for the dataset with a template size greater than or equal to 6.

In the remaining parts of this paper, we introduce related work in Section 2, describe template sketch and textual expression learning in Section 3, present our two-stage system in Section 4, summarize experiment setup and results in Section 5, and conclude this paper in Section 6.

2 Related Work

Automatic math word problem solving methods (Bobrow, 1964a,b; Charniak, 1968, 1969; Briars and Larkin, 1984; Fletcher, 1985; Dellarosa, 1986; Bakman, 2007; Yuhui et al., 2010) developed before 2008 are mostly rule-based. They accept limited well-format input sentences and map them into certain structures by pattern matching. They usually focus on problems with simple math operations such as addition or subtraction. Please see Mukherjee and Garain (2008) for a summary.

In recent years, symbolic and statistical methods have been explored by various researchers. In the symbolic approach, systems transform math word problems to structured representations. Bakman (2007) maps math problems to predefined schema with a table of textual formulas and changing verbs. Liguda and Pfeiffer (2012) uses augmented semantic networks to represent math problems. Shi et al. (2015) parses math problems to

their pre-defined semantic language. However, these methods are only effective in their designated math problem categories and are not scalable to other categories. For example, the method used by Shi et al. (2015) works extremely well for solving number word problems but not others.

In the statistical machine learning approach, Hosseini et al. (2014) solves addition and subtraction problems by extracting quantities as states and derive math concepts from verbs in the training data. Kushman et al. (2014) and Zhou et al. (2015) generalize equations attached to problems with variable slots and number slots. They learn a probabilistic model for finding the best solution equation. Upadhyay et al. (2016) follows their approach and leverage math word problems without equation annotation as external resources. Seo et al. (2015) solves a set of SAT geometry questions with text and diagram provided. Koncel-Kedziorski et al. (2015) and Roy and Roth (2015) target math problems that can be solved by one single linear equation. They map quantities and words to candidate equation trees and select the best tree using a statistical learning model. Mitra and Baral (2016) considers addition and subtraction problems in three basic problem types: "Change", "Part Whole" and "Comparison". They manually design different features for each type, which is difficult to expand to more types.

In summary, previous methods can achieve high accuracy in limited math problem categories, (i.e. (Kushman et al., 2014; Shi et al., 2015)), but do not scale or perform well in datasets containing various math problem types as in Huang et al. (2016), as their designed features are becoming sparse. Their process of acquiring mathematical knowledge is either sparse or based on certain assumptions of specific problem types. To alleviate this problem, we introduce our template sketch construction and fine-grained expressions learning in the next section.

3 Template Sketch Construction

A template sketch contains template information. We define three categories of information for the sketch shown in this section. Next we describe how we construct a template sketch, via aggregation of rich information from training problems. We group problems of the same template in training set as one cluster and collect information. See Figure 2 for the outline of our template sketch construction.

3.1 Definition

Template: It is first introduced in Kushman et al. (2014). It is a unique form of an equation system. For example, given an equation system as follows:

$$2 \cdot x_1 + 4 \cdot x_2 = 34$$
$$x_1 + 4 = x_2$$

This equation system is a solution for a specific math word problem. We replace the numbers with four number slots $\{n_1, n_2, n_3, n_4\}$ and generalize the equations to the following template:

$$n_1 \cdot x_1 + n_2 \cdot x_2 = n_3$$
$$x_1 + n_4 = x_2$$

Alignment: We align numbers in the math problem with the number slots of a template. For the first math problem in Figure 1 with its corresponding template $(1 - n_1) * n_2 = x$, there are two numbers 0.25 and 139.99 to align with two number slots n_1 and n_2, which results in two different alignments.

Kushman et al. (2014) aligns nouns to variable slots $\{x_1, x_2, ...\}$ which leads to a huge hypothesis space and does not perform as well as the number slot alignment only method proposed later by (Zhou et al., 2015). Therefore, we only consider number slot alignment in this paper.

