Solving General Arithmetic Word Problems

Subhro Roy
University of Illinois,
Urbana Champaign
sroy9@illinois.edu

Dan Roth
University of Illinois,
Urbana Champaign
danr@illinois.edu

Abstract

This paper presents a novel approach to automatically solving arithmetic word problems. This is the first algorithmic approach that can handle arithmetic problems with multiple steps and operations, without depending on additional annotations or predefined templates. We develop a theory for expression trees that can be used to represent and evaluate the target arithmetic expressions; we use it to uniquely decompose the target arithmetic problem to multiple classification problems; we then compose an expression tree, combining these with world knowledge through a constrained inference framework. Our classifiers gain from the use of *quantity schemas* that supports better extraction of features. Experimental results show that our method outperforms existing systems, achieving state of the art performance on benchmark datasets of arithmetic word problems.

1 Introduction

In recent years there is growing interest in understanding natural language text for the purpose of answering science related questions from text as well as quantitative problems of various kinds. In this context, understanding and solving arithmetic word problems is of specific interest. Word problems arise naturally when reading the financial section of a newspaper, following election coverage, or when studying elementary school arithmetic word problems. These problems pose an interesting challenge to the NLP community, due to its concise and relatively straightforward text, and seemingly simple semantics. Arithmetic word problems are usually directed towards elementary school students, and can be solved by combining the numbers mentioned in text with basic operations (addition, subtraction, multiplication, division). They are simpler than algebra word problems which require students to identify variables, and form equations with these variables to solve the problem.

Initial methods to address arithmetic word problems have mostly focussed on subsets of problems, restricting the number or the type of operations used (Roy et al., 2015; Hosseini et al., 2014) but could not deal with multi-step arithmetic problems involving all four basic operations. The template based method of (Kushman et al., 2014), on the other hand, can deal with all types of problems, but implicitly assumes that the solution is generated from a set of predefined equation templates.

In this paper, we present a novel approach which can solve a general class of arithmetic problems without predefined equation templates. In particular, it can handle multiple step arithmetic problems as shown in Example 1.

Example 1
Gwen was organizing her book case making sure each of the shelves had exactly 9 *books on it. She has* 2 *types of books - mystery books and picture books. If she had* 3 *shelves of mystery books and* 5 *shelves of picture books, how many books did she have in total?*

The solution involves understanding that the number of shelves needs to be summed up, and that the total number of shelves needs to be multiplied by the number of books each shelf can hold. In addition, one has to understand that the number "2" is not a direct part of the solution of the problem.

While a solution to these problems eventually requires composing multi-step numeric expressions from text, we believe that directly predicting this complex expression from text is not feasible.

At the heart of our technical approach is the novel notion of an *Expression Tree*. We show that the arithmetic expressions we are interested in can always be represented using an Expression Tree that has some unique decomposition properties. This allows us to decompose the problem of mapping the text to the arithmetic expression to a collection of simple prediction problems, each determining the lowest common ancestor operation between a pair of quantities mentioned in the problem. We then formulate the decision problem of composing the final expression tree as a joint inference problem, via an objective function that consists of all these decomposed prediction problems, along with legitimacy and background knowledge constraints.

Learning to generate the simpler decomposed expressions allows us to support generalization across problems types. In particular, our system could solve Example 1 even though it has never seen a problem that requires both addition and multiplication operations.

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1743–1752,
Lisbon, Portugal, 17-21 September 2015.

We also introduce a second concept, that of *quantity schema*, that allows us to focus on the information relevant to each quantity mentioned in the text. We show that features extracted from quantity schemas help reasoning effectively about the solution. Moreover, quantity schemas help identify unnecessary text snippets in the problem text. For instance, in Example 2, the information that "Tom washed cars over the weekend" is irrelevant; he could have performed any activity to earn money. In order to solve the problem, we only need to know that he had $76 last week, and now he has $86.

Example 2
Last week Tom had $74. He washed cars over the weekend and now has $86. How much money did he make from the job?

We combine the classifiers' decisions using a constrained inference framework that allows for incorporating world knowledge as constraints. For example, we deliberatively incorporate the information that, if the problems asks about an "amount", the answer must be positive, and if the question starts with "how many", the answer will most likely be an integer.

Our system is evaluated on two existing datasets of arithmetic word problems, achieving state of the art performance on both. We also create a new dataset of multistep arithmetic problems, and show that our system achieves competitive performance in this challenging evaluation setting.

The next section describes the related work in the area of automated math word problem solving. We then present the theory of expression trees and our decomposition strategy that is based on it. Sec. 4 presents the overall computational approach, including the way we use quantity schemas to learn the mapping from text to expression tree components. Finally, we discuss our experimental study and conclude.

