CS 447 Final Project Report: Universal Algebraic Word Problem Solver

Liming Wang
lwang114@illinois.edu
Xinran Wang
xinranw5@illinois.edu

1 Introduction

The goal of our project is to find the mathematic expression underlying a middleschool-level arithmetic problem given its textual description. We want to make our system to be domain-independent so no domain-specific rules or features are used by the model to learn to solve the problem. Also we want to make our solver end-to-end, so no supervision other than the problem text is used to train it. Further, all four basic mathematical operations are considered (addition, subtraction, multiplication and division). However, we make important assumptions based on the paper we try to implement for our baseline. Namely, the problem does not involve unknowns and can be solved by scanning through the question in one pass.

2 Related Work

Our project is largely based on the idea in [Roy and Roth(2015)], who proposes to use *expression tree* to represent equations and find the least-common-ancestor (LCA) operation between each quantity pair. They showed that the sets of LCA operations for equivalent tree representations of the same mathematical expression are the same under certain assumptions. Therefore, they decomposed the problem into a two-step process: first, train a classifier to score each LCA operations between quantities and second, use beam search to find the best expression tree whose set of LCA operations with the highest score. For problems with irrelevant quantities, they also trained a relevance filter to filter out the irrelevant quantities. The features used to train the classifiers will be discussed in the next section.

It is noteworthy that there are other approaches to solve math word problems. The first work that used quantity schema based on verb categories is done by [Hosseini et al.(2014)]. They annotated the verbs in the problems into several categories and train a classifier to learn the verb categories. Each verb category has its own state variables, such as quantities, units, unknowns, owner, etc. They implemented a finite-state automata that essentially "simulates" the change of quantities described by the problem text and derives the equation in the end.

[Roy and Roth(2016)] proposed a more elaborate approach to solve arithmetic problem using intermediate supervision, from which they derive a more precise quantity schema for each quantity based on the categories of its associated verb. They classify the quantity schema into four categories: transfer, dimensional analysis, explicit math and part-whole relation. They then handcrafted rules to transform a given schema to an operation, such as from "have" and "get" to addition or subtraction. In [Roy and Roth(2017)], they go further to create what they called a unit dependency graph (UDG) to better handle multiplication/division problems. However, all these approaches are label-intensive and may not be suitable once the size of the data grows large.

There are recent approaches that use Q-learning-based and neural machine translation-based models [Huang et al.(2018)Huang, Jing Liu, and Yin]. Their approaches require larger amount of data and more computing power.

3 Baseline Model

3.1 Overview

Suppose our problem consists of a paragraph P, a question Q and in the text there are several quantities $q_1, ..., q_k$. Our baseline model consists of three main parts: A preprocessor extracts essential information from the P, Q and store it into json dictionary format; a feature generator then extracts real-valued features from the preprocessed problem; afterwards, a structural SVM classifier is used to classify the LCA operations between each quantity pairs and generate the expression with the highest score based on its set of LCA operations. At the end, a global inference unit based on beam search used the LCA scores to find the highest-scoring tree.

3.2 Preprocessing

In preprocessing, we extract the relevant information of every quantity. Here we use the concept of quantity schema from [Roy and Roth(2015)]. In a problem P, we extract all the quantities q. For each quantity, we consider the following components:
1. Associated Verb: we detect the verb associated with q by traversing up the dependency tree. The first verb we reach is our target.
2. Subject of Associated Verb: We find the noun phrase, which is subject of the associated verb.
3. Unit: All tokens of the phrase in which the quantity is mentioned are considered as unit tokens. Further, if the parent node of the unit contains words like "each", "of" and "per", the phrase spanned by the parent node will be used as well.
4. Related Noun Phrases: All noun phrases connected to the phrase p containing quantity q will be considered as related noun phrases.
5. Rate: We determine whether quantity q refers to a rate, like '1 dollar per minute'. Then we extract the units 'dollar' and 'minute' to describe the rate. More specifically, we look for words like "per" and "each" in the unit phrase and takes the phrase before or after the word, depending on which side contains more words. The rationale is that phrase like "5 dollars each" often appears at the end of the sentence and phrases "per apple" or "each person" often have the rate unit after the signal word. Since rates sometimes appear in the subject position, such as "each apples cost 5 dollars", we also look for rate in the subject node for each quantity in the quantity schema.

