Proceedings of the IASTED International Conference
Artificial Intelligence and Applications (AIA 2013)
February 11 - 13, 2013 Innsbruck, Austria

TEXT-TO-DIAGRAM CONVERSION: A METHOD FOR FORMAL REPRESENTATION OF NATURAL LANGUAGE GEOMETRY PROBLEMS

Anirban Mukherjee
RCCIIT, Kolkata
India
anirbanm.rcciit@gmail.com

Sarbartha Sengupta
IIT, Bombay
India
sarbartha@cse.iitb.ac.in

Dipanjan Chakraborty
IIT, Delhi
India
dipanjan@cse.iitd.ac.in

Anirban Sen
BESUS, Howrah
India
anirbansen43@yahoo.com

Utpal Garain
ISI, Kolkata
India
utpal@isical.ac.in

ABSTRACT
Natural language geometry problems are translated into formal representation. This is done as an essential step involved in text to diagram conversion. A parser is designed that analyzes a problem statement in order to describe it as a language independent, unambiguous formal representation. Natural language processing tools and a lexical knowledge base are used to assist the parser that finally generates a graph as the parsing output. The parse graph is the formal representation of the input natural language problem. This graph is later translated into another intermediate representation consisting of a set of graphics-friendly statements. High school level geometry problems are used to develop and test the proposed methods. Experimental results show high accuracy of the approach in translating a natural language problem into a formal description.

KEY WORDS
Text to Diagram conversion, Natural language problems, Formal description, Parsing algorithm, Parse graph, Complexity analysis, Knowledgebase.

1. Introduction

Representing text by a diagram is a typical requirement in many branches of science and engineering like physics, geometry, engineering drawing, etc. In these subjects, often, a piece of text describes certain figures or diagrams and the aim is to draw the diagrams from the text. In doing so, the first step is to understand the input text properly so that information required to draw the underlying diagram can be gathered by analyzing the text. There were many earlier attempts to involve machines to understand such text for solving different types of problems. The studies described in [7], [16] and [9] are examples in this direction. A review paper [1] provides a state of the art survey of the relevant studies made till 2007. However, researchers are still continuing to contribute in this field [12].

Drawing the diagrams as described by geometry word problems is a fascinating example in the area of text to diagram conversion problems. A framework to generate diagrams from geometrical text was described in a thought paper [2]. The proposed system consists of several modules as depicted in the schematic diagram of in Fig.1. The problem statement is fed into Parser 1 which converts the natural language description of the problem into a formal representation in the form of a parse graph. In doing this, the system makes use of NLP tools (mainly parts-of-speech tagging) and a geometry knowledge base. The knowledge base serves as a lexical resource to the parsing module.

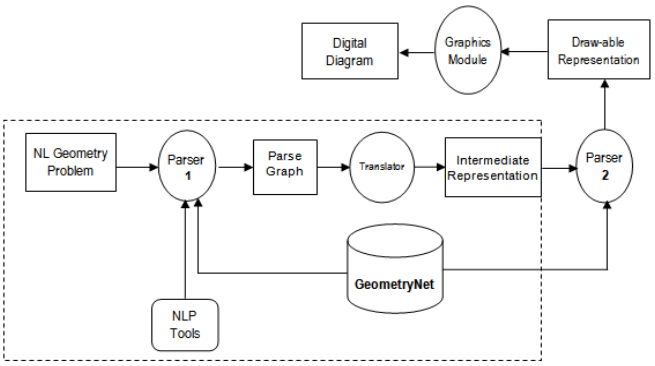

Fig.1 Modules of the Text-to-Diagram Conversion System

Such an idea of using a lexical resource in representing knowledge has been exploited in many other contexts. For example, the paper [11] describes the role of a knowledge base in designing a natural language understanding model. In the context of query answering, the paper [14] examines the task of identifying a knowledge base and using it in answering questions on a wide variety of topics.