3.2 Textual Expressions

For template fragments, there are usually some textual expressions. For example, "n_1 % off" and "n_1 % discount" are both mapped to the template fragment $1 - n_1$.

We employ a statistical framework to automatically mine textual expressions for template fragments from a training dataset. First we parse the equation to a hierarchical tree. In a bottom-up approach, we obtain each possible subtree as a template fragment t_k, which associates with at least one number slot. For each t_k, we use the numbers to anchor the number-related phrases in the problem, replace numbers with"*[NUM]*" and noun phrases with "*[VAR]*", and cluster the phrases $P = \{p_1, p_2, \cdots\}$ with the same t_k across all data given all training problems. Then we compute the longest common substring lcs_{ij}^k between pairs p_i and p_j and calculate tf-idf score of lcs_{ij}^k. We keep the lcs_{ij}^k with scores above certain empirically determined threshold as the textual expressions.

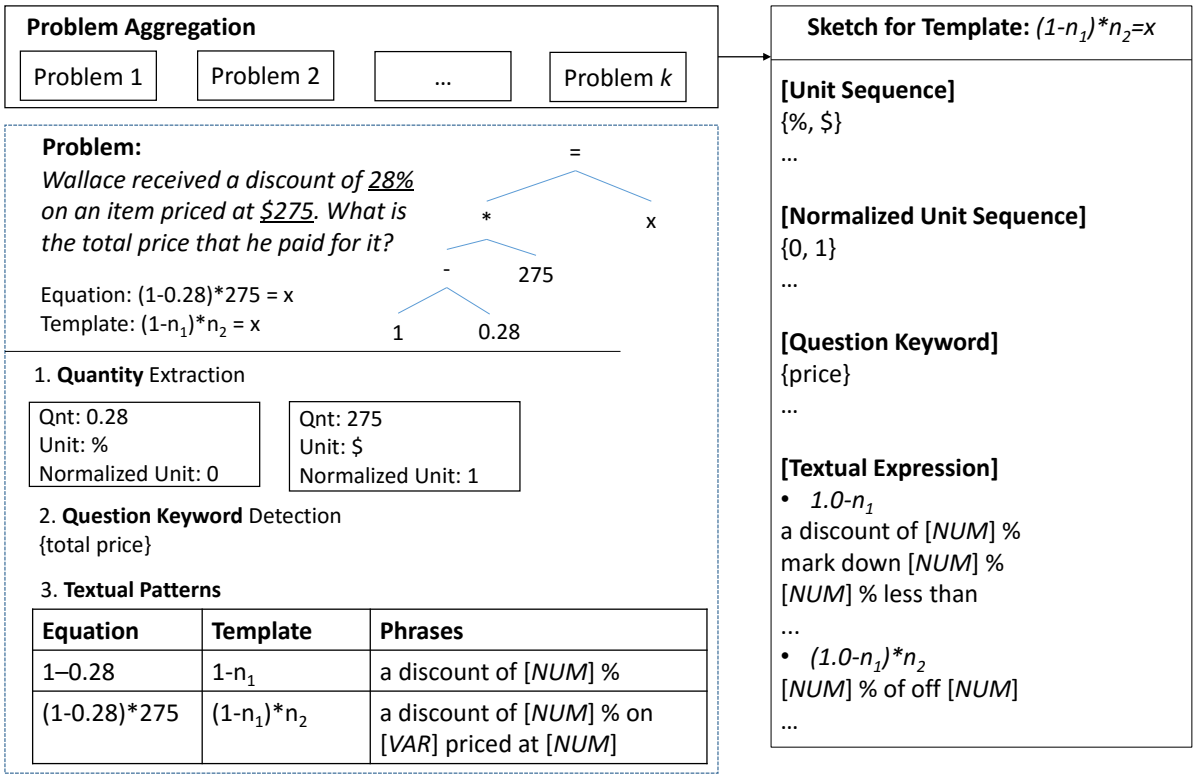

Figure 2: Template Sketch Construction.

