ComSearch: Equation Searching with Combinatorial Mathematics for Solving Math Word Problems with Weak Supervision

Anonymous ACL submission

Abstract

Previous studies have introduced a weakly-supervised paradigm for solving math word problems requiring only the answer value annotation. While these methods search for correct value equation candidates as pseudo labels, they search among a narrow sub-space of the enormous equation space. To address this problem, we propose a novel search algorithm with combinatorial mathematics **ComSearch**, which can compress the search space by excluding mathematical equivalent equations. The compression allows the searching algorithm to enumerate all possible equations and obtain high-quality data. Experimental results show that our method achieves state-of-the-art results, especially for problems with more variables.

1 Introduction

Solving math word problems (MWPs) is the task of extracting a mathematical solution from problems written in natural language. In Figure 1, we present an example of MWP. Based on a sequence-to-sequence (seq2seq) framework that takes in the text descriptions of the MWPs and predicts the answer equation (Wang et al., 2017), task specialized encoder and decoder architectures (Wang et al., 2018b, 2019; Xie and Sun, 2019; Liu et al., 2019; Guan et al., 2019; Zhang et al., 2020b,a; Shen and Jin, 2020), data augmentation and normalization (Wang et al., 2018a; Liu et al., 2020), pre-trained models (Tan et al., 2021; Liang et al., 2021; Shen et al., 2021) and various other studies have been conducted on *full supervision* setting of the task. This setting requires equation expression annotation, which is expensive and time-consuming.

Recently Hong et al. (2021) and Chatterjee et al. (2021) addressed this problem and proposed the *weak supervision* setting, where only the answer value annotation is given for supervision. These methods first extract candidate equations that obtain the correct value and then use them as pseudo labels to train the MWP solving model. However, the solution space is enormous with the bruce-force searching used in these two studies, i.e., $O(n^{2n})$ with n variables. When the number of variables increases, it becomes computationally impossible to traverse all possible equations due to the high computational complexity. Hong et al. (2021) searches among neighbour equations of the wrong model prediction in the solution space via random walk, which lacks robustness and highly relies on initialization. Chatterjee et al. (2021) trains a candidate equation extraction model by using reinforcement learning (RL) to explore the solution space, where the reward is given by whether the equation obtains the correct value. The rewards are sparse in the enormous search space, resulting in relatively low coverage of 14.5% of the examples. Even with beam search, it can only cover 80.1% of the examples.

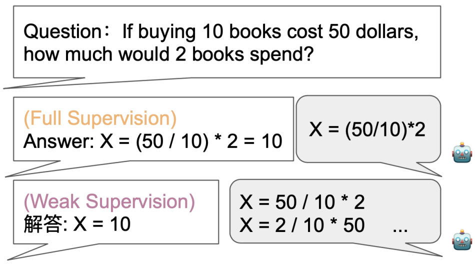

Figure 1: Example of MWP solving system under full supervision and weak supervision.

We observe that although the search space is ample, many equations in the search space are equivalent. For example, $a - b + c + d$ has various mathematically equivalent forms $a - (b - c - d)$, $a + c + d - b$ and so on. The search space could be compressed if such equivalence could be excluded in the searching algorithm. Supported by theories in combinatorial mathematics, we propose a new searching method that searches through only non-equivalent equations in the search space. Our method could be proven to have an approximate complexity of $O(n^n)$, allowing the algorithm to

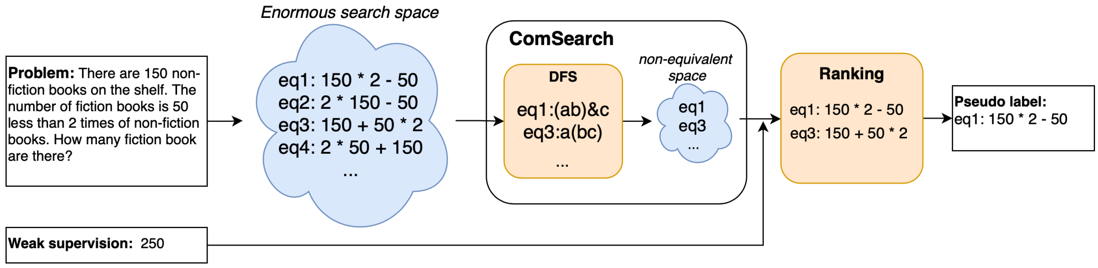

Figure 2: The model overview.