2 Related Work

Previous work in automated arithmetic problem solvers has focussed on a restricted subset of problems. The system described in (Hosseini et al., 2014) handles only addition and subtraction problems, and requires additional annotated data for verb categories. In contrast, our system does not require any additional annotations and can handle a more general category of problems. The approach in (Roy et al., 2015) supports all four basic operations, and uses a pipeline of classifiers to predict different properties of the problem. However, it makes assumptions on the number of quantities mentioned in the problem text, as well as the number of arithmetic steps required to solve the problem. In contrast, our system does not have any such restrictions, effectively handling problems mentioning multiple quantities and requiring multiple steps. Kushman's approach to automatically solving algebra word problems (Kushman et al., 2014) might be the most related to ours. It tries to map numbers from the problem text to predefined equation templates. However, they implicitly assume that similar equation forms have been seen in the training data. In contrast, our system can perform competitively, even when it has never seen similar expressions in training.

There is a recent interest in understanding text for the purpose of solving scientific and quantitative problems of various kinds. Our approach is related to work in understanding and solving elementary school standardized tests (Clark, 2015). The system described in (Berant et al., 2014) attempts to automatically answer biology questions, by extracting the structure of biological processes from text. There has also been efforts to solve geometry questions by jointly understanding diagrams and associated text (Seo et al., 2014). A recent work (Sadeghi et al., 2015) tries to answer science questions by visually verifying relations from images.

Our constrained inference module falls under the general framework of Constrained Conditional Models (CCM) (Chang et al., 2012). In particular, we use the $L + I$ scheme of CCMs, which predicts structured output by independently learning several simple components, combining them at inference time. This has been successfully used to incorporate world knowledge at inference time, as well as getting around the need for large amounts of jointly annotated data for structured prediction (Roth and Yih, 2005; Punyakanok et al., 2005; Punyakanok et al., 2008; Clarke and Lapata, 2006; Barzilay and Lapata, 2006; Roy et al., 2015).

3 Expression Tree and Problem Decomposition

We address the problem of automatically solving arithmetic word problems. The input to our system is the problem text P, which mentions n quantities $q_1, q_2, \ldots, q_n$. Our goal is to map this problem to a read-once arithmetic expression E that, when evaluated, provides the problem's solution. We define a read-once arithmetic expression as one that makes use of each quantity at most once. We say that E is a *valid* expression, if it is such a Read-Once arithmetic expression, and we only consider in this work problems that *can be* solved using valid expressions (it's possible that they can be solved also with invalid expressions).

An expression tree $\mathcal{T}$ for a valid expression E is a binary tree whose leaves represent quantities, and each internal node represents one of the four basic operations. For a non-leaf node n, we represent the operation associated with it as $\odot(n)$, and its left and right child as $lc(n)$ and $rc(n)$ respectively. The numeric value of the quantity associated with a leaf node n is denoted as $\mathrm{Q}(n)$. Each node n also has a value associated with it, represented as $\textsc{Val}(n)$, which can be computed in a

recursive way as follows:

$$\textsc{Val}(n) = \begin{cases} \mathrm{Q}(n) & \text{if } n \text{ is a leaf} \\ \textsc{Val}(lc(n)) \odot(n) \textsc{Val}(rc(n)) & \text{otherwise} \end{cases} \quad (1)$$

For any expression tree $\mathcal{T}$ for expression E with root node n_{root}, the value of $\textsc{Val}(n_{root})$ is exactly equal to the numeric value of the expression E. Therefore, this gives a natural representation of numeric expressions, providing a natural parenthesization of the numeric expression. Fig 1 shows an example of an arithmetic problem with solution expression and an expression tree for the solution expression.

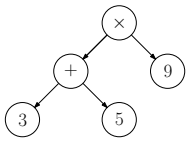
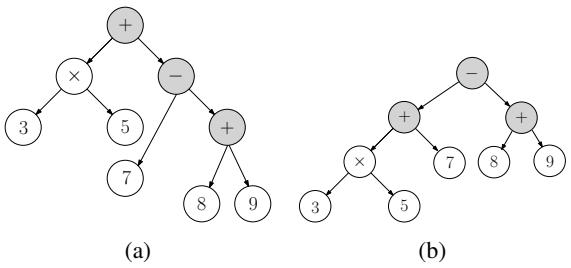
Problem	
Gwen was organizing her book case making sure each of the shelves had exactly 9 *books on it. She has* 2 *types of books - mystery books and picture books. If she had* 3 *shelves of mystery books and* 5 *shelves of picture books, how many books did she have total?*	
Solution	Expression Tree of Solution
$(3 + 5) \times 9 = 72$	

Figure 1: An arithmetic word problem, solution expression and the corresponding expression tree

Definition An expression tree $\mathcal{T}$ for a valid expression E is called ***monotonic*** if it satisfies the following conditions:

1. If an addition node is connected to a subtraction node, then the subtraction node is the parent.
2. If a multiplication node is connected to a division node, then the division node is the parent.

Fig 2 shows two different expression trees for the same expression. Fig 2b is monotonic whereas fig 2a is not.



(a) (b)

Figure 2: Two different expression trees for the numeric expression $(3 \times 5) + 7 - 8 - 9$. The right one is monotonic, whereas the left one is not.