3.3 Slot Type

Number slots in templates have their own type of constraints. For example, in the template $(1-n_1)*n_2 = x$, usually n_1 represents a percentage quantity and n_2 is the quantity of an object.

We model slot types with quantity units, and find the direct governing noun phrase as its 'owner'. For the problem in Figure 2, we extract quantity unit sequence as $\{\%, \$\}$, normalized unit sequence as $\{0, 1\}$ (because % and $ are of different quantity types), and quantity owners as {discount, item}. The slot type information provides important clues to choose the correct template and alignment.

3.4 Question Keyword

Question keyword decides which template we use. Given the following problem setting: "A rectangle has a width of 5cm and a length of 10cm.", we can ask either Q1:"What is the *area* of the rectangle?" or Q2: "What is the *difference* between width and length?". The question keywords *area* and *difference* help our system to decide if is should apply template $n_1 * n_2 = x$ for Q1 and apply template $n_1 - n_2 = x$ for Q2.

We first detect the question sentence (containing keywords "what","how","figure out"...). Then we extract the question keyword on the dependency tree with simple rules that we observed in the dev set (e.g. retrieving nouns with "$attr - nsubj$" dependency relation with keyword "what"). Please note that we favor recall over precision of our detected question keywords since they are used as features instead of hard constrains on template decision. Simple rule-based extraction can already satisfy our need for detecting question keywords in math problems.

4 Two-Stage System

In this section, we describe our two-stage system for solving math problems, including template retrieval and alignment ranking. We show how to apply textual expressions and template sketch to our system.

4.1 Template Retrieval

We use an efficient retrieval module to first narrow our search space and focus only on templates that are likely to be relevant. Let χ denote the set of test problems, and $T = \{t_1, t_2, \ldots, t_j\}$ as the template set in the training data. For each test problem x_i, our goal is to select the correct template

t_j. We define the conditional probability of selecting a template given a problem as follows:

$$p(t_j|x_i;\nu_t) = \frac{\exp(\nu_t \cdot f(x_i,t_j))}{\sum_{t'_j \in T} \exp(\nu_t \cdot f(x_i,t'_j))}$$

where ν_t is the model parameter and $f(x_i,t_j)$ is the feature vector. We apply the Ranking SVM (Herbrich et al., 2000) to minimize a regularized margin-based pairwise loss. We then have the following objective function:

$$\frac{1}{2}\|\nu_t\|^2 + C\sum_i l(\nu_t^T f(x_i,t_j)^+ - \nu_t^T f(x_i,t_l)^-)$$

where superscript "+" indicates the correct instance and "-" indicates the false ones. We use the loss function $l(t) = max(0, 1-t)^2$.

To construct the vector $f(x_i,t_j)$ for template t_j, we use the three categories in the template sketch shown in Table 1. Let $Q(t_j)$ represent the cluster of training problems with template t_j.

Textual Features
Contains textual expressions in each template fragments?
Average Word Overlap with $Q(t_j)$
Max Word Overlap with $Q(t_j)$
Quantity Features
Unit sequence in $Q(t_j)$
Normalized unit sequence in $Q(t_j)$
Question Features
Is Question keyword in $Q(t_j)$

Table 1: Features for template retrieval.

At the phrase level, as we have mined different expressions in 3.2 for slots in templates, we can extract the phrases related to each number or number pair in a test problem and match them with expressions. For example, given a test problem to match template $(1-n_1)*n_2 = x$ in Figure 2, we have two groups of patterns to match, corresponding to $1.0 - n_1$ and $(1.0-n_1)*n_2$ respectively.

Quantity types in a problem are important. We use the unit type sequences and normalized unit type sequence for describing number slot types in a template. In addition, if a number unit type cannot differentiate each number slot, we will make use of number "owner" as defined in subsection 3.3. For example, in the sentence "The width is 3cm and the length is 5cm", we extract two quantities with unit type sequence {cm, cm}; and owner {width, length}.