Individual feature	Value range
Verb unigram	integer index
Verb bag-of-word	vector with 0/1 entries
Bag-of-word for comparative adjectives and adverbs	vector with 0/1 entries
Is a rate	0/1
Which of the units in question	integer
Pairwise feature	Value range
whether two quantities share the same mention of verb	0/1
whether two quantities share the same form of verb	0/1
whether two quantities share the same units	0/1
which unit matches in the two quantities if one of them is a rate	integer
whether $q1 > q2$	0/1
whether "more" "less" "than" appears in the question	0/1

Table 1: Features for LCA Classification

3.3 Feature Extraction

The features we used are summarized in Table. (1). Our features fall into two main categories, the individual features and the pairwise features.

For the individual features, we used the associated verb for each quantity to capture the relation between verb category and the math operation meant by the verb.

For the pairwise features, first we compare the main verb associated with each quantity as the main verb may provide useful information about operations like addition. It also helps the solver to understand the priority on which quantity to use, since quantities that with the same verb mention are likely to form a sub-expression in the solution. For example, "John has 3 apples and Mary has 4 apples", the fact that both 3 and 4 associated with the same main verb "has" is correlated with the fact that the operations between the quantities are either addition and subtraction and that 3 and 4 are added first before other quantities. We compared the associated verbs of two quantities as to whether they are: 1. same in verb form; 2. mentioned in the same place in the sentence. The second feature is stricter than the first one.

Similarly, if a quantity is a rate like "5 dollars per person", the chance is high that the multiplication is involved as the operation. And quantities with the same unit are more likely to be multiplied together, like "5 dollars per person" and "10 person". So we have features indicating the correspondence of rate and units between quantities.

3.4 LCA Classification

An expression tree is a tree data structure that stores the quantities as its leaf nodes and the operations between them as the branch nodes. The later the operation is performed, the higher it is in the tree. An example of an expression tree from [Roy and Roth(2015)] is shown in 1. Throughout our model, we used an *expression tree* to represent our solution instead of a raw equation for several reasons. First, equations and arithmetic expressions are inherently hierarchical in their semantics since the order in which the operations are performed between quantities may alter the meaning as well as the final value of the expression. Therefore, evaluating an expression then has the same amount of computation as a level traversal across the expression tree. Further, not only that one expression tree represents one expression, but one expression also has a unique tree representation — under some conditions.

Let the *least-common-ancestor* (LCA) node between two quantity nodes is the node at the lowest level of which the two quantities are the descendants. Another definition required is the *monotonicity* of an expression tree.

Definition 3.1 (monotonic)**.** *A tree is said to be monotonic if any subtraction "-" node is the descendants of any "+" node, and similarly, any division "/" node is the descendants of any "×" node.*

It has been shown in the paper that the set of LCA labels for each quantity pair in the problem in any monotonic expression tree is the same.

Theorem 3.1. *The branch nodes along any path passing the root node for any monotonic expression tree are the same. Further, the leaf nodes along the path are also the same.*

Proof. Proof by induction. When there are only two quantities in the tree, the result holds trivially. Suppose the result holds for any $k < n$, for a tree of n nodes, consider its root node. Suppose there are two monotonic trees T and T' with one having root node '$+$' and the other '$\times$'. Then let the expressions for the two trees represented by their two children be $s_1, s2$ and s_1', s_2' respectively, then $s_1 + s_2 = s_1' s_2'$. However, this will imply that some quantity appears in both side of the tree as the expression of T' will need distributive rule of multiplication to become the expression of T. This leads to contradiction and the result follows.

By monotonicity, we do not need to consider the case when '$-$' is the root node if '$+$' is present in the tree, and by symmetry, we can prove the other case as well. □

Corollary 3.1.1. *The set of LCA labels for the quantities in any monotonic expression tree is unique.*

Proof. From Thm. (3.1), since the path contains the same operation and quantity nodes, in every operation node of the tree, the left branch and right branch of it will contains the same set of quantities and since any pair consisting of one quantity in each branch has the label of the current node as their LCA label, the set of LCA label is unique. For quantities in the branch, we can always find some lower-level node for which they are in different branches of the node. The result then follows. □

The implication of the uniqueness of LCA label is that we can decompose the problem of solving arithmetic problem into classifying LCA operations for each quantity pair. To that end, first we implemented a PCFG parser to parse the raw equations into expression trees. The generative rules are as follows:

$$\begin{aligned} TOP &\to E \\ E &\to E + E \\ E &\to E - E \\ E &\to E * E \\ E &\to E/E \\ E &\to (E) \\ E &\to Q, \end{aligned}$$

where Q is the quantity node, E is an expression and TOP is the root of the tree. Notice that though the grammar for mathematical expressions are not strictly context-free grammar, as the bracket rules will require check of long-term context, it turns out that it is possible to parse the equations using a variant of PCFG where the parser checks three cells in the parsing chart at a time.