Algorithm 1 *enum_skel(n)*

Require: $n \geq 1$
Intialize empty list skels
for $i \leq n; \quad i = 1; \quad i++$ **do**
  left_list = unit_skel(i)
  right_list = enum_skels(n − i)
  for *left* in *left_list* **do**
    for *right* in *right_list* **do**
      move the start index of *right* to i
      new_skels += left + right
    end for
  end for
  skels += new_skels
end for
return *skels*

find all possible candidate equations with the given variables. We show that 77.5% percent of the examples have only one equation candidate and form high quality and reliable data. We build a ranking module to choose the best pseudo label for examples with multiple candidate equations. Our experimental results demonstrate the effectiveness of our method, achieve state-of-the-art results under the weakly supervised setting.

2 Methodology

We show the pipeline of our method in Figure 2. Our method consists of three modules: The Search with Combinatorial Mathematics (**ComSearch**) module that searches for candidate equations; the MWP model that is trained to predict equations given the natural language text and pseudo labels; the Ranking module that ranks which candidate equation should form the pseudo label.

2.1 ComSearch

Directly searching for non-equivalent equation expressions is difficult, because the searching method needs to consider Commutative law, Associative law and other equivalent forms. To enumerate all non-equivalent equations for four arithmetic operations, we transform the problem to finding skeleton structures that could be enumerated without repeat via deep-first search.

We define the set of non-equivalent equations using four arithmetic operations as S_n. We sort the set to two categories, either $S^{\pm}$ where the outermost operators are $\pm$, such as $a/b - c + d$ and $a+(b*c-d)$, or $S^{\divideontimes}$ where the outermost operators are $\divideontimes$, such as $(a+b)*(c-d/e)$ and $b*(a-c)$. We call the former a *general addition equation* and the latter a *general multiplication equation*:

$$S_n^{\pm} = \{(x_1 \divideontimes (..)) \pm (x_i \divideontimes (..)) \pm ..x_n\} \quad (1)$$

$$S_n^{\divideontimes} = \{(x_1 \pm (..)) \divideontimes (x_i \pm (..)) \divideontimes ..x_n\} \quad (2)$$

These two sets are symmetrical. Consider elements in $S_n^{\pm}$, we can rewrite the equation to x. Thus we can form a mapping $g(\cdot)$ from an general addition equation x to an skeleton structure expression $g(x)$. :

$$\begin{aligned} x = &((x_i \divideontimes (..)) + (x_j \divideontimes (..)) + ..) \\ &- ((x_k \divideontimes (..)) + (x_l \divideontimes (..)) + ..) \\ g(x) = &(x_i(..))(x_j(..))..\&(x_k(..))(x_l(..)).. \end{aligned}$$

The order of x_i within the same layer of brackets is ignored in $g(x)$, that it can deal with the equivalence caused by Commutative law and Associative law. The addition and substraction terms are split by &, that it can deal with equivalence cause by removing brackets. $g(x)$ is a bijection, so the enumeration problem transforms to finding such skeletons:

$$n = 2: ab, a\&b, b\&a$$

$$n = 3: abc, a\&(b\&c), (ab)\&c, ...$$

The enumeration problem of these structures is an expansion of solving Schroeder's fourth problem (Schröder, 1870), which calculates the number of labeled series-reduced rooted trees with n leaves. We use a deep-first search algorithm shown in Algorithm 1 to enumerate these skeletons. It considers the position of the first bracket and then recursively finds all possible skeletons of sub-sequences of the variable sequence $\mathcal{X} = \{x_k\}_{k=1}^{i}$ (Wang, 2021).