In addition, we consider question keywords for templates. For example, if the question keyword is "difference", then $x + n_1 = n_2$ will have a higher probability of being selected than $x = n_1 * n_2$.

We observe that in some cases, one word difference can lead to two different templates. To consider cases in which some templates are very similar (e.g. $x + n_1 = n_2$ and $n_1 + n_2 = x$, part/whole unknown), we retrieve the top ranked N (N=3) templates as candidates for alignment in the next stage.

4.2 Alignment Ranking

For each top N templates from the previous stage, we generate possible alignments $A = \{a_1, a_2, \ldots, a_m\}$ as the candidate equation system for the test problem x_i. We train a ranking model to choose the alignment with the highest probability $p(a_k|x_i,t_j;\nu_a)$, where ν_a is the model parameter vector.

$$p(a_k) = \frac{\exp(\nu_a \cdot f(x_i,a_k))}{\sum_{a'_k \in A} \exp(\nu_{a'} \cdot f(x_i,a'_k))}$$

We use the same ranking model as in template selection stage and the objective function is changed to:

$$\frac{1}{2}\|\nu_a\|^2 + C\sum_i l(\nu_a^T f(x_i,a_k)^+ - \nu_a^T f(x_i,a_l)^-)$$

We design more fine-grained features for each number slot to formulate the alignment feature vector $f(x_i,a_k)$. It contains the following features in Table 2.

Textual Features
Match textual expressions in template fragment aligned to each number slot (pair)
Quantity Features
Aligned unit sequence in $Q(t_j)$
Aligned normalized unit sequence in $Q(t_j)$
Relationship with noun phrase
Optimal number 1 or 2 is used?
Solution Features
Is integer solution?
Is positive solution?

Table 2: Features for alignment ranking.

At the textual level, we want to capture textual expressions describing each number slot. For example, in the template $(1-n_1)*n_2 = x$, we have

mined patterns of $1 - n_1$ in 3.2, such as "a discount of n_1 %", "mark down n_1 %", etc. Given the problem in Figure 2 as the test problem, alignment (**1-0.28**) $* 275 = x$ matches textual expressions, while (**1-275**) $* 0.28 = x$ does not.

For quantity features, we use the alignment-ordered unit sequence. For the problem in Figure 2 mapping to template $(1 - n_1) * n_2 = x$, we have two different alignments: $\{n_1$:0.28, n_2:275$\}$, $\{n_1$:275,n_2:0.28$\}$. Their aligned unit sequences are $\{\%, \$\}$ and $\{\$,\%\}$ respectively. We also use the relations of quantities with noun phrases to differentiate number slot interaction with unknown variable slots and number slots, such as $n_1 * x$ and $n_1 * n_2$.

Some templates have numerical solution properties while others do not. For example, template $x_1 = (n_1 - n_2)/(n_3 - n_4)$ would be less likely to have any strong indication of integer solution properties. We count the percentage of integer/positive solutions from the corresponding problems as the probability that this template prefers an integer/positive solution.

4.3 Model Discussion

Our method has two main differences from previous template-based methods (Kushman et al., 2014; Zhou et al., 2015; Upadhyay et al., 2016).

First, previous methods implicitly model mapping from problem text to templates. We learn fine-grained textual expressions mapped to template fragments; and explicitly model the property of templates with template sketches. Second, previous methods align numbers for all templates in a training set, while we only examine the N most probable templates. This significantly reduces the equation candidate search space. Given a problem in which m numbers align with a template of n number slots, the number of possible equation candidates would be A_m^n. The search space grows linearly with the number of templates in the training data. Suppose $m = 5$, $n = 4$ and we have 1000 templates, the total space would be $(5 * 4 * 3 * 2) * 1000 = 120,000$ for one problem in Zhou et al. (2015), and will be much larger if it considers unknown variable alignment as in (Kushman et al., 2014).

5 Experiments

Settings As demonstrated in Huang et al. (2016), previous datasets for math problems are limited in both scale and diversity. We conduct our experiment on their dataset Dolphin18K. We use the linear subset, containing 10,644 problems in total.