We trained a multiclass Support Vector Machine (SVM) with radial-basis-function kernel to classify the LCA operations between each quantity pairs. The log-probability score of the trained classifier on the validation set is then used as the scores for the global inference unit, as will be discussed in the next section.

$$\begin{aligned} C(P, op) = \min_{\mathbf{w},\mathbf{b}} \frac{1}{2}\|\mathbf{w}\|^2 + \lambda \sum_{q_i,q_j \in \mathcal{Q}(P)} (\zeta_{i,j} - \\ (\mathbf{w}\phi(q_i, q_j, o_{LCA}(q_i, q_j)) + \mathbf{b})) \end{aligned} \tag{1}$$

$$v(q_i, q_j, op; P) = (\mathbf{w}\phi(q_i, q_j, o_{LCA}(q_i, q_j)) + \mathbf{b}) \tag{2}$$

$$\approx \log Pr[op|q_i, q_j, P]. \tag{3}$$

We will often shorthand $v(q_i, q_j, op; P)$ as $v(q_i, q_j, op)$ when it does not cause any confusions.

3.5 Expression Tree Generation

Given the scores for each quantity pairs having each LCA operations, the expression tree can be generated by a best-first search algorithm over all candidate expression trees composed by the quantities appeared in the problems. The tree with the maximum score will be chosen. The score we assigned to each of the candidate tree is:

$$V(\mathcal{T}; P) = \sum_{(q_i,q_j,o_{LCA}(q_i,q_j)) \in \mathcal{T}} v(q_i, q_j, o_{LCA}(q_i, q_j)). \tag{4}$$

The score for a partial state S is the sum of the scores of its partial subtrees:

$$V(S; P) = \sum_{\mathcal{T} \in S} V(\mathcal{T}; P) \tag{5}$$

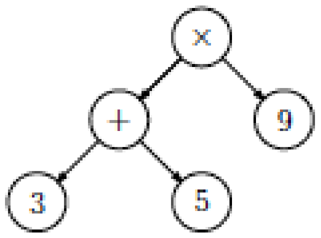

Figure 1: Expression tree example

```
Data: A list of quantities $\mathbf{q}$ with schemas, Dictionary to map quantity pairs
      $(q_i, q_j)$ and each operations to scores $v(q_i, q_j, op; P)$, beam-width $k$
Result: Expression tree $E$ with the highest score $V^*$
k-best-list = [];
while not all states have used all their quantities do
    state = k-best-list.pop() ;
    if state has used all its quantities then
        continue;
    else
        for (node i, node j) in state do
            for each operations op do
                state-next = merge(state, (node i, node j, op));
                state-next.score = updateScore(state.score, (node i, node j, op));
                if checkConstraint(state-next): k-best-list.push(state-next);
            end
        end
    end
end
```

Algorithm 1: BeamSearch($\mathbf{q}$, o_{LCA})

We find the approximately optimal expression tree using beam search with various heuristics. The first heuristic we used is the positive constraint, which dictates that the result of the expression is positive. This constraint can potentially help the system to find better solutions in problems involving subtraction, as word problem rarely gives negative answer in subtraction question; The other is the integer constraint, which limits the result to be integer. The second constraint can be helpful when the question involves division.

The pseudo-code for the beam search is shown in Algorithm. (1).

```
Data: List of quantity pairs $(q_i, q_j)$ and scores for each operations
      $v(q_i, q_j, op|P)$, beam-width $k$
Result: Expression tree $E$ with the highest score $V^*(E|P)$
k-best-list = [];
score = node1.score + node2.score;
for $q_i$ in node1.quantities do
    for $q_j$ in node2.quantities do
        score += $o_{LCA}(q_i, q_j, op)$;
    end
end
return score;
```

Algorithm 2: UpdateScore(score, (node1, node2, op))

4 Experiments

4.1 Experiment Setup

We have tested our LCA solver on two different datasets. The SingleOp dataset contains a list of 562 problems with only a single operation per problem.The MultiArith dataset contains a list of 600 problems with multiple-step expressions. The SingleOp contain irrelevant quantities.

