Heaven's Light is Our Guide

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Rajshahi University of Engineering & Technology, Bangladesh

Learning to Solve Percentage Word Problems by Parsing into Equations

Author

Habibur Rahman

Roll No. 123044

Department of Computer Science & Engineering

Rajshahi University of Engineering & Technology

Supervised by

Julia Rahman

Assistant Professor

Department of Computer Science & Engineering

Rajshahi University of Engineering & Technology

ACKNOWLEDGEMENT

I would like to express my special appreciation and thanks to my supervisor **Julia Rahman**, Assistant Professor, Department of Computer Science & Engineering, Rajshahi University of Engineering & Technology, you have been a tremendous mentor for me. Again I would like to thank you for encouraging this research. Your advice on both research as well as on my career have been priceless.

I would also like to express my sincere appreciation & deepest sense of gratitude to my honorable teacher **Dr. Md. Rabiul Islam**, Head of the Department of Computer Science & Engineering, Rajshahi University of Engineering & Technology. I also like to thank **Rik Koncel-Kedziorski** for his email support regarding the environment setup. Finally, thanks to all of my honorable teachers, friends & well-wishers for their great role to do complete this research.

Date: 30th October, 2017 Habibur Rahman

RUET, Rajshahi

Heaven's Light is Our Guide

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Rajshahi University of Engineering & Technology, Bangladesh

CERTIFICATE

With immense pleasure, it is hereby certified that the thesis **Learning to Solve Percentage Word Problems by Parsing into Equations**, is prepared by **Habibur Rahman**. Roll. 123044, has been carried out under my supervision. The thesis has been prepared in partial fulfillment of requirements for degree of Bachelor of Science in Computer Science and Engineering.

Supervisor	**External Examiner**
_____	_____
Julia Rahman	**Dr. Z Rahman**
Assistant Professor	Assistant Professor
Department of Computer Science & Engineering	Department of Computer Science & Engineering
Rajshahi University of Engineering & Technology	Rajshahi University of Engineering & Technology
Rajshahi-6204	Rajshahi-6204

Abstract

Math word problems are the mathematical problems that are expressed verbally in natural language. Percentage word problem is one of the math word problems related to percentage problems. Percentage word problems involved in our daily life such as interpreting the financial news, predicting sports results or whether update etc.

Solving Percentage Word Problems is still a big challenge in Natural Language Processing. This paper presents an approach to solve Percentage Word Problem by parsing into equations. In our system, we have used semantic parsing to generate equation trees with the help of **Integer Linear Programming (ILP)**. We have trained a local relationship model based on the operand and operators in the equations and a global model based on equations' correctness. In testing, we have scored the generated equation trees based on a local and global model.

We have conducted experiments on a test set about 200 problems and our system **Percentage Word Problem Solver (PWPS)** is able to solve **69.19%** of the problems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Newspaper articles on stock prices, business news, several advertising on product's current rate, and offers on products of super shops are described in Natural Language. Even Sports commentaries, election results, modern Chat bots for Questions and Answers are also in Natural Language.Computers, since their creation, have exceeded human beings in (speed and accuracy of) mathematical calculation. However, it is still a big challenge nowadays to design algorithms to automatically solve several percentage related problems or context. We need an algorithm to solve those percentage word problems.

1.2 Percentage Word Problem

Math problems described in natural language are called math word problem. Percentage Word Problems are the mathematical problems that are also described in natural language or human like language and it contains the word or mathematics related to "Percentage Problems". Sometimes it contains the system of percentage(%). Figure 1.1 shows an example of Percentage Word Problem.

The height of a mountain on a tropical island changes due to volcanic activity. When the mountain was last measured, its height was 3,750 meters. Now it is 10% taller. How tall is the mountain currently?

Figure 1.1: Example of a Percentage Word Problem

1.3 Background

Automatically solving Math Word Problem is a recently hot topic on understanding Natural Language. However, actual journey of solving was started from the 1960s. Algebraic problems of Natural Language was transformed into kernel sentences and handles to solve the problems in STUDENT [3]. In CARPS [4], they use pattern matching based on expressions by the transformed kernel sentences. However, they were limited to rate based problems. In [5], they first introduced tree-based structure to represent the information in the problem. Recent automatically solving math word problems include number word problems [6], logic puzzle problems [7], geometry word problems [8, 9], arithmetic word problems [10], [11] and algebra word problems [2, 12], [13].

1.3.1 Parsing

Parsing is the process of analyzing a string of symbols, either in natural language or in computer languages, conforming to the rules of a formal grammar. The term parsing comes from Latin pars (orationis), meaning part (of speech).

The term has slightly different meanings in different branches of linguistics and computer science. Traditional sentence parsing is often performed as a method of understanding the exact meaning of a sentence or word, sometimes with the aid of devices such as sentence diagrams. It usually emphasizes the importance of grammatical divisions such as subject and predicate. Parsing are basically two types. One is **syntax analysis** and other is **semantic analysis**.