We use two baseline systems for comparison: (1) ZDC (Zhou et al., 2015) is a statistical learning method that is an improved version of KAZB (Kushman et al., 2014)[1]. (2) SIM (Huang et al., 2016) is a simple similarity based method. We do not compare other systems because they only solve one specific type of problem, e.g. Hosseini et al. (2014) only handle addition/subtraction problems and Koncel-Kedziorski et al. (2015) aim to solve problems with one single linear equation.

Experiments are conducted using 5-fold cross-validation with 80% problems randomly selected as training data and the remaining 20% for testing. We report the solution accuracy.

5.1 Overall Evaluation Results

Table 3 shows the overall performance of different systems. In the table, the size of a template is the number of problems corresponding to a template. For example, for templates with a size 100 or larger, their problem counts add up to 1,807.

Template Size	problems	ZDC (%)	SIM (%)	Ours (%)
>=100	1807	34.2	29.7	64.5
>=50	4281	31.1	27.2	39.3
>=20	5392	29.4	25.8	36.9
>=10	6216	25.3	24.6	35.7
>=6	6827	21.7	20.2	**34.6**
>=5	7081	21.6	20.1	34.3
>=4	7262	21.1	19.8	33.8
>=3	7466	20.7	19.7	33.2
>=2	8229	20.6	20.3	32.2
>=1	10644	17.9	18.4	**28.4**

Table 3: Overall evaluation results.

From the table, we observe that our model consistently achieves better performance than the baselines on all template sizes. As the template size becomes larger, all three systems achieve better performance. When template size equals 6 (TS6, as a de-facto template size constrain adopted in ZDC), our model achieve an absolute increase of over 12% (59% relative). This demonstrates the effectiveness of our proposed method.

[1]We ignore KAZB because it does not complete running on the dataset in three days

When including long tail problems with a template size less than 2, performance of all three systems drop significantly. This is because the templates of these problems are not seen in the training set, and thus are difficult to solve using these template-based methods. Still, we have at least 10% absolute (54% relative) accuracy increase on the whole test set compared to the two baselines. Previous template-based methods require templates size larger than 6 in the data as constraints. From the result, we can see that our method relaxes the template size constraint and matches more problems with less frequent templates.

5.2 Accuracy per Template

Here we investigate the performance of different templates. In Table 4, we sample some dominant templates and report their accuracies. For our model, we report both template retrieval accuracy and final solution accuracy.

As we can see, our method performs better than the baselines for most dominant templates. Performance of the dominant templates can reach an accuracy level of 60%. This proves that our template sketch and textual expressions are effective in capturing rich template information.

To our surprise, some templates tend to perform better than others even with smaller template sizes. For example, $x_1 = n_1 - n_2$, which represents the subtraction problem, has 63 problems but performs not as well as $x_1 = (n_1 - n_2)/(n_3 - n_4)$ which has 48 problems. We look into their corresponding problems and find out that $x_1 = n_1 - n_2$ are applied to more themes in natural language than $x_1 = (n_1 - n_2)/(n_3 - n_4)$, which are almost about the theme of "coordinate slope".

In our model, there is a gap between template retrieval accuracy and final solution accuracy, which means that although we select the correct template candidates for the problem, the alignment model cannot rank the equations correctly.

5.3 Two-Stage Evaluation

Next, we evaluate the performance of our two-stage system. Accuracy of template retrieval and alignment ranking is shown in Table 5.

For template retrieval accuracy, Hit@N means the correct template for a problem is included in the top N list returned by our model. We estimate the best achievable performance by using oracle template retrieval. The result is **47.1%** (Hit@ALL), which means 47.1% of the templates exist more than once in the problem set. Please note that our template retrieval evaluation may be underestimated, since in some cases, a test problem can be solved by different templates.