Our decomposition relies on the idea of monotonic expression trees. We try to predict for each pair of quantities q_i, q_j, the operation at the lowest common ancestor (LCA) node of the monotonic expression tree for the solution expression. We also predict for each quantity, whether it is relevant to the solution. Finally, an inference module combines all these predictions.

In the rest of the section, we show that for any pair of quantities q_i, q_j in the solution expression, any monotonic tree for the solution expression has the same LCA operation. Therefore, predicting the LCA operation becomes a multiclass classification problem.

The reason that we consider the monotonic representation of the expression tree is that different trees could otherwise give different LCA operation for a given pair of quantities. For example, in Fig 2, the LCA operation for quantities 5 and 8 can be $+$ or $-$, depending on which tree is considered.

Definition We define an ***addition-subtraction chain*** of an expression tree to be the maximal connected set of nodes labeled with addition or subtraction.

The nodes of an addition-subtraction (AS) chain C represent a set of terms being added or subtracted. These terms are sub-expressions created by subtrees rooted at neighboring nodes of the chain. We call these terms the ***chain terms*** of C, and the whole expression, after node operations have been applied to the chain terms, the ***chain expression*** of C. For example, in fig 2, the shaded nodes form an addition-subtraction chain. The chain expression is $(3\times5)+7-8-9$, and the chain terms are 3×5, 7, 8 and 9. We define a ***multiplication-division (MD) chain*** in a similar way.

Theorem 3.1. *Every valid expression can be represented by a monotonic expression tree.*

Proof. The proof is procedural, that is, we provide a method to convert any expression tree to a monotonic expression tree for the same expression. Consider a non-monotonic expression tree E, and without loss of generality, assume that the first condition for monotonicity is not valid. Therefore, there exists an addition node n_i and a subtraction node n_j, and n_i is the parent of n_j. Consider an addition-subtraction chain C which includes n_i, n_j. We now replace the nodes of C and its subtrees in the following way. We add a single subtraction node n_-. The left subtree of n_- has all the addition chain terms connected by addition nodes, and the right subtree of n_- has all the subtraction chain terms connected by addition nodes. Both subtrees of n_- only require addition nodes, hence monotonicity condition is satisfied. We can construct the monotonic tree in Fig 2b from the non-monotonic tree of Fig 2a using this procedure. The addition chain terms are 3×5 and 7, and the subtraction chain terms are 8 and 9. As as was described above, we introduce the root subtraction node in Fig 2b and attach the addition chain terms to the left and the subtraction chain terms to the right. The same line of reasoning can be used to handle the second condition with multiplication and division replacing addition and subtraction, respectively.

□

Theorem 3.2. *Consider two valid expression trees $\mathcal{T}_1$ and $\mathcal{T}_2$ for the same expression E. Let C_1, C_2 be the chain containing the root nodes of $\mathcal{T}_1$ and T_2 respectively. The chain type (addition-subtraction or multiplication-division) as well as the the set of chain terms of C_1 and C_2 are identical.*

Proof. We first prove that the chains containing the roots are both AS or both MD, and then show that the chain terms are also identical.

We prove by contradiction that the chain type is same. Let C_1's type be "addition-subtraction" and C_2's type be "multiplication-division" (without loss of generality). Since both C_1 and C_2 generate the same expression E, we have that E can be represented as sum (or difference) of two expressions as well as product(or division) of two expressions. Transforming a sum (or difference) of expressions to a product (or division) requires taking common terms from the expressions, which imply that the sum (or difference) had duplicate quantities. The opposite transformation adds same term to various expressions leading to multiple uses of the same quantity. Therefore, this will force at least one of C_1 and C_2 to use the same quantity more than once, violating validity.

We now need to show that individual chain terms are also identical. Without loss of generality, let us assume that both C_1 and C_2 are "addition-subtraction" chains. Suppose the chain terms of C_1 and C_2 are not identical. The chain expression for both the chains will be the same (since they are root chains, the chain expressions has to be the same as E). Let the chain expression for C_1 be $\sum_i t_i - \sum_i t'_i$, where t_i's are the addition chain terms and t'_i are the subtraction chain terms. Similarly, let the chain expression for C_2 be $\sum_i s_i - \sum_i s'_i$. We know that $\sum_i t_i - \sum_i t'_i = \sum_i s_i - \sum_i s'_i$, but the set of t_i's and t'_i's is not the same as the set of s_i and s'_i's. However it should be possible to transform one form to the other using mathematical manipulations. This transformation will involve taking common terms, or multiplying two terms, or both. Following previous explanation, this will force one of the expressions to have duplicate quantities, violating validity. Hence, the chain terms of C_1 and C_2 are identical.