In the context of converting natural language statements into formal representation, it is relevant to mention some studies [6], [10] in the field of Software Engineering (SE) modeling where formal models have been generated from informal textual description. There the goal is to analyze natural language specification to discover a underlying software model (e.g., object oriented model). The study in [15] attempts a similar task of mapping sentences to hierarchical representations of their underlying meaning. For this purpose, the authors presented an algorithm for learning a generative model of natural language sentences together with their formal meaning representations with hierarchical structures. Later on, the authors in [4] attempted to translate natural language sentences to formulae in a formal or a knowledge representation language. They use a syntactic combinatorial categorical parser to parse natural language sentences and also to construct the semantic meaning of the sentences as directed by their parsing. In our study, we follow a graph theoretic parsing approach that analyzes the natural language statement and generates two intermediate tables. These tables are finally converted into a parse graph. The idea is to extract knowledge from text and

DOI: 10.2316/P.2013.793-042

represent the knowledge in a graphical form. In the context of SE modeling, converting textual description into graphs has been attempted before [13]. However, the idea of involving a parsing module has not been introduced there. Moreover, the method has not been thoroughly evaluated and therefore, true potential of the approach cannot be readily understood. This paper attempts to bridge the gap.

Once the first level parser provides a formal description of the input statement, the parse graph in our approach, goes through a Translator which generates a graphics-friendly summary of the parse graph. This summary is termed as intermediate representation which can be considered another language independent, unique, unambiguous representation of the input problem statement. A second parser, i.e. Parser 2 acts on the intermediate formal representation of the problem statement, aided by a geometry knowledge base, to generate a set of arguments of graphical functions. A graphical module can then generate the required diagram using the graphic functions.

As the formal representation of the input problem largely dictates the success of the overall system, the first level parser, i.e. Parser 1 is considered as the most important part of the system. This paper, deals precisely with this module. This paper describes the design methodology of Parser 1 and the associated Translator. The input to the parser is a school-level geometry problem in English language text and the output is a language-free structured representation of the problem statement. This representation is free from ambiguities that normally appear in a natural language text. Therefore, the salient contributions of this paper are:

1) It describes a graph-based novel approach to build an unambiguous language-free intermediate representation from school level geometry word problems stated in a natural language.

2) The intermediate representation can be used in many applications, for example, drawing of the underlying diagram, language translation, problem solving, etc.

3) The parser's and the translator's accuracies are tested on an exhaustive test-set generated from a school level text books. The failure cases point to the future research needed to improve the parsing method.

The rest of the paper is organized as follows. Section 2 of the paper describes our approach; Section 2.2 describes the parsing approach with reference to the knowledge base (Section 2.1) while Section 2.3 presents the Translator. The utility of the intermediate representation is also highlighted in the context of the text to diagram conversion problem. Section 3 describes the evaluation strategy. Finally, Section 4 concludes the paper highlighting the future scope of the work and ways to overcome probable shortcomings.

2. Our approach

Given a geometry problem statement in natural language, the goal of this study is to understand the knowledge conveyed by the text and translate it into a formal representation. This goal is achieved by a parser and a translator using the geometry knowledge base, namely GeometryNet [3] and NLP tool, namely a parts-of-speech tagger. The parsing algorithm makes use of two intermediate tables in order to construct a graph as the parsing output. The translator works on this parse graph and generates a summary of the problem statement. The different steps of parsing and translation are described below following a brief introduction to GeometryNet.

2.1 The Knowledge Base

The GeometryNet [3] knowledge base has been built borrowing major ideas of the WordNet [5], a popular lexical database often used for natural language processing tasks. GeometryNet lists 51 geometric entities (e.g. line, parallelogram, circle, triangle, quadrilateral etc.) and entity parameters (e.g. coordinates, angle, slope, length etc.), 50 entity attributes (e.g. isosceles, concentric, common etc.) and 35 interaction types or relations (e.g. produce, intersect, bisect etc.) between the entities. From implementation point of view GeometryNet is a text file where the semantics of all the above terms are expressed in terms of equations involving their arguments. For example, the information that is stored in GeometryNet about the parallelogram entity is:

```
parallelogram
n=4
vertex_1
vertex_2
vertex_3
vertex_4
side_1
side_2
side_3
side_4
#((x2-x1)^2+(y2-y1)^2)^0.5
=((x4-x3)^2+(y4-y3)^2)^0.5
#((x3-x2)^2+(y3-y2)^2)^0.5
=((x4-x1)^2+(y4-y1)^2)^0.5
#m1=m3
#m2=m4
end
```

Here the Net defines that a parallelogram has four sides: side_1, side_2, side_3, side_4 and four vertices: vertex_1, vertex_2, vertex_3, vertex_4. The slope of a line is denoted by m. The parameter conditions of parallelogram are the four mathematical relations prefixed with #. The first two # relations depict opposite sides of a parallelogram are equal in length. The other two # relations denote the slopes of the opposite sides are same i.e. they are parallel. The generic information about vertex and side are derived from the following entries in the Net.

```
vertex  point  side  line      m
point   (x,y)  line  point_1   #(y2-y1)/(x2-
end     end    end   point_2   x1)
                     m         end
                     end
```

The attributes and relations are also represented in a similar way.