We then use the top N templates as input for both our alignment ranking and ZDC. From the table, we have the following observations: (1) Hit@3 performs better compared to Hit@1 for both systems. This confirms our claim that some templates are similar and we need to incorporate more fine-grained features to differentiate in the alignment step; (2) It obtains the highest accuracy when $N = 3$ and decreases when N gets larger. Both systems get benefits from our template retrieval which helps retrieve relevant templates and reduce the hypothesis space of equations; (3) Given the same N templates input, our alignment ranking achieves better performance than ZDC. This implies that our features are more indicative.

5.4 Feature Ablation

This section describes our feature ablation study.

Template Retrieval In Table 6, we conduct three configurations against our model (FULL). Each ablated configuration corresponds to one category of our template sketch. From the table, we can see that all three categories of features contribute to system performance. We remove QUANTITY results in the worse performance comparing to the FULL model.

Alignment Ranking In Table 7, N means to select the top N templates in the previous stage for alignment. The column "Correct Template" represents how well the alignment model can perform given the correct template input for alignment. Our alignment model (FULL) performs the best compared to the three ablated settings, which confirms the effectiveness of template properties.

5.5 Error Analysis

We have observed that template-based methods have difficulty solving problems with small template sizes, especially for cases that have a single problem instance (i.e. template size $= 1$). We sample 100 problems in which our system makes mistakes in the dev set of Dolphin18K and summarize them in Table 8.

Quantity Type The types of quantities are difficult to determined. For the example problem in

				Ours	
Template	problems	ZDC (%)	SIM (%)	Template retrieval Acc (%)	Final Acc (%)
$n_1 * x_1 = n_2$	548	26.3	23.9	87.0	58.7
$n_1/x_1 = n_2/n_3$	453	21.4	29.8	94.1	61.5
$x_1 = n_1 * n_2$	403	23.6	28.0	78.9	63.4
$n_1 * x_1 + n_2 * x_2 = n_3$; $x_1 + x_2 = n_4$	300	86.3	69.7	94.9	85.8
$x_1 = n_1 * n_2 * n_3$	103	22.3	32.0	67.0	55.0
$x_1 + x2 = n_1$ $x1 - x_2 = n2$	80	39.7	48.8	79.4	65.1
$x_1 = n_1 - n_2$	63	11.7	15.9	50.7	23.4
$x_1 = (n_1 - n_2)/(n_3 - n_4)$	48	14.9	18.8	95.7	89.4

Table 4: Accuracy Per Template. Template retrieval acc reports percent of templates appears in one of the top 3 templates returned by our method.

Hit@N	1	2	3	4	5	10	20	50	ALL
Template retrieval Acc (%)	17.5	22.4	26.3	27.2	28.0	30.2	32.7	35.2	47.1
Final Acc (%)	24.9	27.6	**28.4**	27.9	27.4	25.3	22.3	22.1	20.1
ZDC (%)	19.5	20.1	20.1	19.9	19.8	19.1	18.9	18.6	17.9

Table 5: Results of template retrieval and final accuracy with different top N templates retrieved.

Model	Hit @1 (%)	Hit @3 (%)	Hit @10 (%)
FULL	17.5	26.3	30.2
-TEXTUAL	14.1	24.7	28.4
-QUANTITY	11.4	23.4	25.9
-QUESTION	16.9	25.4	29.8

Table 6: Feature ablation of template retrieval.

Model	Correct Template (%)	N= 1 (%)	N= 3 (%)
FULL	34.5	24.9	28.4
-TEXTUAL	31.9	22.2	25.1
-QUANTITY	29.2	20.9	23.3
-SOLUTION	26.3	18.7	21.2

Table 7: Feature ablation of alignment ranking.

the table, if we can detect "24 male" is the same as "men", the problem can be solved.

Relation/State Detection If we can identify the changed states or mathematical relations between variables, we can solve this category of problems. In the example problem, it is important to understand that "commission is taken out" is my money state.

External Knowledge This requires specific mathematical models, such as permutation and combination, or commonsense knowledge, e.g. a dice has 6 sides.