A **parser** is a software component that takes input data (frequently text) and builds a data structure – often some kind of parse tree, abstract syntax tree or other hierarchical structure – giving a structural representation of the input, checking for correct syntax in the process. The parsing may be preceded or followed by other steps, or these may be combined into a single step. The parser is often preceded by a separate lexical analyser, which creates tokens from the sequence of input characters; alternatively, these can be combined in scannerless parsing. Parsers may be programmed by hand or may be automatically or semi-automatically generated by a parser generator. Parsing is complementary to templating, which produces formatted output. These may be applied to different domains, but often appear together, such as the scanf/printf pair, or the input (front end parsing) and output (back end code generation) stages of a compiler. The overview of the parsing is shown in the figure below:

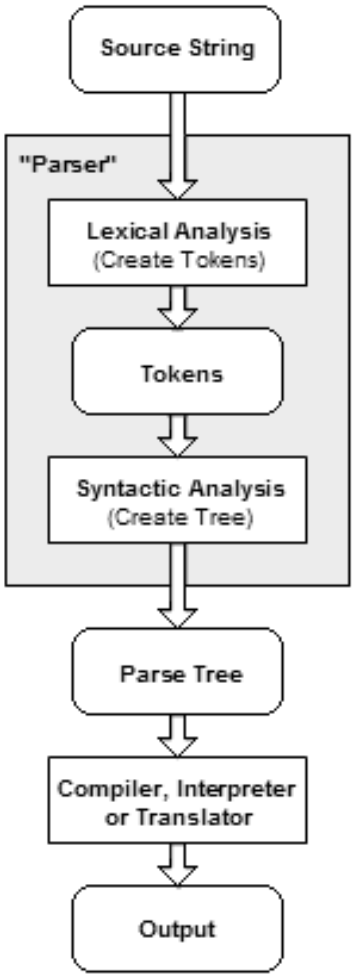

Figure 1.2: Parsing Diagram [1]

1.3.2 Semantic Parsing

Semantic parsing is the process of mapping a natural-language sentence into a formal representation of its meaning. Semantics concerns its meaning: rules that go beyond mere form (e.g., the number of arguments contained in a call to a subroutine matches the number of formal parameters in the subroutine definition – cannot be counted using Context Free Grammar, type consistency):

- Defines what the program means
- Detects if the program is correct
- Helps to translate it into another representation

In semantic parsing, there have been many works. Language grounding for interpretation of a

sentence in world representation has related to many works [14–24]. We discuss three pioneering work closely related to our work.

1.3.3 Verb Categorization Technique

In [10], they tried to solve addition and subtraction problems by verb categories to update a world representation derived from problem text. They ground the problem text to semantic entities and containers. Based on learned verb categories, their system works well for addition and subtraction. They investigates the task of learning to solve such problems by mapping the verbs in the problem text into categories that describe their im- pact on the world state. While the verbs category is crucial, some elements of the problem are irrelevant. For instance, the fact that three kittens have spots is immaterial to the solution.

1.3.4 Template Matching Technique

In [12], they introduce a general method for solving algebra problems. This work can align a word problem to a system of equations with one or two unknowns. They learn a mapping from word problems to equation templates using global and local features from the problem text. However, the large space of equation templates makes it challenging for this model to learn to find the best equation directly, as a sufficiently similar template may not have been observed during training.

1.3.5 Hybrid of Verb Categorization and Template Matching Technique

In ALGES [2], they tried to solve the problem of solving multiple sentenced algebraic word problems by generating and ranking the equation trees. They use a richer semantic representation of the problem text and a bottom-up approach to learning the relations between spans of texts and arithmetic operators. Then score the equations using a global form of the problem to produce the final result. ALGES combined the previous methods to use in broader scope like, Addition, Subtraction, Multiplication and Division for solving single variable problems.

ALGES learns to map spans of text to arithmetic operators, to combine them given the global context of the problem, and to choose the "best" tree corresponding to the problem. The training set for ALGES consists of unannotated algebraic word problems and their solution. Solving the equation represented by such a tree is trivial. ALGES is able to solve word prob-

lems with single-variable equations. In contrast to [10] ALGES covers $+, -, *$, and $/$. The work of [12] has broader scope but we show that it relies heav- ily on overlap between training and test data.

1.4 Objectives

Solving Word Problems requires semantic parsing and reasoning across sentence to find equations. In our system, we have used verb categorization to ground the problem text and divide them entities, containers, and quantities by semantic parsing. After that we have mapped possible equation trees based on those.

In previous, math word problems are tried to solve with verb categorization [10] and template based method [12]. ALGES [2] is a hybrid method which combines both verb categorization and template-based method for solving single variable addition, subtraction, multiplication and division problems.

Our work is related to ALGES, where we converted the percent related number to a fraction and force the problem text to covert it like the problem for ALGES. That is related to using ILP to enforce global constraints in NLP applications [25]. Like previous [26–29], ALGES used ILP to form candidate equations which are then used to generate training data for classification. ALGES attempts to parser re-rank the equations [30, 31].

1.5 Contributions

Our contributions for solving percentage word problems are as follows:

1. We have converted and preprocessed the problem text like, addition, subtraction, multiplication and division problems from the percentage problems;

2. A new scoring equation for generating solutions;

3. A newly build efficient dataset on Percentage Word Problems;

4. Finally, a system Percentage Word Problem Solver named as **PWPS** that can solve 69.19% Percentage Word Problems.

1.6 Organization of Thesis

Chapter 2 is dedicated for the methodology of the system in details. Flowchart, Algorithm are described section by section. In the subsection, Constructing Tree, Generating Equation and Testing and Training methods are described.

Chapter 3 depicts the experimental evaluation of the system. Dataset, experimental setup are described in details in this chapter.

Chapter 4 is dedicated for the result and the performance analysis. It contains Comparison, Ablation Study and Error Analysis.

Chapter 5 represents the summary of this research work and highlights the overall contribution. A light also given on the future scope of math word problem solving.

Appendix A presents the system installation details.

Chapter 2

Methodology

2.1 Introduction

Percentage word problem solver (PWPS) is the system to solve percentage word problems with the help of ALGES which is a step towards solving math word problems automatically. It first converts the problem text to usable in ALGES, generates equations, train local model and global model and finally predict the solution. We discussed the system in details in the following sections.

In our system, we have used ALGES to a broader scope to solve percentage word problems. We have converted the problem text first. Then generate equation trees by semantic parsing and **Integer Linear Programming (ILP)** to solve the problem.