2.2 The Parsing Approach

The overall parsing algorithm can be decomposed into the following five steps.

• **Step 1**: POS (parts-of-speech) tag the input statement and break it into lines.

• **Step 2:** Build an entity table listing all the proper nouns (NNPs), the corresponding common nouns (NNs) and the attributes of the NNPs (if any).

• **Step 3:** Use GeometryNet to complete the entity table if remained incomplete in Step 2.

• **Step 4:** Construct a parse graph from the entity table. The NNPs are represented by nodes and are decomposed into basic entities in constructing the graph.

• **Step 5:** Find connectors to establish relationship between a pair of entities and complete the parse graph built in Step 4.

2.2.1 Parsing Step 1

The input problem statement is first tagged. The Brill POS tagger [8] is used for this purpose. We will be using the tagged problem in Table I as an example throughout the description of our approach.

Table I
The problem statement after pos tagging

X(NNP) is(VBZ) the(DT) midpoint(NN) of(IN) side(NN) BC(NNP) of(IN) parallelogram(NN) ABCD(NNP).(.)AX(NNP) when(IN) produced(VBN) meets(VBZ) CD(NNP) at(IN) Q(NNP).(.) CQ(NNP) is(VBZ) extended(VBN) to(IN) P(NNP) and(CC) ABPQ(NNP) is(VBZ) a(DT) parallelogram(NN).(.) Prove(VB) that(IN) CD(NNP)=(SYM) CQ(NNP)=(SYM) PQ(NNP).(.)

Next the tagged statement is broken into lines as shown in Table II. The lines are found following a rule based approach where a group of words that ends with a comma(,), a semicolon(;), the word 'and' or a period(.) is labeled as a line. In Table II, the third line in the problem statement is broken into two as an 'and' usually forms a connection between two complete sentences. Several cases arise when connectors like 'and' or ',' appear in a problem statement:

Table II
The problem statement broken into lines

X(NNP) is(VBZ) the(DT) midpoint(NN) of(IN) side(NN) BC(NNP) of(IN) parallelogram(NN) ABCD(NNP) .(.)
AX(NNP) when(IN) produced(VBN) meets(VBZ) CD(NNP) at(IN) Q(NNP) .(.)
CQ(NNP) is(VBZ) extended(VBN) to(IN) P(NNP) and(CC)
ABPQ(NNP) is(VBZ) a(DT) parallelogram(NN) .(.)
Prove(VB) that(IN) CD(NNP) =(SYM) CQ(NNP) =(SYM) PQ(NNP) .(.)

• If an 'and' or a ',' separates two complete sentences: As mentioned in the example in Table II, in such a situation the line will be broken into two.

• If an 'and' or a ',' separates two or more proper nouns (NNPs): For example, '`X, Y, and Z are the midpoints of sides AB, BC, and AC of a triangle respectively`'. Here, the 'and's or ','s encountered do not join two different sentences as in the example in Table II; rather, they separate more than one NNPs of the same type. If such cases appear in a problem statement, the division discussed previously will not take place.

• If an 'and' or a ',' separates two different types of proper nouns (like a line and an angle): In this case the division will take place. For example, '`X is the midpoint of BC, Y is a point on AB such that . .`'. Here, the ',' lies between two different types of the NNPs. In such cases the division will occur as usual and will not be ignored as discussed above.

• If an 'and' or a ',' separates two equations: They will be divided into two lines. For example,

'`AB = BC and CD = DF`' will call for a division at 'and'.

2.2.2 Parsing Step 2

The next step is to build a table listing all the Proper Nouns (NNPs), the corresponding common nouns (NNs) and the attributes of the NNPs (if any) as featured in the problem. In the problem statement stated in Table I, X, BC, ABCD, AX, Q, CD, CQ, P, and ABPQ are NNPs and parallelogram, side, and midpoint are the only NNs mentioned in the problem. From this information, we can construct the table partially as in Table III.

Table III
Partial entity table

X	midpoint	none
BC	line	none
ABCD	parallelogram	none
AX	–	none
CD	–	none
Q	–	none
CQ	–	none
P	–	none
ABPQ	parallelogram	none

The dashes (-) represent the NNPs whose corresponding NNs are not mentioned directly in the problem. The third column stores the special features associated with a particular entity, if mentioned in the problem statement. For example, isosceles, equilateral are special types of triangles which build on the features of a triangle.