To be noticed, because there is at least one $+$ operator for each equation, the left side of $\&$ must not be empty while the right part has no restrictions. Thus we define the *unit_skel(i)* equation to return possible skeletons with only one or none $\&$ and no brackets. This constraint is equivalent to finding non-empty subsets and its complement of the variable sequence $\mathcal{X}$. The enumeration algorithm of non-empty subsets is trivial and omitted here.

$$unit\_skel(i) = \{(A\&\overline{A})|A \subseteq \mathcal{X}; A \neq \emptyset\} \quad (3)$$

We transform the skeletons back to equations to obtain all non-equivalent equations S_n. Given the compressed search space, we substitute the values for variables in the equation templates and use the equations which value matches with the answer number as candidate equations.

2.2 MWP Solving Model

We follow Hong et al. (2021) and Chatterjee et al. (2021) and use Goal-driven tree-structured MWP solver (GTS) (Xie and Sun, 2019) as the MWP model. GTS is a seq2seq model with the attention mechanism that uses a bidirectional long short term memory network (BiLSTM) as the encoder and LSTM as the decoder. GTS also uses a recursive neural network to encode subtrees based on its children nodes representations with the gate mechanism. With the subtree representations, this model can well use the information of the generated tokens to predict a new token.

2.3 Ranking

While ComSearch enumerates equations that are non-equivalent without repeat, some variable sets can coincidentally form multiple equations with the same correct value, as we show in Figure 2. The equations $150 * 2 - 50$ and $150 + 50 * 2$ are non-equivalent, their values are equal, while only $150 * 2 - 50$ is the correct solution.

To process these data, we build a ranking module to choose the best candidate equation. We first train the MWP model with the pseudo data that only one equation matches with the answer. Then we use the trained model to perform *self-learning* on the data with two or more candidate equations, and assign a score to each candidate and use the candidate with the highest score as the pseudo label of the example. We add the ranked data to the training data and re-train the model from scratch.

Model	Term	#	Prop(%)
-	All Data	23,162	-
Ours	Too Long	233	1.0
	Power Operator	51	0.2
	Single	17,959	77.5
	Multiple	3,931	17.0
	Data	21,890	**94.5**
WARM	Data (w/o beam)	-	14.5
	Data (w/ beam)	-	80.1

Table 1: Statistics of ComSearch Results.

#Variable	Bruce-Force	ComSearch	Ratio
1	1	1	1
2	8	6	1.3
3	192	68	2.8
4	9,216	1,170	7.9
5	737,280	27,142	27.2
6	88,473,600	793,002	111.6

Table 2: Empirical Results of Search Space Size.

3 Experiments

3.1 Dataset

We evaluate our proposed method on the Math23K dataset. It contains 23,161 math word problems annotated with solution expressions and answers. We only use the problems and final answers. We evaluate our method on both 5-fold cross validation and train-test setting of Wang et al. (2018a). The train-test setting is evaluated by the three-run average.

3.2 Statistics

We give statistics of ComSearch in Table 1. Among the 23,162 examples, 233 have more than 6 variables that we filter them out, and 51 use the power operation that our method is not applicable. 94.5% of the examples find at least one equation that can match the answer value, significantly higher than WARM, which covers only 80.1% of the examples. LBF dynamically searches for candidate equations, and this measurement is not applicable. 17,959 examples match with only one equation, and 3,931 examples match with two or more equations that need the ranking module to choose the pseudo label further. We give the distribution of the matched template in the appendix.

3.2.1 Compression of Search Space

We show the empirical compression of the search space with ComSearch in Table 2. As we can see, the compression ratio of ComSearch increases as the variable number grows, up to more than 100 times when the number of variables reaches 6.