In training, we train local relationship model from the generated equations and their results to set an operator between two operands and a global model based on the equations solution label by correct or incorrect.

In testing, set a score for the generated equations from the local and global model and choose the equation with the highest score as the final equation for a problem text.

2.2 Flowchart of PWPS

Figure 2.1 gives the overview of our proposed system **Percentage Word Problem Solver (PWPS)**. It firstly take the problem text as input that is a string, then passes through the converting portion. After equation generating, there is two-phase. One is training phase, and another is testing phase.

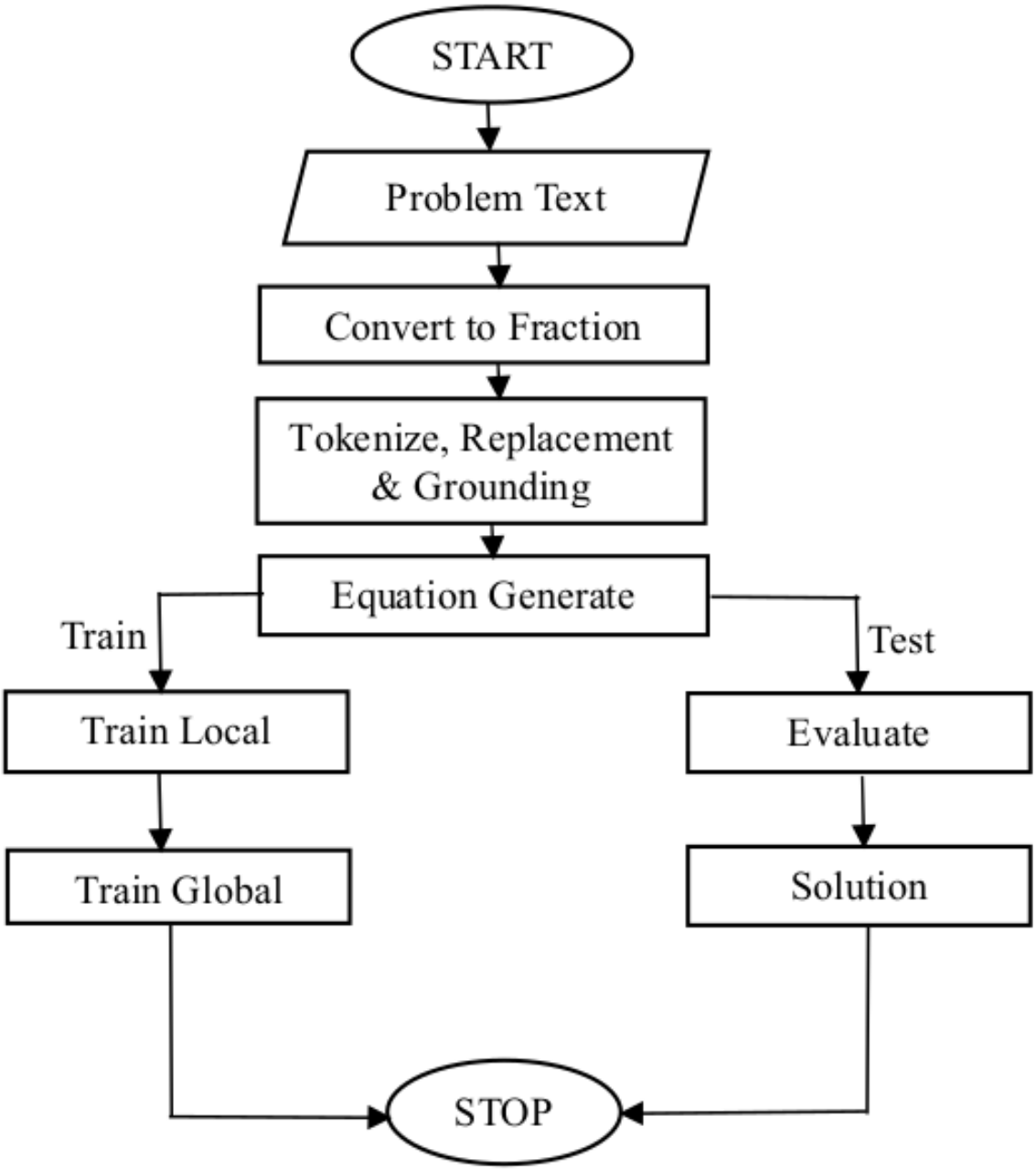

Figure 2.1: Flowchart of PWPS

All the training steps in the algorithm are discussed in this paper as **"Learning"**, and testing steps are discussed as **"Inference"**.

Learning contains train local and global model, and inference contains equation evaluation and solution checking concerning the actual solution of the problem text. Algorithm for PWPS is given in the Fig. 2.2.

2.3 Algorithm of PWPS

Algorithm of Percentage Word Problem Solver (PWPS) consists of two part. One is "Learning" and the other is "Inference".

Learning

1. Make a pair of problem text, p and solution, l
2. Covert to fraction as (1)
3. Tokenize, replace and ground p to base Qsets, S
4. T_i is the generated top M candidate trees by ILP
5. 5. $T_{l_i} \leftarrow$ Select best equation trees based on the solution, l
6. Extract Local Model features for $Qset < s_1, s_2 >$ labeled with op
7. Extract Global Model features by T_i and solution of T_i labeled with positive or negative
8. $L_{local} \leftarrow$ Train Local Model labeled with operator
9. $G_{global} \leftarrow$ Train Global Model labeled with Positive or negative

Inference

1. Step 1 – 7 from Learning
2. Select best equation based on the score from local and global training model like (2)
3. Evaluate the equation
4. Output the solution

Figure 2.2: Algorithm of PWPS

I have discussed the process in details in the following subsections.