□

Consider an expression tree $\mathcal{T}$ for a valid expression E. For a distinct pair of quantities q_i, q_j participating in expression E, we denote by n_i, n_j the leaves of the expression tree $\mathcal{T}$ representing q_i, q_j, respectively. Let $n_{LCA}(q_i, q_j; \mathcal{T})$ to be the lowest common ancestor node of n_i and n_j. We also define $order(q_i, q_j; \mathcal{T})$ to be true if n_i appears in the left subtree of $n_{LCA}(q_i, q_j; \mathcal{T})$ and n_j appears in the right subtree of $n_{LCA}(q_i, q_j; \mathcal{T})$ and set $order(q_i, q_j; \mathcal{T})$ to false otherwise. Finally we define $\odot_{LCA}(q_i, q_j; \mathcal{T})$ for a pair of quantities q_i, q_j as follows :

$$\odot_{LCA}(q_i, q_j, \mathcal{T}) = \begin{cases} + & \text{if } \odot(n_{LCA}(q_i, q_j; \mathcal{T})) = + \\ \times & \text{if } \odot(n_{LCA}(q_i, q_j; \mathcal{T})) = \times \\ - & \text{if } \odot(n_{LCA}(q_i, q_j; \mathcal{T})) = - \text{ and } order(q_i, q_j; \mathcal{T}) = true \\ -_{reverse} & \text{if } \odot(n_{LCA}(q_i, q_j; \mathcal{T})) = - \text{ and } order(q_i, q_j; \mathcal{T}) = false \\ \div & \text{if } \odot(n_{LCA}(q_i, q_j; \mathcal{T})) = \div \text{ and } order(q_i, q_j; \mathcal{T}) = true \\ \div_{reverse} & \text{if } \odot(n_{LCA}(q_i, q_j; \mathcal{T})) = \div \text{ and } order(q_i, q_j; \mathcal{T}) = false \end{cases} \quad (2)$$

Definition Given two expression trees $\mathcal{T}_1$ and $\mathcal{T}_2$ for the same expression E, $\mathcal{T}_1$ is *LCA-equivalent* to $\mathcal{T}_2$ if for every pair quantities q_i, q_j in the expression E, we have $\odot_{LCA}(q_i, q_j, \mathcal{T}_1) = \odot_{LCA}(q_i, q_j, \mathcal{T}_2)$.

Theorem 3.3. *All monotonic expression trees for an expression are LCA-equivalent to each other.*

Proof. We prove by induction on the number of quantities used in an expression. For all expressions E with 2 quantities, there exists only one monotonic expression tree, and hence, the statement is trivially true. This satisfies our base case.

For the inductive case, we assume that for all expressions with $k < n$ quantities, the theorem is true. Now, we need to prove that any expression with n nodes will also satisfy the property.

Consider a valid (as in all cases) expression E, with monotonic expression trees $\mathcal{T}_1$ and $\mathcal{T}_2$. From theorem 3.2, we know that the chains containing the roots of $\mathcal{T}_1$ and $\mathcal{T}_2$ have identical type and terms. Given two quantities q_i, q_j of E, the lowest common ancestor of both $\mathcal{T}_1$ and $\mathcal{T}_2$ will either both belong to the chain containing the root, or both belong to one of the chain terms. If the LCA node is part of the chain for both $\mathcal{T}_1$ and $\mathcal{T}_2$, monotonic property ensures that the LCA operation will be identical. If the LCA node is part of a chain term (which is an expression tree of size less than n), the property is satisfied by induction hypothesis.

□

The theory just presented suggests that it is possible to uniquely decompose the overall problem to simpler steps and this will be exploited in the next section.

4 Mapping Problems to Expression Trees

Given the uniqueness properties proved in Sec. 3, it is sufficient to identify the operation between any two relevant quantities in the text, in order to determine the unique valid expression. In fact, identifying the operation between *any* pair of quantities provides much needed redundancy given the uncertainty in identifying the operation from text, and we exploit it in our final joint inference.

Consequently, our overall method proceeds as follows: given the problem text P, we detect quantities $q_1, q_2, \ldots, q_n$. We then use two classifiers, one for relevance and other to predict the LCA operations for a monotonic expression tree of the solution. Our training makes use of the notion of quantity schemas, which we describe in Section 4.2. The distributional output of these classifiers is then used in a joint inference procedure that determines the final expression tree.

Our training data consists of problem text paired with a monotonic expression tree for the solution expression. Both the relevance and LCA operation classifiers are trained on gold annotations.

4.1 Global Inference for Expression Trees

In this subsection, we define the scoring functions corresponding to the decomposed problems, and show how we combine these scores to perform global inference. For a problem P with quantities $q_1, q_2, \ldots, q_n$, we define the following scoring functions:

1. $\textsc{Pair}(q_i, q_j, op)$: Scores the likelihood of $\odot_{LCA}(q_i, q_j, \mathcal{T}) = op$, where $\mathcal{T}$ is a monotone expression tree of the solution expression of P. A multiclass classifier trained to predict LCA operations (Section 4.4) can provide these scores.

2. $\textsc{Irr}(q)$: Scores the likelihood of quantity q being an irrelevant quantity in P, that is, q is not used in creating the solution. A binary classifier trained to predict whether a quantity q is relevant or not (Section 4.3), can provide these scores.