Equation Decomposition The limitation of template-based approaches is that they require test problems belonging to one of the templates seen in training. Thus, for problems corresponding to template sizes less than 2, we can decompose templates into smaller units and conduct learning more precisely. We then need to generate the equations, which is also a challenge.

6 Conclusion and Future Work

In this paper, we propose a novel approach to solving math word problems with rich information of templates. We learn mappings between textual expressions and template fragments. Furthermore, we automatically construct sketches for each template. We implement a two-stage system, including template retrieval and alignment ranking. Experiments show that our method performs signifi-

Category	Math Problem
Quantity Type (10%)	The ratio of **women** to **men** in a certain club is 3 to 2. If there are 24 **male** club members, then how many female club members are there?
Relation/State Detection (12%)	If I am selling something for $25,000 and **a 7% commission is taken out**, how much money will I **be left** with?
External Knowledge (23%)	Find the probability that total score is 10 or more given at least one dice show 6 if 2 dice red & blue thrown?
Equation Decomposition (55%)	The average weight of A, B and C is 45 kg. If the average weight of A and B is 40 kg and that of B and C is 43 kg, the weight of B is?

Table 8: Error Categorization.

cantly better than two state-of-the-art systems.

Based on our error analysis, we plan to improve our model by detecting quantity types more accurately, learning relations and incorporating commonsense knowledge. For long tail problems with a template size less 2, we want to utilize the fine-grained expressions we have learned and decompose equations for learning. Then we can reason with small equation units to generate a final equation in testing. We would like to leverage semantic parsing and transform math problems to a more structured representation. Additionally, we plan to apply our findings to generating math problem.

7 Acknowledgments

This work is supported by the National Natural Science Foundation of China (61472453, U1401256, U1501252, U1611264). Thanks to the anonymous reviewers for their helpful comments and suggestions.

References

Yefim Bakman. 2007. Robust understanding of word problems with extraneous information. Http://arxiv.org/abs/math/0701393.

Daniel G. Bobrow. 1964a. Natural language input for a computer problem solving system. Technical report, Cambridge, MA, USA.

Daniel G. Bobrow. 1964b. Natural language input for a computer problem solving system. Ph.D. Thesis.

Diane J. Briars and Jill H. Larkin. 1984. An integrated model of skill in solving elementary word problems. *Cognition and Instruction*, 1(3):245–296.

Eugene Charniak. 1968. Carps, a program which solves calculus word problems. Technical report.

Eugene Charniak. 1969. Computer solution of calculus word problems. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, pages 303–316.

Denise Dellarosa. 1986. A computer simulation of children's arithmetic word-problem solving. *Behavior Research Methods, Instruments, & Computers*, 18(2):147–154.

Charles R. Fletcher. 1985. Understanding and solving arithmetic word problems: A computer simulation. *Behavior Research Methods, Instruments, & Computers*, 17(5):565–571.

Ralf Herbrich, Thore Graepel, and Klaus Obermayer. 2000. *Large Margin Rank Boundaries for Ordinal Regression*, chapter 7.

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*.

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 52nd Annual Meeting of the Association for Computational Linguistics*.

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

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*.

Christian Liguda and Thies Pfeiffer. 2012. Modeling math word problems with augmented semantic networks. In *Natural Language Processing and Information Systems. International Conference on Applications of Natural Language to Information Systems (NLDB-2012)*, pages 247–252.

Arindam Mitra and Chitta Baral. 2016. Learning to use formulas to solve simple arithmetic problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

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

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. The Association for Computational Linguistics.

Minjoon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. 2015. Solving geometry problems: Combining text and diagram interpretation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

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*.

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 the 2016 Conference on Empirical Methods in Natural Language Processing*.

Ma Yuhui, Zhou Ying, Cui Guangzuo, Ren Yun, and Huang Ronghuai. 2010. Frame-based calculus of solving arithmetic multistep addition and subtraction word problems. *Education Technology and Computer Science, International Workshop*, 2:476–479.

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*.