We trained our LCA classifier with cross-validation. For the MultiArith dataset, we followed [Roy and Roth(2015)] and split the data into six folds with one fold as the validation set. Since each fold contains a different type of expression involving a different operation pair, we can test how well our model generalizes across different operations.

To extract the associated verbs and subjects, we employed the Stanford parser and the Stanford dependency parser to parse the problem text on a sentence level. For unit and rate extraction, instead of using a shallow trunk parser as mentioned in [Roy and Roth(2015)], we used the Stanford parser for the purpose. For the individual quantity features, we experimented with various kinds of word representation such as bag-of-word and unigram. No lemmatization is used during the feature extraction since we found it does not have significant effect on the performance of our system.

We used two types of precision metrics to evaluate our system, namely, the strict precision score, which counts the percentage of correctly predicted expression for the problem and the relax precision score, which counts the ratio of correct LCA labels found by the system. We used the precision function from the NLTK to compute our metrics.

	SingleOp	MultiArith
Pair feature only	31.3/32.2	17.6/46.6
Unigram + pair	32.0/33.0	12.7/20.0
BoW + pair	**53.6**/54.5	**25.5**/50.5

Table 2: Summary of Expression precision of LCA solver on the three datasets. Each entry is of the format strict metric/relaxed metric

5 Result Analysis

We trained various models with different combinations of features, and the result is summarized in Table. (2). For the MultiArith dataset, we used both integer and positive constraints; for the other two datasets, only positive constraint is used since the quantities can be fractions in those problems.

Notice that the relaxed and strict score for SingleOp and AddSub are almost the same as the problems only involve single operation, and the discrepancy comes from the fact that our system does not use irrelevance classifier which makes our system predicts extra LCA operations and leads to partially correct predictions.

We found that the individual features with bag-of-words as the word features help improved the performance of our baseline system, which only uses quantity-pairwise features. The integer constraints seem to help the system to narrow down the search space for questions involving division and the positive constraints help the system to narrow down the search space for questions involving subtraction.

However, with unigram feature our model performs worse than without it, which may due to the fact that the integer indices are too big compared to the logical values in the pairwise features and shadows the effect of the latter. It also suggests that pairwise features are easier for our model to learn than with only individual features.

	MultiArith	SingleOp
No heuristics	0	32.0
Positive heuristic	3.3	31.3
Integer heuristic	14.8	-
Integer + Positive heuristic	15.2	-

Table 3: Effect of heuristics across datasets

We found the heuristics such as positive constraint during beam search boosts the performance of our system in MultiArith dataset by 3 % and 15 % respectively. The total contribution is then the sum of the contributions of the two constraints. However, the positive does not seem to help as much in the singleOp dataset, possibly due to the existence of irrelevant data. That is, the result that leads to negative value may contain the right relations between the relevant quantities in the dataset.

	Add Sub	Sub Add	Multiply Add	Add Divide	Multiply Sub	Sub Divide
All pairwise features	71	0	13	5	9	8
No verb features	12	0	4	5	15	12
No unit features	67	0	0	0	0	0
No question features	90	0	15	5	9	8

Table 4: Ablation study of features in MultiArith across problem types. Only pairwise features evaluated with strict precisions are reported.

We also examined the effect of different pairwise features in solving different types of problems, as summarized in Table. (4). The result indicates that verb-related features contribute greatly to recognize addition and subtraction and a 50 % relative reduction in performance in recognizing problems with addition and subtraction. However, it can occasionally mislead the system when solving other types of problems such as problem with multiplication and subtraction and problems with subtraction and division.

The unit features, on the other hand, contribute mainly to the recognition of multiplication and division, as without it, the model fails to solve most problems with multiplication or division involved.

The question feature turns out to be completely useless in solving problems in MultiArith, possibly due to the fact that it involves multiple operations and whether "more" or "less" appears in the question matters little to the overall expression.

6 Error Analysis

There are two main sources of error: The errors in the parsers, which degrade the quality of the quantity schema, and the ambiguity of the features, which leads the model to confuse between LCA labels.

6.1 Parsing Error

The syntactic parser we used occasionally fails to find the quantities. We found that it is because it occasionally confused the numeric tag CD with adjectives tag JJ. Sometimes the syntactic parser also confuses the pronoun "one" as a number. So instead, we have to implement a seperate method to detect quantities. Still, the parser cannot detect quantities in hyphenized adjectives such as "5-day" and "3-dollar".