For an expression E, let $\mathcal{I}(E)$ be the set of all quantities in P which are not used in expression E. Let $\mathcal{T}$ be a monotonic expression tree for E. We define $\text{Score}(E)$ of an expression E in terms of the above scoring functions and a scaling parameter w_{IRR} as follows:

$$\text{Score}(E) = w_{\text{IRR}} \sum_{q \in \mathcal{I}(E)} \textsc{Irr}(q) + \sum_{q_i, q_j \notin \mathcal{I}(E)} \textsc{Pair}(q_i, q_j, \odot_{LCA}(q_i, q_j, \mathcal{T})) \quad (3)$$

Our final expression tree is an outcome of a constrained optimization process, following (Roth and Yih, 2004; Chang et al., 2012). Our objective function makes use of the scores returned by $\textsc{Irr}(\cdot)$ and $\textsc{Pair}(\cdot)$ to determine the expression tree and is constrained by legitimacy and background knowledge constraints, detailed below.

1. **Positive Answer**: Most arithmetic problems asking for amounts or number of objects usually have a positive number as an answer. Therefore, while searching for the best scoring expression, we reject expressions generating negative answer.

2. **Integral Answer**: Problems with questions such as 'how many" usually expect integral solutions. We only consider integral solutions as legitimate outputs for such problems.

Let $\mathcal{C}$ be the set of valid expressions that can be formed using the quantities in a problem P, and which satisfy the above constraints. The inference algorithm now becomes the following:

$$\arg\max_{E \in \mathcal{C}} \text{Score}(E) \quad (4)$$

The space of possible expressions is large, and we employ a beam search strategy to find the highest scoring constraint satisfying expression (Chang et al., 2012). We construct an expression tree using a bottom up approach, first enumerating all possible sets of irrelevant quantities, and next over all possible expressions, keeping the top k at each step. We give details below.

1. **Enumerating Irrelevant Quantities**: We generate a state for all possible sets of irrelevant quantities, ensuring that there is at least two relevant quantities in each state. We refer to each of the relevant quantities in each state as a term. Therefore, each state can be represented as a set of terms.

2. **Enumerating Expressions**: For generating a next state S' from S, we choose a pair of terms t_i and t_j in S and one of the four basic operations, and form a new term by combining terms t_i and t_j with the operation. Since we do not know which of the possible next states will lead to the optimal goal state, we enumerate all possible next states (that is, enumerate all possible pairs of terms and all possible operations); we prune the beam to keep only the top k candidates. We terminate when all the states in the beam have exactly one term.

Once we have a top k list of candidate expression trees, we choose the highest scoring tree which satisfies the constraints. However, there might not be any tree in the beam which satisfies the constraints, in which case, we choose the top candidate in the beam. We use $k = 200$ in our experiments.

In order to choose the value for the w_{IRR}, we search over the set $\{10^{-6}, 10^{-4}, 10^{-2}, 1, 10^2, 10^4, 10^6\}$, and choose the parameter setting which gives the highest accuracy on the training data.

4.2 Quantity Schema

In order to generalize across problem types as well as over simple manipulations of the text, it is necessary to train our system only with relevant information from the problem text. E.g., for the problem in example 2, we do not want to take decisions based on how Tom earned money. Therefore, there is a need to extract the relevant information from the problem text. To this end, we introduce the concept of a *quantity schema* which we extract for each quantity in the problem's text. Along with the question asked, the quantity

schemas provides all the information needed to solve most arithmetic problems.

A quantity schema for a quantity q in problem P consists of the following components.

1. **Associated Verb** For each quantity q, we detect the verb associated with it. We traverse up the dependency tree starting from the quantity mention, and choose the first verb we reach. We used the easy first dependency parser (Goldberg and Elhadad, 2010).

2. **Subject of Associated Verb** We detect the noun phrase, which acts as subject of the associated verb (if one exists).

3. **Unit** We use a shallow parser to detect the phrase p in which the quantity q is mentioned. All tokens of the phrase (other than the number itself) are considered as unit tokens. Also, if p is followed by the prepositional phrase "of" and a noun phrase (according to the shallow parser annotations), we also consider tokens from this second noun phrase as unit tokens. Finally, if no unit token can be extracted, we assign the unit of the neighboring quantities as the unit of q (following previous work (Hosseini et al., 2014)).

4. **Related Noun Phrases** We consider all noun phrases which are connected to the phrase p containing quantity q, with NP-PP-NP attachment. If only one quantity is mentioned in a sentence, we consider all noun phrases in it as related.

5. **Rate** We determine whether quantity q refers to a *rate* in the text, as well as extract two unit components defining the rate. For example, "7 kilometers per hour" has two components "kilometers" and "hour". Similarly, for sentences describing unit cost like "Each egg costs 2 dollars", "2" is a rate, with units "dollars" and "egg".