2.2.3 Parsing Step 3

The main task at this step is to recognize the types of the remaining NNPs (other than X, BC, ABCD, and ABPQ in

the above problem) whose corresponding common nouns are not mentioned explicitly in the problem statement. Here we use the GeometryNet [3] that stores all possible geometric entities and the number of points (that are used to represent them) against each of them as in Table IV.

Table IV
Table containing entity names and number of points required to define it

parallelogram	4
square	4
rectangle	4
triangle	3
arc	3
..	..
..	..

The method counts the number of points in the proper noun (ABCD for example has 4 points) whose NN field is blank in Table III. This shall be filled up with the required NN from GeometryNet (by comparing the number of points of the corresponding NNP and Table IV data in the GeometryNet). For this, we always consider the first entry1 in the knowledge-base whose number of points match the number of points in the NNP of the problem. This works because if a specialized entity were to occur in a problem, the type would invariably be mentioned, e.g. if 'AB' is mentioned in a problem, it can easily be inferred that it is a line segment, but in case of ABCD it has to be mentioned whether it is a square or a parallelogram or something else. As a result, for the problem in Table I, the completed table will be as shown in Table V.

Table V
Complete entity table

X	midpoint	none
BC	line	none
ABCD	parallelogram	none
AX	line	none
CD	line	none
Q	point	none
CQ	line	none
P	point	none
ABPQ	parallelogram	none

Sometimes we may encounter NNs without NNPs associated with them in the problem statement. For example consider, 'bisector of angle ABC meets PQ at O'. For this statement, the initial entity table using the method described before will be as in Table VI.

Table VI
Example of an NNP-less table

–	bisector	none
ABC	angle	none
PQ	line	none
O	point	none

Note that the NNP for bisector is blank in table VI. For these cases we assign default values. The method checks which letters (of the English alphabet) are not used in the problem for a vertex of an entity and then creates a set of those unused letters. Next, the GeometryNet [3] is consulted to know the number of points required to construct the NN. The information stored in the GeometryNet against the entity 'bisector' tells that it is a line and hence represented by two points. So our method assigns DE against it (since D and E are not used in the problem statement). The resulting table becomes one as shown in Table VII.

Table VII
Default names set for NNs without a name

DE	bisector	none
ABC	angle	none
PQ	line	none
O	point	none

If in a problem, an NN is a plural (e.g., 'bisectors of angle ABC and angle BC A meet at O . . .'), then we must determine how many NNPs are being indicated by that plural NN. The plural NN is then replaced by its singular type and the required numbers of duplicates are created for that NN. Default values are assigned for all the duplicates. For example, the processing of the statement, 'the bisectors of the angles ABC, BCA and CAB meet at O', will assign three default names DE, FG, and HI for the three bisectors although only the word bisectors has been mentioned in the problem statement.

2.2.4 Parsing Step 4

This step generates a graph from the entity table. At the initial stage the graph contains all the NNPs as listed in the entity table along with their corresponding NNs and special attributes (if any). At the same time, all entities are decomposed into the most atomic geometric entities which can completely define the entity (like sides and vertices for a polygon). Therefore, all polygons are decomposed into their sides and the sides into the corresponding vertices. All these entities (those explicitly mentioned in the problem and those derived after decomposing them) form the nodes of the graph. During decomposition, it is ensured that an entity is not repeated. For example line AB can be derived from both parallelogram ABCD and triangle ABC occurring in a single problem statement. However, only the first derivation is retained and the others are rejected. Now, there can be ambiguities if both line AB and line BA, or parallelogram ABCD and parallelogram CDAB are present in the graph. In such cases too repetitions are detected and rejected. This is achieved by reducing each entity to its canonical form. The canonical form of each entity name is derived by sorting the name on ASCII value. For example, the polygon, if named DABC, is renamed to ABCD as the canonical name. It must be noted that canonicalization is not done for entities like angles and arcs as the order of vertices is important for these so that ABC and BAC are not essentially the same.

To connect the nodes in the graph, parent-child relationships are kept track of during decomposition. Each entity derived from another entity is a child of that entity and they are connected in the graph. The edge itself is, however, non-directional. For the problem statement in Table I the entities after canonicalization and removal of repetitions are: X BC B C ABCD AB CD AD A D AX Q CQ P ABPQ BP PQ AQ. The graph formed from the problem statement in Table I until now is as shown in Fig. 2. Note that only the names of the entities are shown and the attributes are omitted to avoid cluttering.