2.3.1 Convert to Fraction

To solve percentage problems as addition, subtraction, multiplication or division, we need to convert the percentage related number to fraction with respect to number 100. If the given problem text, p has a number "x%" then, we convert it to "y", where y less than x and $x, y \in \mathbb{R}^+$.

$$y = \frac{x}{100}, where, x > 0 \tag{2.1}$$

2.3.2 Tokenize, Replacement and Grounding

In order to build equation trees from the problem text, P we tokenized the text to words. We changed the symbols of the problem text to corresponding word. $ is changed to $dollar$, % to $times$ where $times$ enforce ALGES to count the statement as a multiplication operation.

Problem text of Fig. 1.1, is converted to fractions and replace the '%' with **times** Fig. 2.3.

The height of a mountain on a tropical island changes due to volcanic activity. When the mountain was last measured, its height was 3,750 meters. Now it is **0.10 times** taller. How tall is the mountain currently?

Figure 2.3: Converted to Fraction and '%' sign replace by 'times' in the problem text

A *Quantified Set* or *Qset* is a node to model problem text quantities and their properties. To generate equation tress, we need to combine the Qsets. A base Qset is a tuple of *ent, qnt, adj, loc, vrb* and *ctr*. The properties are described in the table below:

Table 2.1: The process of forming a single Qset [2]

Item	Properties
qnt	*qnt (Quantity)* is a numerical determiner in the problem text, P
ent	*ent (Entity)* is a noun related to qnt
loc	*loc (Location)* is a noun related to ent
vrb	*vrb (Verb)* is a governing verb
ctr	*ctr (Container)* is the subject of the verb governing

A Qset is ground as a compact representation of the properties based on Table 2.1. Grounded Qsets are two types. One is – **Normal Qset** and **Target Qset**. Target is the Qset where *what*, *how many* or *how much* words or phrases are presents.

Qnt: 930 Ent: Bookmark	Qnt: 0.10 Ent: None	Qnt: x Ent: Bookmark

Figure 2.4: Grounded Qsets from Fig. 2.3

Space of possible equation trees is reduced by reordering the Qsets. ALGES [3] employed three some rules to reorder the Qsets as in TABLE 2.2.

Table 2.2: Rules for reordering Qsets [2]

1. Move *Qset* s_i to immediately after *Qset* s_j if the container of s_i is the entity of s_j and is quantified by *'each'*
2. Move *target Qset* to the front of the list if the question statement includes keywords *start* or *begin*.
3. Move *target Qset* to the end of the list if the problem text includes keyword *left*, *remain*, and *finish*.
4. Move target Qset to the textual location of an intermediate reference with the same *ent* if its *num* property is the determiner *some*.

Reordered Qsets are then combined by some arithmetic operators. If *a* and *b* are two Qsets, then *a* new *Qset c* can be formatted as $c \leftarrow (a, b, op)$, where *op* is the operator.

2.3.3 Generate Equations

ALGES uses **Integer Linear Programming (ILP)** to generate equation trees from the base Qsets. These equations are then used for learning and inferencing the system PWPS and selects best *M* candidate equations for a given problem text, *p*.

For problem text, *p* and *n* base Qsets, PWPS builds *ILP(P)* over the space of postfix equations $E = e_1, e_2, ..., e_L$ of length L and k numeric constants, $k' = n–k$ unknowns, *r* binary operators and *q "types"* of Qsets like ALGES.

In TABLE 2.3, the notations for generating candidate equation trees are given.

Table 2.3: ILP Notation for candidate equations model [2]

INPUT	
p	Problem Text
n	Number of base Qsets
k	Numeric Constant
k'	Number of Unknowns
r	Number of Binary Operators
m	Number of Possible Symbols (n+r)
$type_j$	type of jth base Qsets
M	desired number of candidate equations
L	desired length of postfix notations
OUTPUT	
E	Postfix equation to be generated

In Fig. 2.4, we showed the subset of the candidate equation trees based on the problem text, p in Fig. 1.1, x is the unknown variable where the left one is correct and right one is incorrect parse tree.

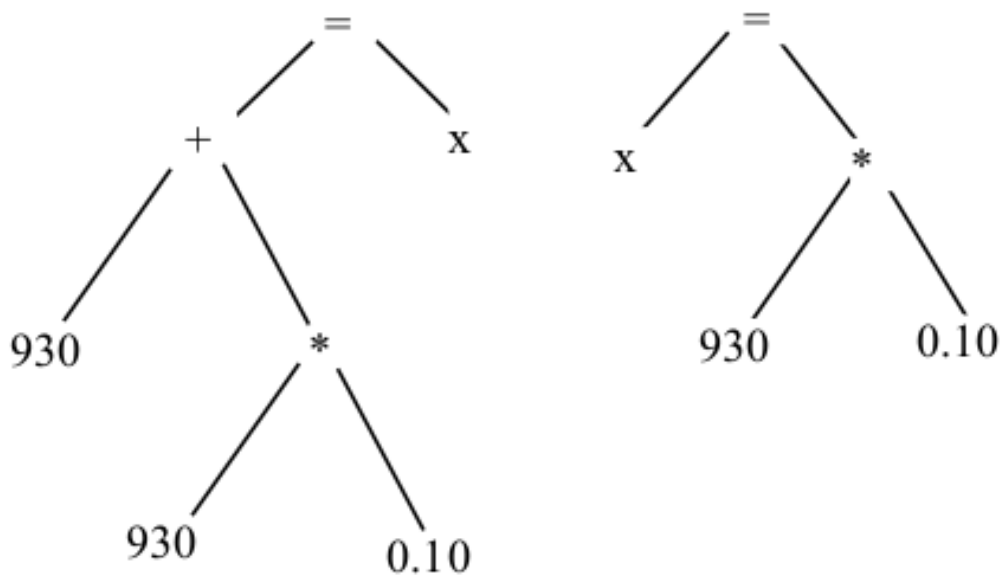

Figure 2.5: Candidate equation tree generated with ILP

2.3.4 Learning

In learning, our system will learn from the score of the equations based on the solution of the problem text, *p* like ALGES. Our dataset contains problem text-solution pairs $(w_i, l_i), where, i = 1, 2, ..., N$. Learning process can be divided into two parts. One is – **Local Qset Relationship Model**, and another is – **Global Equation Model**. Local Model and Global Models are trained based on the problem text, *p* and solution of that problem.