In addition to extracting the quantity schemas for each quantity, we extract the surface form text which poses the question. For example, in the question sentence, "How much will John have to pay if he wants to buy 7 oranges?", our extractor outputs "How much will John have to pay" as the question.

4.3 Relevance Classifier

We train a binary SVM classifier to determine, given problem text P and a quantity q in it, whether q is needed in the numeric expression generating the solution. We train on gold annotations and use the score of the classifier as the scoring function $\textsc{Irr}(\cdot)$.

4.3.1 Features

The features are extracted from the quantity schemas and can be broadly categorized into three groups:

1. **Unit features**: Most questions specifically mention the object whose amount needs to be computed, and hence questions provide valuable clue as to which quantities can be irrelevant. We add a feature for whether the unit of quantity q is present in the question tokens. Also, we add a feature based on whether the units of other quantities have better matches with question tokens (based on the number of tokens matched), and one based on the number of quantities which have the maximum number of matches with the question tokens.

2. **Related NP features**: Often units are not enough to differentiate between relevant and irrelevant quantities. Consider the following:

Example 3
Problem : *There are 8 apples in a pile on the desk. Each apple comes in a package of 11. 5 apples are added to the pile. How many apples are there in the pile?*
Solution : $(8 + 5) = 13$

   The relevance decision depends on the noun phrase "the pile", which is absent in the second sentence. We add a feature indicating whether a related noun phrase is present in the question. Also, we add a feature based on whether the related noun phrases of other quantities have better match with the question. Extraction of related noun phrases is described in Section 4.2.

3. **Miscellaneous Features**: When a problem mentions only two quantities, both of them are usually relevant. Hence, we also add a feature based on the number of quantities mentioned in text.

We include pairwise conjunction of the above features.

4.4 LCA Operation Classifier

In order to predict LCA operations, we train a multiclass SVM classifier. Given problem text P and a pair of quantities p_i and p_j, the classifier predicts one of the six labels described in Eq. 2. We consider the confidence scores for each label supplied by the classifier as the scoring function $\textsc{Pair}(\cdot)$.

4.4.1 Features

We use the following categories of features:

1. **Individual Quantity features**: Dependent verbs have been shown to play significant role in solving addition and subtraction problems (Hosseini et al., 2014). Hence, we add the dependent verb of the quantity as a feature. Multiplication and division problems are largely dependent on rates described in text. To capture that, we add a feature based on whether the quantity is a rate, and whether any component of rate unit is present in

the question. In addition to these quantity schema features, we add selected tokens from the neighborhood of the quantity mention. Neighborhood of quantities are often highly informative of LCA operations, for example, "He got 80 more marbles", the term "more" usually indicates addition. We add as features adverbs and comparative adjectives mentioned in a window of size 5 around the quantity mention.

2. **Quantity Pair features**: For a pair (q_i, q_j) we add features to indicate whether they have the same dependent verbs, to indicate whether both dependent verbs refer to the same verb mention, whether the units of q_i and q_j are the same and, if one of them is a rate, which component of the unit matches with the other quantity's unit. Finally, we add a feature indicating whether the value of q_i is greater than the value of q_j.

3. **Question Features**: Finally, we add a few features based on the question asked. In particular, for arithmetic problems where only one operation is needed, the question contains signals for the required operation. Specifically, we add indicator features based on whether the question mentions comparison-related tokens (e.g., "more", "less" or "than"), or whether the question asks for a rate (indicated by tokens such as "each" or "one").

We include pairwise conjunction of the above features. For both classifiers, we use the Illinois-SL package [1] under default settings.

5 Experimental Results

In this section, we evaluate the proposed method on publicly available datasets of arithmetic word problems. We evaluate separately the relevance and LCA operation classifiers, and show the contribution of various features. Lastly, we evaluate the performance of the full system, and quantify the gains achieved by the constraints.

5.1 Datasets

We evaluate our system on three datasets, each of which comprise a different category of arithmetic word problems.

1. **AI2 Dataset**: This is a collection of 395 addition and subtraction problems, released by (Hosseini et al., 2014). They performed a 3-fold cross validation, with every fold containing problems from different sources. This helped them evaluate robustness to domain diversity. We follow the same evaluation setting.

2. **IL Dataset**: This is a collection of arithmetic problems released by (Roy et al., 2015). Each of these problems can be solved by performing one operation. However, there are multiple problems having the same template. To counter this, we perform a few modifications to the dataset. First, for each problem, we replace the numbers and nouns with the part of speech tags, and then we cluster the problems based on unigrams and bigrams from this modified problem text. In particular, we cluster problems together whose unigram-bigram similarity is over 90%. We next prune each cluster to keep at most 5 problems in each cluster. Finally we create the folds ensuring all problems in a cluster are assigned to the same fold, and each fold has similar distribution of all operations. We have a final set of 562 problems, and we use a 5-fold cross validation to evaluate on this dataset.