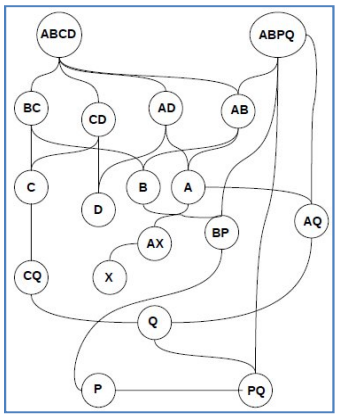

Fig. 2 Parse graph produced for the problem in Table I

Although in this case the graph is connected, there may appear certain cases in problem statements where the graph formed until this step is disconnected. Moreover, the graph formed until this step does not contain any information about the relationships between the entities. For instance, X is the midpoint of BC cannot be inferred from this graph. Therefore, this much information may not lead to the correct generation of the diagrams. This leads to the formulation of the next step.

2.2.5 Parsing Step 5

This is the final step of the parsing method. The goal of this step is to complete the remaining task, i.e. to establish relationships between isolated sub-graphs so that a single connected graph with complete information about the problem statement is obtained. In general geometry word problems, the usual terms or phrases that establish a relation between two entities are commonly *of, in, produced, has, extended, meets*, etc. Our design is flexible enough to add more such terms as and when they appear. We call such terms connectors, in general. The detection of connectors in our approach is a rule based one. The parts of speech tags against the words in the problem statement are used. The terms with tags NNP, IN, VBZ, VBN and every tag that connects two NNPs in some way are marked. Some verbs (with tag VBZ) that do not contain any substantial information are treated as stop words and are ignored. Examples of such verbs are: *is, was, prove, find, show, calculate, completed* etc.

At this stage, another table named as entity-connector table is built. Each row of the table has the NNPs (along with their types) and connectors of the corresponding line in the problem (in the same sequence that they appear in the problem). Note that lines have been identified at Step 1. Explicit rules for classifying connectors and retaining them in the current table are:

• All NNPs are retained in the table.

• All NNs corresponding to each of the proper nouns are retained.

• Any verb which is encountered during the course of parsing a problem statement is retained. Such words (which act as verbs), usually have a tag VBZ or VBN in the problem statement.

• Any mathematical symbol specified in the problem statement is retained. These usually have a SYM tag against them.

• Constants, identified in the problem statement by the tag CD, are retained.

• Words identified with an IN tag serve as essential relationships between entities. Such words usually denote some sort of containment of one entity in the other (e.g., of). They are retained.

• There are certain conditions that are satisfied for the sake of clarity and to avoid misinterpretation. For example,

- Whenever the word at occurs in the problem statement, it is retained.
- The word to is compulsorily retained.
- Words that have the JJ tag associated with them are stored in the array if and only if the word to succeeds them.

• There also are certain storage rules that are followed strictly for refinement of the idea about the problem. Many words, which are stored under the purview of these rules, may not at all appear in the final output. But their temporary storage becomes imperative when it comes to identifying the multiplicity of geometric entities correctly.

The word *a* is stored temporarily for future refinement purposes.

- The article *the* is also stored temporarily in the table for correct interpretation.
- For the problem statement in Table I, the entity-connector table produced at this step is shown in Table VIII.

Table VIII
The entity-connector table for the problem in Table I

X (midpoint)	of_1	BC(line)	of_2	ABCD (parallelogram)	
AX(line)	produced_1	meets_1	CD(line)	at_1	Q(point)
CQ(line)	extended_1	to_1	P(point)		
CD(line)	=_1	CQ(line)			
CQ(line)	=_2	PQ(line)			

In the entity-connector table, each connector is appended with an integer. These numbers uniquely identify

each connector and hence, same name connectors can easily be distinguished from each other. The types of the entities are obtained from the complete entity table as built in Step 3. However, there may occur certain NNP-less cases, e.g., 'The bisectors of AB and AC intersect at O....' In such cases, there is provision in our method to assign default names as explained before. These default values are then linked to the NNs in the problem statements. This is helped by the fact that all the entities are stored in Table V in the order that they occur in the problem. Therefore, in order to construct the entity-connector table, i.e. Table VIII, the complete entity table in Table V and the tagged statement as in Table I are consulted. However, the following two cases may be noted:

• The NNP-less NN is singular: the NNP extracted from the previous table is directly assigned to the NN in the current table.