The size of the Bruce-Force search space could be directly calculated, which is $n! * (n-1)! * 4^{n-1}$. If we consider the exponential generating function of $card(S_n)$, based on Smooth Implicit-function Schema, we can have an approximation of S_n: $card(S_n) \sim C * n^{n-1}$, which shows our searching method compresses the search space more than exponential level. We give proof in the appendix.

3.3 Results

We compare our weakly-supervised models' math word problem solving accuracy with two baselines methods in Table 3.

Chatterjee et al. (2021) proposed **WARM** that uses RL to train an equation candidate generation model with the reward of whether the value of the equation is correct. Since the reward signal is sparse due to the enormous search space, it uses beam search to further search candidates.

Hong et al. (2021) proposed **LBF**, a learning-by-fix algorithm that searches in neighbour space of the predicted wrong answer by random walk and tries to find a fix equation that holds the correct value as the candidate equation. *memory* saves the candidates of each epoch as training data.

We reproduced the results of LBF with their official code and found that LBF lacks robustness. We observe that its performance highly relies on the initialization of the model. When fewer candidates are extracted at early-stage training, the performance drops drastically since LBF relies on random walks in an enormous search space. Our method achieves state-of-the-art performance and outperforms other baselines up to 3.8% and 2.7% on train-test and cross-validation settings. Our method is also more robust with minor variance.

Model	Valid(%)	Test(%)	CV(%)
WARM	-	54.3	-
LBF	57.2($\pm$0.5)	55.4($\pm$0.5)	55.2($\pm$1.2)
+memory	56.6($\pm$6.9)	55.1($\pm$6.2)	56.3($\pm$6.2)
Ours	**60.1**($\pm$0.2)	**59.2**($\pm$0.3)	**59.0**($\pm$0.9)
Single	60.0	58.0	-
Random	57.3	56.3	-
GTS	-	75.6	74.3

Table 3: Results on Math23K. $\pm$ denotes the variance of 3 runs for valid/test, and 5 folds for Cross Validation.

#Var	LBF(%)	ComSearch(%)	Prop(%)
1	**75.0**	50.0	1.6
2	**75.2**	73.4	33.1
3	56.2	**62.9**	48.5
4	4.8	**25.8**	12.4
5	3.2	**16.1**	3.1
6	0	**28.6**	0.7
7	0	**25.0**	0.4

Table 4: Results of different number of variables.

3.3.1 Ablation Study

We perform an ablation study with train-test setting in Table 3. *Single* denotes using the 17,959 examples that only match with one equation, the model achieves 58.0% performance, which is slightly lower than using all data and the ranking module, out-performing other baseline models. This shows that the examples with only one matching could be considered highly reliable and achieve comparable performance with a smaller training data size. *Random* denotes removing the ranking module and randomly sampling an equation for the examples that match with two or more equations. We observe a performance drop of 2.9% point without the ranking module, showing that our ranking module improves the performance.

3.3.2 Study on Number of Variables

In Table 4, we show the comparison of model performance on examples of a different number of variables. For the examples with 1 or 2 variables, LBF has a slight performance advantage since the search space is small and nearly not compressed. While the variable number grows, our method achieves better performance on examples with more variables and larger search space, which demonstrates the efficiency of ComSearch.

4 Conclusion

This paper proposes ComSearch, a searching method based on Combinatorial Mathematics, to extract candidate equations for Solving Math Word Problems under weak supervision. ComSearch compresses the enormous search space of equations beyond the exponential level, allowing the algorithm to enumerate all possible non-equivalent equations to search for candidate equations. Our experiments show that our method obtains high-quality pseudo data for training, achieves state-of-the-art performance under weak supervision settings, outperforming strong baselines, especially for the examples with more variables.