3. **Commoncore Dataset**: In order to test our system's ability to handle multi-step problems, we create a new dataset of multi-step arithmetic problems. The problems were extracted from *www.commoncoresheets.com*. In total, there were 600 problems, 100 for each of the following types:

   (a) Addition followed by Subtraction
   (b) Subtraction followed by Addition
   (c) Addition and Multiplication
   (d) Addition and Division
   (e) Subtraction and Multiplication
   (f) Subtraction and Division

   This dataset had no irrelevant quantities. Therefore, we did not use the relevance classifier in our evaluations.

   In order to test our system's ability to generalize across problem types, we perform a 6-fold cross validation, with each fold containing all the problems from one of the aforementioned categories. This is a more challenging setting relative to the individual data sets mentioned above, since we are evaluating on multi-step problems, without ever looking at problems which require the same set of operations.

5.2 Relevance Classifier

Table 2 evaluates the performance of the relevance classifier on the AI2 and IL datasets. We report two accuracy values: Relax - fraction of quantities which the classifier got correct, and Strict - fraction of math problems, for which all quantities were correctly classified. We report accuracy using all features and then removing each feature group, one at a time.

[1] http://cogcomp.cs.illinois.edu/page/software_view/Illinois-SL

	AI2		IL		CC	
	Relax	Strict	Relax	Strict	Relax	Strict
All features	**88.7**	**85.1**	**75.7**	**75.7**	60.0	25.8
No Individual Quantity features	73.6	67.6	52.0	52.0	29.2	0.0
No Quantity Pair features	83.2	79.8	63.6	63.6	49.3	16.5
No Question features	86.8	83.9	73.3	73.3	**60.5**	**28.3**

Table 1: Performance of LCA Operation classifier on the datasets AI2, IL and CC.

	AI2		IL	
	Relax	Strict	Relax	Strict
All features	94.7	89.1	**95.4**	**93.2**
No Unit features	88.9	71.5	92.8	91.0
No NP features	**94.9**	**89.6**	95.0	91.2
No Misc. features	92.0	85.9	93.7	89.8

Table 2: Performance of Relevance classifier on the datasets AI2 and IL.

We see that features related to units of quantities play the most significant role in determining relevance of quantities. Also, the related NP features are not helpful for the AI2 dataset.

5.3 LCA Operation Classifier

Table 1 evaluates the performance of the LCA Operation classifier on the AI2, IL and CC datasets. As before, we report two accuracies - Relax - fraction of quantity pairs for which the classifier correctly predicted the LCA operation, and Strict - fraction of math problems, for which all quantity pairs were correctly classified. We report accuracy using all features and then removing each feature group, one at a time.

The strict and relaxed accuracies for IL dataset are identical, since each problem in IL dataset only requires one operation. The features related to individual quantities are most significant; in particular, the accuracy goes to 0.0 in the CC dataset, without using individual quantity features. The question features are not helpful for classification in the CC dataset. This can be attributed to the fact that all problems in CC dataset require multiple operations, and questions in multi-step problems usually do not contain information for each of the required operations.

5.4 Global Inference Module

Table 3 shows the performance of our system in correctly solving arithmetic word problems. We show the impact of various contraints, and also compare against previously best known results on the AI2 and IL datasets. We also show results using each of the two constraints separately, and using no constraints at all.

	AI2	IL	CC
All constraints	72.0	**73.9**	**45.2**
Positive constraint	**78.0**	72.5	36.5
Integral constraint	71.8	73.4	39.0
No constraint	77.7	71.9	29.6
(Hosseini et al., 2014)	77.7	-	-
(Roy et al., 2015)	-	52.7	-
(Kushman et al., 2014)	64.0	73.7	2.3

Table 3: Accuracy in correctly solving arithmetic problems. First four rows represent various configurations of our system. We achieve state of the art results in both AI2 and IL datasets.

The previously known best result in the AI2 dataset is reported in (Hosseini et al., 2014). Since we follow the exact same evaluation settings, our results are directly comparable. We achieve state of the art results, without having access to any additional annotated data, unlike (Hosseini et al., 2014), who use labeled data for verb categorization. For the IL dataset, we acquired the system of (Roy et al., 2015) from the authors, and ran it with the same fold information. We outperform their system by an absolute gain of over 20%. We believe that the improvement was mainly due to the dependence of the system of (Roy et al., 2015) on lexical and neighborhood of quantity features. In contrast, features from quantity schemas help us generalize across problem types. Finally, we also compare against the template based system of (Kushman et al., 2014). (Hosseini et al., 2014) mentions the result of running the system of (Kushman et al., 2014) on AI2 dataset, and we report their result here. For IL and CC datasets, we used the system released by (Kushman et al., 2014).

The integrality constraint is particularly helpful when division is involved, since it can lead to fractional answers. It does not help in case of the AI2 dataset, which involves only addition and subtraction problems. The role of the constraints becomes more significant in case of multi-step problems and, in particular, they contribute an absolute improvement of over 15% over the system without constraints on the CC dataset. The template based system of (Kushman et al., 2014) performs on par with our system on the IL dataset. We believe that it is due to the small number of equation templates in the IL dataset. It performs poorly on the CC dataset, since we evaluate on unseen problem types, which do not ensure that equation templates in the test data will be seen in the training data.