• The NNP-less NN is plural: there will be as many copies of the NN as required in the previous table along with the corresponding default names. Accordingly the default names must be juxtaposed around the connector in the current table. For example, for the statement stated above, the entity- connector table is shown in Table IX.

Table IX
Building of the entity-connector table for an NNP-less statements

DE(bisec-tor)	FG(bisec-tor)	of_1	AB(line)	AC(line)	inter-sect_1	at_1	O(point)

Once the connectors are identified, they are included in the graph of Step 4. The cardinality of a connector is determined as follows:

1) 1 to 1, e.g. BC (bisector) of_1 PQR (angle)

2) 1 to many, e.g. O (point) lies_1 on_1 AB (line) CD (line)

3) many to 1, e.g. AB (line) BC (line) intersect_1 at_1 O (point)

4) many to many, e.g. AB (bisector) BC (bisector) of_1 PQR (angle) RST (angle)

In this part, all multi-word connectors (like intersect at) are appended into one using an underscore ( ). So intersect_1 at_1 becomes intersect_1_at_1. New nodes are created for the connectors and appropriate edges created in the graph. The resulting graph for the problem statement in Table I is shown in Fig. 3 (the connectors are shown in boxes).

Finally, note that while parsing a statement like '...α of β of γ meets E at δ...', the connection is made as: α --- meets_1_at_1 --- δ instead of γ and δ being connected through *meets at*. For example, a statement like '...The bisector BE of angle B of triangle ABC meets CD at O..'. will result in the following connections in the graph: BE (bisector) --- meets_1_at_1 --- O (point) instead of ABC and O being connected through *meets at*.

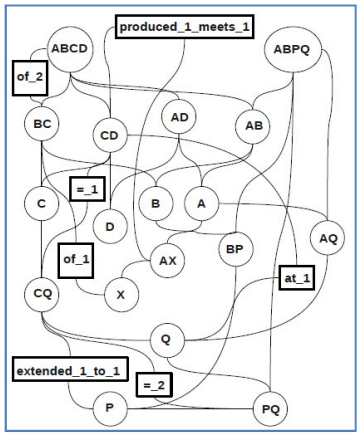

Fig. 3 Connected parse graph - from Fig.2 and Table VIII

2.3 Translation of the Parse Graph into a Structured Summary or Intermediate Representation

The parser generates a parse output in the form of a graph. The graph can be considered as a formal representation of an input statement. However, in the context of the text-to-diagram conversion problem, this graph cannot directly generate the graphic functions required to draw the underlying diagram. Therefore, we design a translator that translates the parse graph into a graphics-friendly intermediate representation. This intermediate representation can be considered as a structured summary from the parse graph. This summary is directly used to frame the graphic functions to draw the underlying diagram.

The translator works as follows. Firstly, all the entities are represented in NN = NNP form e.g., midpoint = B. Next the entities are numbered distinctly according to their appearance. e.g., midpoint_1 = B. If an entity has an attribute *attr* it is represented as *attr(NN 1=NNP)*. The entities are then merged according to the relations between them. If two entities are connected with a connector *conn* then, in the summary, it appears as:

conn_1(attr1(NN1_1=NNP1), attr2(NN2_1=NNP2))

For instance, the summary of the statement 'The line CQ is extended to P' is:

extended_1_to_1(line_1=CQ,point_1=P).

If a relation shows a sense of belonging of one entity to another then it is represented as:

attr1(NN1 1=NNP1).atr2(NN2 1=NNP2), e.g. 'The midpoint of BC is X' is represented as:

(midpoint_1=X).(side_1=BC).

For the problem statement in Table I the summary generated from the graph in Fig. 3 is Table X.

Table X
The formal representation of the parse graph

```
midpoint_1=X
side_1=BC
point_1=B
point_2=C
parallelogram__1=ABCD
line_1=AB
line_2=CD
line_3=AD
point_3=A
point_4=D
line_4=AX
point_5=Q
line_5=CQ
point_6=P
parallelogram_2=ABPQ
line_6=BP
line_7=PQ
line_8=AQ
(midpoint_1=X).(side_1=BC)
(side_1=BC).(parallelogram_1=ABCD)
produced_1_meets_1(line_2=CD,line_4=AX)
at_1(line_2=CD,point_5=Q)
extended_1_to_1(line_5=CQ,point_6=P)
=_1(line_2=CD,line_5=CQ)
=_2(line_5=CQ,line_7=PQ)
```

2.3.1 Utility of the Intermediate Representation

The intermediate representation or summary generated by the translator is natural language-free. It does not contain any of the language constructs (words, phrases, punctuation, etc.). Hence, it can be considered as a unique representation of geometric information extracted from the problem statement. We term this geometric information as the useful information for further reference i.e. for diagram drawing or problem solving purpose.