2.3.5 Local Qset Relationship Model

Local Qset Relationship model is learned from the equation tree. For each equation tree, two base Qset s_1 and s_2 are used to extract the features and labeled with op as train data. If $op \in \{+, -, *, /\}$ then, $L_{local} = \theta^T f_{local}(s_1, s_2)$ where f_{local} is the feature vector between the Qsets.

2.3.6 Features in Local Qset Relationship Model

Given the richness of the textual possi- bilities for indicating a math operation, the features are designed over semantic and intertextual relation- ships between Qsets, as well as domain-specific lex- ical features. The feature vector includes three main feature categories (Table 2.4).

First, single set features include syntactic and po- sitional features of individual Qsets. For example, they include indicator features for whether elements of a short lexicon of math-specific terms such as ‘add’ and ‘times’ appear in the vicinity of the set reference in the text. Also, following [10], we include a vector that captures the distance between the verbs associated with each Qset and a small collection of verbs found to be useful in categorizing arithmetic operations in that work, based upon their Lin Similarity [32].

Second, relationships between Qsets are described w.r.t. various Qset properties described in section 4. These include binary features like whether one Qset's container property matches the other Qset's entity (a strong indicator of multiplication), or the distance between the verbs associated with each set based upon their Lin Similarity.

Third, target quantity features check the matching between the target Qset and the current Qset as well as math keywords in the target sentence.

2.3.7 Linear Similarity

Semantic similarity is a metric defined over a set of documents or terms, where the idea of distance between them is based on the likeness of their meaning or semantic content as opposed to similarity which can be estimated regarding their syntactical representation (e.g. their string format). These are mathematical tools used to estimate the strength of the semantic relationship between units of language, concepts or instances, through a numerical description obtained according to the comparison of information supporting their meaning or describing their nature. The term semantic similarity is often confused with semantic relatedness. Semantic relatedness includes any relation between two terms, while semantic similarity only includes “is a” relations. For example, “car” is similar to “bus”, but is also related to ”road” and “driving”.

Our method computes the similarity between two verbs v_1 and v_2 from the similarity between all the senses (from WordNet) of these verbs (Equation 2.2). We compute the similarity between two senses using linear similarity. The similarity between two synsets sv_1 and sv_2 are penalized according to the order of each sense for the corresponding verb. Intuitively, if a synset appears earlier in the set of synsets of a verb, it is more likely to be considered as the correct meaning. Therefore, later occurrences of a synset should result in reduced similarity scores. The similarity between two verbs v_1 and v_2 is the maximum similarity between two synsets of the verbs:

$$sim(v_1, v_2) = \max_{sv:synset(v)} \frac{lin - sim(sv_1, sv_2)}{\log(p_1 + p_2)} \tag{2.2}$$

where sv_1 , sv_2 are two synsets, p_1 , p_2 are the position of each synset match, and $lin - sim$ is the linear similarity.

2.3.8 Global Equation Model

Our system train global equation model to score the equation trees as in ALGES. $G_{global} = \gamma^T f_{global}(p, t)$ where f_{global} is the feature vector capturing the trees, t and the problem text, p. Root node will set based on the local model's prediction of *left* and *right* of the equal operator.

2.3.9 Features in Global Equation Model

Features f global are explained in Table 2.4. They include the number of violated soft constraints in the ILP, the probabilities of the left and right subtrees of the root as provided by the local model, and global lexical features. Additionally, the three local feature sets are applied to the left and right Qsets.

Table 2.4: Features used for Local and Global Model [2]

1. **Single Qset Features (Qset A)**
    - What argument of its governing verb A?
    - Is A a subset of another set?
    - Is A a compound?
    - Math keywords found in the context of A?
    - Verb Lin Distance from known verb categories?
2. **Relational features between Qsets**
    - Entity Match
    - Adjective Overlap
    - Location Match
    - Distance in text
    - Lin Similarity
3. **Target Qset Features**
    - Which one is target Qset?
    - Entity Matched with target entity?
4. **Root Node Features**
    - Number of ILP constraints violated by equation
    - Scores of left and right subtrees of root

2.3.10 Inference

The inference is the testing steps in Fig. 2.1 In inference for a problem text, *P* firstly *ILP(p)* generates the candidate equations. On the candidate equations, the score is calculated from local Qset Relationship model and Global Equation Model. Moreover, the final score for a

candidate equation is calculated through Eq. 2.3. In ablation study ALGES shows that Global Model Score has better impact than Local Relationship Model.

$$p(t|p) = (\alpha * \prod_{t_i \in t} L_{local}(t_i|p)) + (\beta * G_{global}(t|p)) \tag{2.3}$$

Where t_i is the subtree and t are the roots of the equation, α is the bias for **Local Model Score**, β is the bias for the **Global model score** and $\beta = (1 - \alpha)$. Among all the scores, the candidate equation with the highest score is selected for the final equation.

2.4 Conclusion

In PWPS, score is generated from two different model. That are called as score from Local Qset Relation Model score and Global Equation Model Score. And the final decision is made based on these score which is in the inference.