5.5 Discussion

The leading source of errors for the classifiers are erroneous quantity schema extraction and lack of understanding of unknown or rare verbs. For the relevance classifier on the AI2 dataset, 25% of the errors were due to mistakes in extracting the quantity schemas and 20% could be attributed to rare verbs. For the LCA operation classifier on the same dataset, 16% of the errors were due to unknown verbs and 15% were due to mistakes in extracting the schemas. The erroneous extraction of accurate quantity schemas is very significant for the IL dataset, contributing 57% of the errors for the relevance classifier and 39% of the errors for the LCA operation classifier. For the operation classifier on the CC dataset, 8% of the errors were due to verbs and 16% were due to faulty quantity schema extraction. Quantity Schema extraction is challenging due to parsing issues as well as some non-standard rate patterns, and it will be one of the future work targets. For example, in the sentence, "How many 4-dollar toys can he buy?", we fail to extract the rate component of the quantity 4.

6 Conclusion

This paper presents a novel method for understanding and solving a general class of arithmetic word problems. Our approach can solve all problems whose solution can be expressed by a read-once arithmetic expression, where each quantity from the problem text appears at most once in the expression. We develop a novel theoretical framework, centered around the notion of monotone expression trees, and showed how this representation can be used to get a unique decomposition of the problem. This theory naturally leads to a computational solution that we have shown to uniquely determine the solution - determine the arithmetic operation between any two quantities identified in the text. This theory underlies our algorithmic solution - we develop classifiers and a constrained inference approach that exploits redundancy in the information, and show that this yields strong performance on several benchmark collections. In particular, our approach achieves state of the art performance on two publicly available arithmetic problem datasets and can support natural generalizations. Specifically, our approach performs competitively on multistep problems, even when it has never observed the particular problem type before.

Although we develop and use the notion of expression trees in the context of numerical expressions, the concept is more general. In particular, if we allow leaves of expression trees to represent variables, we can express algebraic expressions and equations in this framework. Hence a similar approach can be targeted towards algebra word problems, a direction we wish to investigate in the future.

The datasets used in the paper are available for download at *http://cogcomp.cs.illinois.edu/page/resource_view/98*.

Acknowledgments

This research was sponsored by DARPA (under agreement number FA8750-13-2-0008), and a grant from AI2. Any opinions, findings, conclusions or recommendations are those of the authors and do not necessarily reflect the view of the agencies.

References

R. Barzilay and M. Lapata. 2006. Aggregation via Set Partitioning for Natural Language Generation. In *Human Language Technologies - North American Chapter of the Association for Computational Linguistics*, June.

J. Berant, V. Srikumar, P. Chen, A. V. Linden, B. Harding, B. Huang, P. Clark, and C. D. Manning. 2014. Modeling biological processes for reading comprehension. In *Proceedings of EMNLP*.

M. Chang, L. Ratinov, and D. Roth. 2012. Structured learning with constrained conditional models. *Machine Learning*, 88(3):399–431, 6.

P. Clark. 2015. Elementary School Science and Math Tests as a Driver for AI: Take the Aristo Challenge! In *Proceedings of IAAI*.

J. Clarke and M. Lapata. 2006. Constraint-based sentence compression: An integer programming approach. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 144–151, Sydney, Australia, July. ACL.

Y. Goldberg and M. Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750, Los Angeles, California, June.

M. J. Hosseini, H. Hajishirzi, O. Etzioni, and N. 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 2014, October 25-29, 2014, Doha, Qatar*, pages 523–533.

N. Kushman, L. Zettlemoyer, R. Barzilay, and Y. Artzi. 2014. Learning to automatically solve algebra word problems. In *ACL*, pages 271–281.

V. Punyakanok, D. Roth, and W. Yih. 2005. The necessity of syntactic parsing for semantic role labeling. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1117–1123.

V. Punyakanok, D. Roth, and W. Yih. 2008. The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2).

D. Roth and W. Yih. 2004. A linear programming formulation for global inference in natural language tasks. In Hwee Tou Ng and Ellen Riloff, editors,

Proc. of the Conference on Computational Natural Language Learning (CoNLL), pages 1–8. Association for Computational Linguistics.

D. Roth and W. Yih. 2005. Integer linear programming inference for conditional random fields. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 737–744.

S. Roy, T. Vieira, and D. Roth. 2015. Reasoning about quantities in natural language. *Transactions of the Association for Computational Linguistics*, 3.

F. Sadeghi, S. K. Divvala, and A. Farhadi. 2015. Viske: Visual knowledge extraction and question answering by visual verification of relation phrases. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June.

M. J. Seo, H. Hajishirzi, A. Farhadi, and O. Etzioni. 2014. Diagram understanding in geometry questions. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2831–2838.