The summary is unique for the same problem stated in different ways in a natural language. We have followed the same logical methodology while designing the system. The system has been tested for many problems. It has been observed that for most of them, the sentence constructs, phrasing, punctuation, etc. have no effect on the final representation (summary). That is, the same problem stated in different ways, generate the same summary ultimately.

The summary is also unambiguous. Absence of ambiguity is an essential feature for any NLP implementation. Especially when the discussed system deals with generation of information that should be ready to be used for diagram generation and problem solving purpose, this feature becomes indispensable.

The summary contains meaningful yet optimal information about different geometric elements and relations of a diagram; but it does not contain the information regarding the position or size at which the elements are to be drawn actually to generate the diagram. Parser 2, with the help of GeometryNet knowledge base, can instantiate the different geometric parameters or variables listed in or resolved from the summary and thus generate a set of arguments of specific graphic functions (e.g. line-draw). A graphics module can then generate accurate (geometrically correct) diagram through several calls to the graphic functions.

3. Evaluation

Our evaluation strategy consists of two steps: (i) evaluate whether graph generated by the parser is correct and (ii) whether the intermediate description or summary generated by the translator upon translating the parse graph is correct.

3.1 Test Set and Evaluation Strategy

We develop a test data set for evaluating the parser and the translator. The test set consists of 51 geometry problems taken from school level mathematics books of grades 8, 9 and 10 across various school boards in India. The test set problems were not used in the developmental stage. They are also different from the problems used in constructing the GeometryNet. The test set is generated with some extra care so that the problems reveal enough diversity as encountered in high school geometry books. The same problem (i.e. generates the same final diagram) may be narrated differently in different books. Such problems are also considered to check uniqueness of the method. Note that if the final diagram is same for two differently narrated problems, then their formal descriptions should also be same. For each test problem, the intended parse graph and summary (or intermediate representation) are available.

The correctness of both the graph (the output of parser) and the summary (the output of the translator) is checked manually. Though the parser might have been evaluated by a graph-matching approach (matching between the parser's output graph and the groundtruthed graph for the input), we preferred to do it manually. This is because a graph matching approach may give us some matching score which says little about the parser's performance. Note that the parser's accuracy depends on whether the entity table and the entity-connector tables are built properly. Any mistake in generating entities of these tables would result in mistakes in the final parse graph. Therefore, in order to check the accuracy of all the intermediate stages of parsing, we followed a manual evaluation of the method. For an input problem, if the parser generated graph does not match with its intended graph, we trace back to locate problems in one of the five parsing steps.

The same is done for evaluating the translator's accuracy. For an input problem, the translator's output is checked with the intended intermediate representation of the problem and the errors are located and analyzed manually. The evaluation scheme is binary in nature. For an input problem either the output (of the parser and/or the translator) is correct or is not. Partially correct output may generate partially correct drawing of the underlying diagram (or language translation etc.) and therefore, evaluation of partially correct parsed output can be addressed in future. However, using our binary evaluation scheme we found that the present method perfectly parses and translates 42 out of

51 problems. We analyze the errors for the 9 cases where the method fails and find that all errors are due to problems in parsing; the output parse trees are wrong. For example in the following problem, AOC is actually a straight line. But since it has been denoted using three letters, our approach fails to recognize it: '`OA, OB, OC, OD meets at O. If angle AOB + angle BOC = angle COD + angle DOA, prove that AOC is a straight line.`' However, no case is encountered where translation of a correct parse tree results in a wrong intermediate representation. This is due to the straightforward nature of the translation module that takes a parse tree and converts it into a summary consisting of a set of graphics-friendly statements.

4. Conclusion

In the context of automatic text to diagram conversion, translation of a natural language problem statement into a formal description is attempted. The input natural language statement is parsed and then converted into an intermediate formal representation. The parser is a rule based one. It makes use of a parts-of-speech tagger and an ontology, called the GeometryNet that contains knowledge about geometry terms and relations. The parser generates two intermediate tables to construct a parse tree as the final outcome. The parse tree is later translated into a summary consisting of a set of statements. The summary is basically a graphics-friendly, language independent, unambiguous intermediate representation of the input problem. The entire approach is tested with 51 high school problems out of which 42 are correctly parsed and errors for the other 9 problems are analyzed. To the best of our knowledge, this is one of the pioneering attempts that presents an algorithmic approach along with exhaustive tests and validations in the context of automatic text to diagram conversion.