Chapter 3

Experimental Evaluation

3.1 Introduction

The experiments are complicated by the fact that **PWPS** is limited to single equations of percentage word problems, and ALGES can only handle single-equations algebra problem with only addition, subtraction, multiplication and division. Our main experimental result is to find the solution of Percentage word problems.

For experiment our system **PWPS**, there is two parts. One is Experimental Setup and the other is Dataset. In the following section, I have discussed those in details.

3.2 Experimental Setup

We use the Stanford De- pendency Parser in CoreNLP 3.4 [33] to obtain syntactic information used for grounding and feature computation. For the ILP model, we use CPLEX 12.6.1 (IBM ILOG, 2014) [34] to generate the top $M = 100$ equation trees with a maximum stack depth of 10, aborting exploration upon hitting $10K$ feasible solutions or 30 seconds. We use Python's SymPy package for solving equations for the unknown. For the local and global models,we use Random forest classifier [35, 36](Discussed in Appendix B).

3.3 Classifier

3.3.1 Random Forests

Random forests or **random decision forests** are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over-fitting to their training set.

Features of Random Forests

We assume that the user knows about the construction of single classification trees. Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

- It is unexcelled in accuracy among current algorithms.
- It runs efficiently on large data bases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets.
- Generated forests can be saved for future use on other data.
- Prototypes are computed that give information about the relation between the variables and the classification.

- It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data.
- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outliers detection.
- It offers an experimental method for detecting variable interactions.

Algorithm of Random Forest

Decision trees are a popular method for various machine learning tasks. Tree learning "come closest to meeting the requirements for serving as an off-the-shelf procedure for data mining", because it is invariant under scaling and various other transformations of feature values, is robust to inclusion of irrelevant features, and produces inspectable models. However, they are seldom accurate.

In particular, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance.[3]:587–588 This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

Tree bagging

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, ..., x_n$ with responses $Y = y_1, ..., y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For $b = 1, ..., B$:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(x') \tag{3.1}$$

or by taking the majority vote in the case of classification trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on x':

$$\sigma = \sqrt{\frac{\sum_{b=1}^{B}(f_b(x') - \hat{f})^2}{B-1}}. \tag{3.2}$$

The number of samples/trees, B, is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. An optimal number of trees B can be found using cross-validation, or by observing the out-of-bag error: the mean prediction error on each training sample x_i, using only the trees that did not have x_i in their bootstrap sample. The training and test error tend to level off after some number of trees have been fit.

From bagging to random forests

The above procedure describes the original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated.

Typically, for a classification problem with p features, $\sqrt{p}$ (rounded down) features are used in each split. For regression problems the inventors recommend $p/3$ (rounded down) with a minimum node size of 5 as the default.

3.4 Dataset

This work deals with percentage word problems that map to single equations with varying length. Every equation may involve multiple math operations including multiplication, division, subtraction, and addition over non-negative rational numbers and one variable.

We collected a new dataset from `http://math-aids.com`, `http://ixl.com`, `https://www.khanacademy.org` and `http://algebra.com`. Dataset statistics is given below in TABLE 3.1.

Table 3.1: Dataset Statistics

Statistics	#
Number of Problems in Dataset	185
Number of Sentences in Dataset	592
Number of Words in Dataset	5698
Average Sentences per Problem	3.2
Average Words per Problem	30.8

Chapter 4

Result and Performance Analysis

4.1 Result

For calculating the performance of our system, we applied 5-fold cross-validation. For calculating the accuracy of our system, we use Eq. 4.1.

$$Accuracy = \frac{T_c}{T_s} \tag{4.1}$$

Where T_c is the number of problems for which our system **PWPS** gives correct solution and T_s is the number of total problems. Fig. 4.1 shows the accuracy with respect to several values for α and β in Eq.2.3. In horizontal axes the values of α and in vertical axes it shows the accuracy (%).

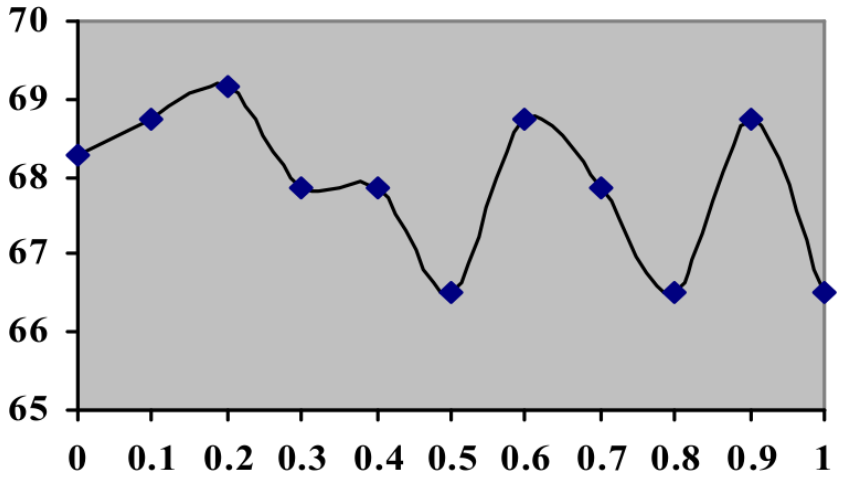

Figure 4.1: Accuracy Versus α

In our dataset, a total number of problems is *185*, and our system could solve problems *128* correctly. Based on this, the accuracy of our system is **69.19%**. Fig. 4.1 show the accuracy for different values of α. In reverse that is also for β. The values in horizontal axes represents the value for α and the values in vertical axes represents the accuracy based on that. We have seen that the accuracy sometimes increases and sometime decreased based on α. We have changed the value of α by *0.1* and for $\alpha = 0.2$ where $\beta = 1–0.2 = 0.8$ is the optimal value for Eq. 2.3 and the accuracy is maximized.