A Proof for Search Space Approximation

Because there is at least one $+$ or $*$ operator for each equation (i.e. $-a-b-c$ is illegal), the target S_n is not symmetric and is hard to directly approximate. We need two assisting targets to form the approximate. This proof majorly relies on Flajolet and Sedgewick (2009).

We first consider target U that considers only $+$, $*$ and $/$ three operators. We sort it into two categories: U^+ that the outermost operator is $+$ and U^* that the outermost operator is $*$. Equations such as $\frac{1}{a} * \frac{1}{b-c}$ are still considered illegal.

We can have the construction of U:

$$U^+ = Z + SET_{\geq}(U^*) \tag{4}$$
$$U^* = Z + (2^2-1) * SET_{=2}(U^+) \tag{5}$$
$$+ (2^3-1) * SET_{=3}(U^+)... \tag{6}$$

We apply symbolic method to obtain the EGF of the constructions:

$$U^+(z) = z + \sum_{k\geq 2} \frac{1}{k!}[U^*(z)]^k \tag{7}$$
$$= z + [e^{U^*(z)} - 1 - U^*(z)] \tag{8}$$
$$U^*(z) = z + \sum_{k\geq 2} \frac{2^k-1}{k!}[U^+(z)]^k \tag{9}$$
$$= z + e^{2U^+(z)} - e^{U^+(z)} - U^+(z) \tag{10}$$

Meanwhile we have:

$$U(z) = U^+(z) + U^*(z) - z \tag{11}$$

Next we consider target T that $-a-b-c$ is considered legal. Similarly we define $T^{\pm}$ and T^*. We consider the construction:

$$T^{\pm} = 2Z + SET_{\geq}(T^*) \tag{12}$$
$$T^* = 2Z + 2[(2^2-1) * SET_{=2}(T^{\pm}/2) \tag{13}$$
$$+ (2^3-1) * SET_{=3}(T^{\pm}/2)...] \tag{14}$$

With symbolic method we have:

$$T^{\pm}(z) = 2z + \sum_{k\geq 2} \frac{1}{k!}[U^*(z)]^k \tag{15}$$
$$= 2z + [e^{T^*(z)} - 1 - T^*(z)] \tag{16}$$
$$T^*(z) = 2z + 2\sum_{k\geq 2} \frac{2^k-1}{k!}[T^{\pm}(z)/2]^k \tag{17}$$
$$= 2z + 2e^{T^{\pm}(z)} - 2e^{T^{\pm}(z)/2} - T^{\pm}(z) \tag{18}$$

The illegal equations such as $-a-b-c$ in T equals to the counts of $a+b+c$, which is actually U. So we have:

$$S(z) = T(z) - U(z) \tag{19}$$

We now have the EGF of S_n.

With Smooth implicit-function schema and Stirling approximiation function we have, for an EGF $y(z) = \sum_{n\geq 0} y_n z^n$, Let $G(z,w) = \sum_{m,n\geq 0} g_{m,n} z^m w^n$, thus $y(z) = G(z, y(z))$:

$$n! * [z^n]y(z) \sim \frac{c * n!}{\sqrt{2\pi n^3}} * r^{-n+1/2} \tag{20}$$
$$\sim \frac{c\sqrt{2\pi n r}}{\sqrt{2\pi n^3}} (\frac{1}{r})^n (\frac{n}{e})^n \tag{21}$$
$$= \frac{c\sqrt{r}}{n} (\frac{n}{re})^n \tag{22}$$

while r:

$$G(r,s) = s \tag{23}$$
$$\frac{\partial G(r,s)}{\partial w} = 1 \tag{24}$$

and c:

$$c = \sqrt{\frac{\partial G(r,s)/\partial z}{\partial^2 G(r,s)/\partial w^2}} \tag{25}$$

We still need the two assisting targets to perform the approximation. We have:

$$U^+(z) = e^{z + e^{2U^+(z)} - e^{U^+(z)} - U^+(z)} \tag{26}$$
$$- e^{2U^+(z)} + e^{U^+(z)} + U^+(z) - 1 \tag{27}$$

Let $G(z,w) = z + e^{2w} - e^w - ln(1 + e^{2w} - e^w)$, considering 23 and 25, r, s and c would be constant numbers.