The future extension of this study involves further use of the structured summary or the intermediate representation. Fig. 1 shows that the intermediate representation is next converted (with the help of the GeometryNet) into a draw-able representation to facilitate the diagram drawing process. Once the diagram is drawn, the entire approach can be tested in an end-to-end manner, i.e. for a given set of problem statements, how many problems are successfully converted into underlying diagrams. The intermediate representation or the parse tree can also be used for machine translation. Moreover, the present study considers geometry problems for solving text to diagram conversion problem. Upon achieving satisfactory results for this class of problems, the system can be next extended to work on finding intermediate representation schemes for problems occurring in other disciplines like Physics, Engineering, etc.

References

[1] A. Mukherjee & U. Garain, A Review of Methods for Automatic Understanding of Natural Language Mathematical Problems, *Artificial Intelligence Review, 29*(2), 2008, 93–122.

[2] A. Mukherjee & U. Garain, Understanding of natural language text for diagram drawing, *Proc. of IASTED International Conf. on Artificial Intelligence and Soft Computing*, Palma De Mallorca, Spain, 2009, 138-145.

[3] A. Mukherjee, U. Garain, & Mita Nasipuri, On construction of a GeometryNet, *Proc. of 25th IASTED International Multi-Conference: Artificial Intelligence and Applications (AIAP)*, 2007, 530-536.

[4] C. Baral, M. Gonzalez, J. Dzifcak, & J. Zhou. Using Inverse Lambda And Generalization To Translate English To Formal Languages, *Proc. of Int. Conf. on Computational Semantics*, Oxford, England, 2011.

[5] C. Fellbaum, W*ordnet: an electronic lexical database* (MIT Press, 1998).

[6] Ch. Kop & H.C. Mayr. Mapping functional requirements: from natural language to conceptual schemata, *Proc. of 6th Int. Conf. on Software Engineering Advances (ICSEA)*, Cambridge, USA, 2002.

[7] D. G. Bobrow, Natural language input for a computer problem solving system, PhD thesis, Department of Mathematics, MIT, Cambridge, 1964.

[8] E. Brill, A simple rule-based part of speech tagger. HLT '91: *Proc. of the Workshop on Speech and Natural Language*, Morristown, NJ, USA, 1992, 112-116.

[9] G. S. Novak Jr., Computer understanding of physics problems stated in natural language, *American Journal of Computational Linguistics*, Microfiche 53, 1976.

[10] J. F. M. Burg & R.P. van de Riet, Analyzing informal requirements specifications: a first step towards conceptual modeling, *Proc. of 2nd Int. Workshop on Applications of Natural Language to Information Systems*, Amsterdam, 1996.

[11] K. D. Forbus, C.K. Riesbeck, L. Birnbaum, K. Livingston, A. Sharma, & L. Ureel. Integrating natural language, knowledge representation and reasoning, and analogical processing to learn by reading, *Proc. of Conf. on Artificial Intelligence (AAAI)*, Vancouver, BC, 2007.

[12] N. Do, H. P. Truong, & T. T. Tranan, Approach for translating mathematics problems in natural language to specification language COKB of intelligent education software. *Proc. Int. Conf. on Artificial Intelligence and Education (ICAIE)*, 2010, 324 -330.

[13] M. Ilieva & H. Boley, Representing textual requirements as graphical natural language for UML diagram generation, *Proc. of 20th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE)*, San Francisco, CA, USA, 2008, 478-483.

[14] V. K. Chaudhri , B. Bredeweg, R. Fikes, S. McIlraith, & M. P. Wellman, A categorization of KR&R methods for requirement analysis of a query answering knowledge base. *Proc. of 6th Int. Conf. on Formal Ontology in Information Systems (FOIS)*, 2010, 158-171.

[15] W. Lu, H. T. Ng, W. S. Lee, & L. S. Zettlemoyer, A generative model for parsing natural language to meaning representations, *Proc. of Empirical Methods in Natural Language Processing (EMNLP)*, 2008.

[16] W. Wong, S. Hsu, S. Wu, C. Lee, & W. Hsu, LIM-G: learner-initiating instruction model based on cognitive knowledge for geometry word problem comprehension. *Computers and Education, 48*(4), 2007. 582–601.