4.1.1 Comparison

From Fig. 4.1, we can see that our system's performance is high for $\alpha = 0.2$ and $\beta = (1–0.2) = 0.8$.

Table 4.1: Comparison Between scoring based on equation of PWPS and ALGES

Equation of System	Accuracy (%)
PWPS	**69.19**
ALGES	67.84

Table 4.1 shows that the accuracy of **PWPS** is **69.19%** which is better than the score using ALGES's equation.

4.1.2 Ablation Study

In order to determine the effect of various components of our system on its overall performance, we perform the following ablations:

No Local Model

We test our system without a local model for generating the equations. Moreover, it is only based on Global Model. So, $\alpha = 0.0$ and $\beta = 1.0$. Without Local Model Eq. 2.3 is like below:

$$p(t|p) = (\beta * G_{global}(t|p)) \tag{4.2}$$

No Global Model

Here, we test our system without the global model for generating the equations which are based on the all local score of the equation. For $\alpha = 1.0$ and $\beta = 0.0$ equation without Global Model 2.3 is like below:

$$p(t|p) = (\alpha * \prod_{t_i \in t} L_{local}(t_i|p)) \tag{4.3}$$

Table 4.2 shows the result of ablation study of our system. Accuracy of PWPS is better than the No Local Model and No Global Model system.

Table 4.2: Ablation Study of PWPS

Method	Accuracy (%)
PWPS	**69.19**
No Local Model	68.28
No Global Model	66.52

Table 4.2 shows the result of ablation study of our system. Accuracy of **PWPS** is better than the No Local Model and No Global Model system.

4.2 Error Analysis

Parsing errors cause a wrong grounding into the designed representation. For example, the parser treats 'regular' as a noun modified by the number '13', leading our system to treat 'regular' as the entity of a Qset rather than 'Coffee'. Despite the improvements that come from PWPS , a portion of errors are attributed to grounding and ordering issues. For instance, the system fails to correctly distinguish between the sets of wheels, and so does not get the movement-triggering container relationships right. Semantic limitations are another source of errors. For example, PWPS does not model the semantics of 'three consecutive numbers'.

Table 4.3: Error in PWPS

Error Type	Problem Text (%)
Parsing Issues	Kira's Cafe has regular coffee and decaffeinated coffee. This morning, the cafe served *13* regular coffees and *39* decaffeinated coffees. What percentage of the coffees served were regular?
Grounding Issues	There are 24 bicycles and 14 tricycles in the storage at Danny's apartment building. Each bicycle has 2 wheels and each tricycle has 3 wheels. What percentage of wheels are there in bicycle?

Finally, **PWPS** is not able to infer quantities when they are not explicitly mentioned in the text.

Chapter 5

Conclusion and Future Works

5.1 Conclusion

We introduced **PWPS**, a new outline method for solving single variable Percentage Word Problems. **PWPS** converts the problem in such a way that can be solve like addition, subtraction, multiplication or division problem. In **PWPS**, we followed the way of generating equation trees using Integer Linear Programming (ILP) and training local and global model like ALGES, but we changed the way of scoring and that gives more accuracy comparing direct uses of ALGES in converted problem text of percentage word problems.

5.2 Future Works

At present, we have focused on single variable percentage word problems. In near feature, we hope to extend our system for multi-variable percentage word problems. The accuracy of **PWPS** can be further improved by optimizing the errors. Moreover, this system can be farther expanded to other domains like physics, chemistry and so on. We hope our dataset will help the researcher to expand math word problem solving. Our code and dataset are publicly available at `https://github.com/habibrahmanbd/PWPS`.

Appendix A

Environment Setup

We have used **Linux (Ubuntu 16.04 LTS)** as our operating system. So, the following process shown only for Linux Environment.

A.1 Java

For installing Java Compiler, the following command should run from the terminal.

```
sudo apt-get install default-jdk
sudo apt-get install default-jre
```

A.2 Python 2 and Pip

We used **python 2** as our programming language. For installing Python 2 and Pip, the following command should run from the terminal.

```
sudo apt-get install python2
sudo apt-get install pip2
```

A.3 Stanford Dependency Parser CoreNLP 3.4 Server

For parser, we have used Stanford Dependency Parser CoreNLP 3.4. This is available at `http://nlp.stanford.edu/software/stanford-corenlp-full-2014-06-16.zip`

A.4 Running Server

Following command should run for installing the server essentials and other packages

```
sudo pip install pexpect unidecode jsonrpclib
git clone https://bitbucket.org/torotoki/corenlp-python.git
cd corenlp-python
wget http://nlp.stanford.edu/software/stanford-corenlp-full
   -2014-08-27.zip
unzip stanford-corenlp-full-2014-08-27.zip
```

Then, to launch a server:

```
python corenlp/corenlp.py
```

Optionally, you can specify a host or port:

```
python corenlp/corenlp.py -H 0.0.0.0 -p 3456
```

That will run a public JSON-RPC server on port 3456. And you can specify Stanford CoreNLP directory:

```
python corenlp/corenlp.py -S stanford-corenlp-full-2014-08-27/
```

A.5 PWPS Running

To get our code run the following command:

```
git clone https://github.com/habibrahmanbd/PWPS
```

After that, for running our system:

```
cd PWPS/
./PWPS problemset.json
```

where, $problemset.json$ is the dataset.

References

[1] Wikipedia, "Parsing," https://en.wikipedia.org/wiki/Parsing.

[2] R. Koncel-Kedziorski, A. S. Hannaneh Hajishirzi, O. Etzioni, and S. D. Ang, "Parsing algebraic word problems into equations," *Empirical Methods in Natural Language Processing*, pp. 1132–1142, 2015.