So we have:

$$n![z^n]U^+(z) \sim \frac{c_1\sqrt{r_1}}{n} (\frac{n}{r_1 e})^n \tag{28}$$

Similarly we can approximate U^*, $T^{\pm}$ and T^*:

$$n![z^n]U^*(z) \sim \frac{c_2\sqrt{r_1}}{n} (\frac{n}{r_2 e})^n \tag{29}$$
$$n![z^n]T^{\pm}(z) \sim \frac{c_3\sqrt{r_2}}{n} (\frac{n}{r_3 e})^n \tag{30}$$
$$n![z^n]T^*(z) \sim \frac{c_4\sqrt{r_2}}{n} (\frac{n}{r_4 e})^n \tag{31}$$

So we have:

$$u_n = n![z^n]U(z) \sim \frac{(c_1+c_2)\sqrt{r_1}}{n} (\frac{n}{r_1 e})^n \tag{32}$$
$$t_n = n![z^n]T(z) \sim \frac{(c_3+c_4)\sqrt{r_2}}{n} (\frac{n}{r_2 e})^n \tag{33}$$

Since $S(z) = T(z) - U(z)$, the subtraction of u_n and t_n would be our approximation. However

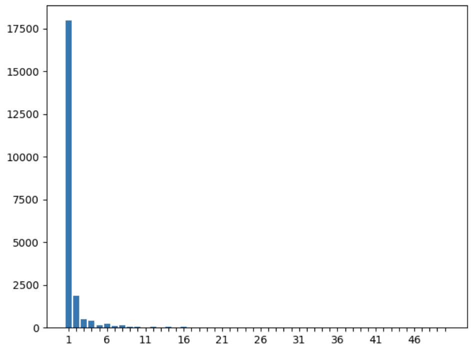

Figure 3: Distribution of Candidate Equation Number.

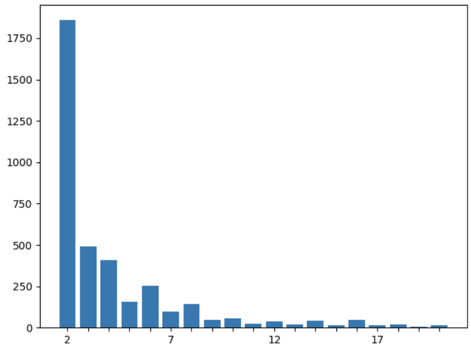

Figure 4: Distribution of Candidate Equation Number.

we observe that $r_1 \gg r_3$, that u_n can be ignored. So we have:

$$s_n = n![z^n]S(z) \sim \frac{(c_3 + c_4)\sqrt{r_2}}{n}(\frac{n}{r_2 e})^n \quad (34)$$

Q.E.D.

B Distribution of Candidate Equations

The largest candidate equation number of one example is 3914. We show the distribution of candidate equations in Figure 3 and 4. The x axis represent the the number of candidate, while the y axis represents the number of examples that have x candidate equations. We can see from Figure 3, which includes examples that have 1 to 50 candidates, it is a long tail distribution that most examples only have a few candidate equations. From Figure 4, where we zoom in and focus on examples that have 2 to 20 candidates, we can see that there are a lot of examples that have more than 2 candidate equations, and the ranking module is essential.

References

L. Carlitz and J. Riordan. 1956. The number of labeled two-terminal series-parallel networks. *Duke Mathematical Journal*, 23(3):435 – 445.

Oishik Chatterjee, Aashish Waikar, Vishwajeet Kumar, Ganesh Ramakrishnan, and Kavi Arya. 2021. A weakly supervised model for solving math word problems.