The dependency parser also failed ocassionally to detect the associated verb of the quantity because the verb such as "was" in "each question was worth 5 points", "worth" was treated as the head verb instead and simply tracing upward the dependency tree from the quantity may not lead to any verb. Also, the verbs are sometimes recognized incorrectly such as in the sentence "each chocalate costs 5 dollars and each cake costs 3 dollars", in which case both of the verbs "costs" are recognized as improper nouns (NN), thus providing no associated verbs for the quantities.

Another issue is with the units. In sentences like "Carol and her mom were picking carrots from their garden. Carol picked 29 and her mother picked 16", the unit "carrots" does not appear in quantities like "16" and "29", so our model chose the unit nearby. However, the choice of whether to choose the unit nearest to the left or right can lead to errors. For example, in the sentence "If she graded 4, but then another 18 were turned in", "another" was recognized as noun and therefore the unit of "18". As a result, "4" then borrowed the unit in 18 and were also assigned the wrong unit.

6.2 Feature Error

The features we defined above for LCA classification can be incorrect, which will lead to an incorrect generation. To analyze this problem, we took a look at the generated features and the gold features. We found out that there are still some expressions not caught with our model. For example, sentence "Luke was trying to expand his game collection. He bought 2 games from a friend and bought 2 more at a garage sale. If 2 of the games didn't work, how many good games did he end up with?" "More" can be easily considered as “add” but in the problem, generated features are incorrect because we cannot capture those information. For example, for sentence like ”Henry had 11 dollars. For his birthday he got 18 more dollars but spent 10 on a new game. How much money does he have now?” Most labels of this sentence are incorrect due to reason mentioned above. For the dataset MultiArith, there are 88 sentences contain “more” and we find out that almost for every sentence of them are labeled incorrectly. Some of the mistakes can be ignored because many sentences in the dataset has the same structure, the only differences between them are the subject and quantity. But the remaining are still to be solved. Although our solver checks comparative adjectives and adverbs, the error may be due to erroneous parsing which mislabels the comparative adjectives.

Also, for some sentences containing ’eaten’ or “sold”, those words refer to specific operation such as addition and subtraction but features of the sentences containing them may not recognize the pattern because it is only provided binary features in our system. Thus, it will fail to recognize words that are similar in meaning but drastically different form. This flaw in our system accounts for the reason why in the cross-validation result of our second fold is so bad, as it seems to contain much richer types of different verbs. For example, in sentences like “Kaleb had 34 books. If he sold 17 of them and used the money he earned to buy 7 new books. ” The verbs “had”and ”buy” both indicate gaining new quantities but the verb-match features will be zero, making it hard to differentiate with the features of other operations.

7 Conclusion

In this project, we built a learning system to automatically solve arithmetic problem of any domain. We then analyzed the effect of each features as well as the common cause of errors made by our system. In the future, we would like to first improve the algorithm for extracting quantity schema; we would like to find a more context-dependent embedding and pairwise features for verbs; then we would like to implement the irrelevance classifier to make our model more robust to irrelevant quantities; we would also like to try the intermediate labeling approach to incorporate more general world knowledge into our solver; in the end, we would like to modify our solver to solve algebraic problems involving unknowns or more advanced math problems with operations beyond the four basic operations. Our code is available at `https://github.com/wangxr14/Algebraic-Word-Problem-Solver`

References

[Hosseini et al.(2014)] Mohammad Javad Hosseini et al. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics. http://www.aclweb.org/anthology/D14-1058.

[Huang et al.(2018)Huang, Jing Liu, and Yin] Danqing Huang, Chin-Yew Lin Jing Liu, and Jian Yin. 2018. Neural math word problem solver with reinforcement learning. In *Proceedings of the 27th International Conference on Computational Linguistics*. Association for Computational Linguistics, page 213–223. http://aclweb.org/anthology/C18-1018.

[Roy and Roth(2015)] Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pages 17–21. http://www.emnlp2015.org/proceedings/EMNLP/pdf/EMNLP202.pdf.

[Roy and Roth(2016)] Subhro Roy and Dan Roth. 2016. Illinois math solver: Math reasoning on the web. In *Proceedings of NAACL-HLT*. Association for Computational Linguistics, pages 52–56. http://www.aclweb.org/anthology/N16-3011.

[Roy and Roth(2017)] Subhro Roy and Dan Roth. 2017. Illinois math solver: Math reasoning on the web. In 31^st *AAAI Conference on Artificial Intelligence*. AAAI.