[3] D. Bobrow, "Natural language input for a computer problem-solving system," *Report MAC-TR-1, Project MAC, MIT, Cambridge*, 1964a.

[4] E. Charniak, "Carps: a program which solves calculus word problems," *Report MAC-TR-51, Project MAC, MIT, Cambridge*, 1968.

[5] C. Liguda and T. Pfeiffer, "Modeling math word problems with augmented semantic networks," *NLDB*, pp. 247–252, 2012.

[6] S. Shi, Y. Wang, C.-Y. Lin, X. Liu, and Y. Rui, "Automatically solving number word problems by semantic parsing and reasoning," *Empirical Methods in Natural Language Processing*, pp. 1132–1142, 2015.

[7] A. Mitra and C. Baral, "Learning to automatically solve logic grid puzzles," *Empirical Methods in Natural Language Processing*, 2015.

[8] M. J. Seo, H. Hajishirzi, A. Farhadi, and O. Etzioni, "Diagram understanding in geometry questions," *AAAI*, 2014.

[9] M. Seo, H. Hajishirzi, A. Farhadi, O. Etzioni, and C. Malcolm, "Solving geometry problems: Combining text and diagram interpretation," *Empirical Methods in Natural Language Processing*, 2015.

[10] M. J. Hosseini, H. Hajishirzi, O. Etzioni, and N. Kushman, "Learning to solve arithmetic word problems with verb categorization," *Empirical Methods in Natural Language Processing*, pp. 523–533, 2014.

[11] S. Roy and D. Roth, "Solving general arithmetic word problems," *Empirical Methods in Natural Language Processing*, 2015.

[12] N. Kushman, Y. Artzi, L. Zettlemoyer, and R. Barzilay, "Learning to automatically solve algebra word problems," *Association for Computational Linguistics*, pp. 271–281, 2014.

[13] L. Zhou, S. Dai, and L. Chen, "Learn to solve algebra word problems using quadratic programming," *Empirical Methods in Natural Language Processing*, 2015.

[14] S. R. K. Branavan, H. Chen, L. S. Zettlemoyer, and R. Barzilay, "Reinforcement learning for mapping instructions to actions," *ACL/AFNLP*, pp. 82–90, 2009.

[15] P. Liang, M. I. Jordan, and D. Klein, "Learning semantic correspondences with less supervision," *ACL/AFNLP*, pp. 91–99, 2009.

[16] D. L. Chen, J. Kim, and R. J. Mooney, "Training a multilingual sportscaster: Using perceptual context to learn language," *JAIR*, pp. 397–435, 2010.

[17] A. Bordes, N. Usunier, and J. Weston, "Label ranking under ambiguous supervision for learning semantic correspondences," *ICML*, pp. 103–110, 2010.

[18] Y. Feng and M. Lapata, "How many words is a picture worth? automatic caption generation for news images," *ACL*, pp. 1239–1249, 2010.

[19] H. Hajishirzi, M. Rastegari, A. Farhadi, and J. K. Hodgins, "Semantic understanding of professional soccer commentaries," *Uncertainty in Artificial Intelligence*, 2012.

[20] H. Hajishirzi, J. Hockenmaier, E. T. Mueller, and E. Amir, "Reasoning about robocup soccer narratives," *Uncertainty in Artificial Intelligence*, pp. 291–300, 2011.

[21] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox, "Learning to parse natural language commands to a robot control system," *International Symposium on Experimental Robotics (ISER)*, 2012.

[22] Y. Artzi and L. Zettlemoyer, "Weakly supervised learning of semantic parsers for mapping instructions to actions," *TACL*, pp. 49–62, 2013.

[23] M. Yatskar, L. Vanderwende, and L. Zettlemoyer, "See no evil, say no evil: Description generation from densely labeled images," *Lexical and Computational Semantics*, p. 110, 2014.

[24] B. Hixon, P. Clark, and H. Hajishirzi, "Learning knowledge graphs for question answering through conversational dialog," *North American Chapter of the Association for Computational Linguistics*, 2015.

[25] D. Roth and W. tau Yih, "A linear programming formulation for global inference in natural language tasks," *Association for Computational Linguistics*, pp. 1–8, 2004.

[26] V. Srikumar and D. Roth, "A joint model for extended semantic role labeling," *EMNLP*, 2011.

[27] D. Goldwasser and D. Roth, "Learning from natural instructions," *IJCAI*, 2011.

[28] J. Berant, V. Srikumar, P.-C. Chen, A. V. Linden, B. Harding, B. Huang, P. Clark, and C. D. Manning, "Modeling biological processes for reading comprehension," *EMNLP*, 2014.

[29] F. Liu, J. Flanigan, S. Thomson, N. Sadeh, and N. A. Smith, "Toward abstractive summarization using semantic representations," *North American Chapter of the Association for Computational Linguistics*, 2015.

[30] M. Collins, "Discriminative re-ranking for natural language parsing," *Computational Linguistics*, pp. 25–70, 2005.

[31] R. Ge and R. J. Mooney, "Discriminative re-ranking for semantic parsing," *Association for Computational Linguistics*, 2006.

[32] D. Lin, "An information-theoretic definition of similarity," *ICML*, pp. 296–304, 1998.

[33] M.-C. D. Marneffe, B. MacCartney, and C. D. Manning, "Generating typed dependency parses from phrase structure parses," *LREC*, pp. 449–454, 2006.

[34] I. I. C. O. S. 12.6.1, "Ibm ilog," 2014.

[35] H. T. Kam, “Random decision forests,” *International Conference on Document Analysis and Recognition*, pp. 278–282, 1995.

[36] H. Kam, “The random subspace method for constructing decision forests,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 832–844, 1998.