Philippe Flajolet and Robert Sedgewick. 2009. *Analytic Combinatorics*. Cambridge University Press.

Wenyv Guan, Qianying Liu, Guangzhi Han, Bin Wang, and Sujian Li. 2019. An improved coarse-to-fine method for solving generation tasks. In *Proceedings of the The 17th Annual Workshop of the Australasian Language Technology Association*, pages 178–185, Sydney, Australia. Australasian Language Technology Association.

Yining Hong, Qing Li, Daniel Ciao, Siyuan Huang, and Song-Chun Zhu. 2021. Learning by fixing: Solving math word problems with weak supervision. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(6):4959–4967.

W. Knödel. 1951. Über zerfällungen. *Monatshefte für Mathematik*, 55:20–27.

Yihuai Lan, Lei Wang, Qiyuan Zhang, Yunshi Lan, Bing Tian Dai, Yan Wang, Dongxiang Zhang, and Ee-Peng Lim. 2021. Mwptoolkit: An open-source framework for deep learning-based math word problem solvers.

Jierui Li, Lei Wang, Jipeng Zhang, Yan Wang, Bing Tian Dai, and Dongxiang Zhang. 2019. Modeling intrarelation in math word problems with different functional multi-head attentions. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6162–6167.

Zhenwen Liang, Jipeng Zhang, Jie Shao, and Xiangliang Zhang. 2021. Mwp-bert: A strong baseline for math word problems.

Qianying Liu, Wenyu Guan, Sujian Li, Fei Cheng, Daisuke Kawahara, and Sadao Kurohashi. 2020. Reverse operation based data augmentation for solving math word problems. *arXiv preprint arXiv:2010.01556*.

Qianying Liu, Wenyv Guan, Sujian Li, and Daisuke Kawahara. 2019. Tree-structured decoding for solving math word problems. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2370–2379, Hong Kong, China. Association for Computational Linguistics.

John Riordan and Claude E Shannon. 1942. The number of two-terminal series-parallel networks. *Journal of Mathematics and Physics*, 21(1-4):83–93.

Ernst Schröder. 1870. Vier combinatorische probleme. *Zeitschrift für Mathematik und Physik*, 15.

Jianhao Shen, Yichun Yin, Lin Li, Lifeng Shang, Xin Jiang, Ming Zhang, and Qun Liu. 2021. Generate & rank: A multi-task framework for math word problems. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2269–2279.

Yibin Shen and Cheqing Jin. 2020. Solving math word problems with multi-encoders and multi-decoders. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2924–2934, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Minghuan Tan, Lei Wang, Lingxiao Jiang, and Jing Jiang. 2021. Investigating math word problems using pretrained multilingual language models.

Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. 2018a. Translating a math word problem to a expression tree. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1064–1069.

Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. 2018b. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, Bing Tian Dai, and Heng Tao Shen. 2019. Template-based math word problem solvers with recursive neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7144–7151.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854, Copenhagen, Denmark. Association for Computational Linguistics.

Yun Wang. 2021. *The Math You Never Thought Of*. Posts & Telecom Press Co., Ltd., Beijing.

Zhipeng Xie and Shichao Sun. 2019. A goal-driven tree-structured neural model for math word problems. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 5299–5305.

Dongxiang Zhang, Lei Wang, Luming Zhang, Bing Tian Dai, and Heng Tao Shen. 2019. The gap of semantic parsing: A survey on automatic math word problem solvers. *IEEE transactions on pattern analysis and machine intelligence*, 42(9):2287–2305.

Jipeng Zhang, Roy Ka-Wei Lee, Ee-Peng Lim, Wei Qin, Lei Wang, Jie Shao, and Qianru Sun. 2020a. Teacher-student networks with multiple decoders for solving math word problem. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4011–4017. International Joint Conferences on Artificial Intelligence Organization. Main track.

Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. 2020b. Graph-to-tree learning for solving math word problems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3928–3937.