

NATURAL
LANGUAGE ENGINEERING



CAMBRIDGE
UNIVERSITY PRESS

**Text-to-Diagram Conversion: a Method for Formal
Representation of Natural Language Geometry Problems**

Journal:	<i>Natural Language Engineering</i>
Manuscript ID:	NLE-ARTC-11-0554
Manuscript Type:	Article
Date Submitted by the Author:	29-Dec-2011
Complete List of Authors:	Sengupta, Sarbartha; Indian Institute of Technology, Bombay, Computer Science and Engineering Sen, Anirban; Tata Consultancy Services, Software Design Chakraborty, Dipanjan; Indian Institute of Technology, Delhi, Computer Science and Engineering Mukherjee, Anirban; RCC Institute of Information Technology, Information Technology Garain, Utpal; Indian Statistical Institute, Computer Vision & Pattern Recognition Unit
Keywords:	Natural Language Processing in Computer-Assisted Language Learning, Parsing

SCHOLARONE™
Manuscripts

Text-to-Diagram Conversion: a Method for Formal Representation of Natural Language Geometry Problems

Sarbartha Sengupta, Indian Institute of Technology, Bombay;

sarbartha@cse.iitb.ac.in

Anirban Sen, *Tata Consultancy Services, India*; anir.sen@tcs.com

Dipanjan Chakraborty, Indian Institute of Technology, Delhi;

dipanjan.siy10@cse.iitd.ac.in

Anirban Mukherjee, *RCC Institute of Information Technology, Kolkata*;

anirban.mukherjee@rcciit.in

Utpal Garain, *Indian Statistical Institute, Kolkata*; 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, namely the GeometryNet 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.

Index Terms

Text to Diagram conversion, Geometry, Natural language problem statements, Formal description, Parsing algorithm, GeometryNet, Parse graph, Complexity analysis.

I. INTRODUCTION

The problem of generating diagram from text is a problem occurring in many branches of science and engineering like physics, geometry, engineering drawing, etc. In these problems, 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. The paper in [1] provides a state of the art survey of the relevant studies and the further research needs. The review paper attempts to cover the significant relevant works till 2007. However, researchers are still continuing to contribute in this field [12].

Drawing the diagrams as described by geometry text 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]. In order to generate diagrams from text, the proposed system consists of several modules. Fig. 1 shows a schematic diagram of the overall system. 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 the *GeometryNet* knowledge base [3]. The main idea of developing the GeometryNet is borrowed from the concept of Word Net [5]. The GeometryNet serves as a lexical resource to the parsing

module. 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.

The goal of designing the first level parser in Fig. 1 is to convert the input natural language statement into a formal representation. In the field Software Engineering (SE) modeling, several researchers have interest in generating formal models from informal textual description [6], [10]. Their goal is to analyze natural language specification to discover a underlying software model (e.g., object oriented model). After from SE community, many NLP researchers have attempted this problem. 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] attempt to translate natural language sentences to formulae in a formal or a knowledge representation language. They use a syntactic combinatorial categorial 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 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. The goal of generating this intermediate representation is to make suitable for subsequent applications. For example, in the context of the text-to-diagram conversion a second parser, i.e. *parser 2* acts on the intermediate formal representation of the problem statement, aided by the *GeometryNet* [3] knowledge base, to generate a set of graphical functions. A graphical module can then generate the required diagram from the set of functions.

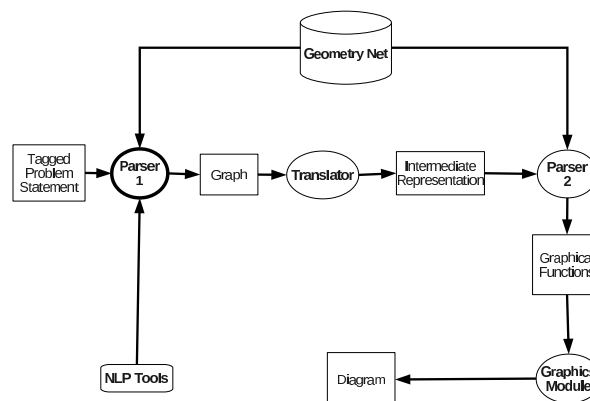


Fig. 1. Automatic text-to-diagram conversion: a schematic diagram.

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 required 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 the school level geometry problem statement stated in a natural language.
- 2) The parse graph is translated into an intermediate representation that can be used in many applications, for example, drawing of the underlying diagram, language translation, problem solving, etc.
- 3) The time complexities of the parser and the translator have been analyzed to show their efficiency.
- 4) The parser's and the translator's accuracies are tested on an exhaustive test-set generated from a collection of high school level text books. Analysis of the errors show the future research needed to improve the parsing method.

The rest of the paper is organized as follows. Section II of the paper describes our approach to solve the problem. Section III presents the translator that translates the parser's output, i.e. the graph into a graphics-friendly format. The utility of this intermediate representation is also highlighted in the context of the text to diagram conversion problem. The time complexities of the parser and the translator are addressed in section IV. Section V describes the evaluation strategy, data set and experimental results including an error analysis in section V-C. Finally, Section VI concludes the paper highlighting the future scope of the work and ways to overcome probable shortcomings.

II. 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 along with a translator. The parser analyzes the statement using natural language processing tool namely, a parts-of-speech tagger and a lexical knowledge base, namely the GeometryNet. 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 goal of generating this summary is to convert the formal description as a graphics-friendly intermediate representation so that this representation later on can be used to draw the underlying diagram. Firstly, we present the essence of the GeometryNet and then the parsing algorithm is described. Finally, the approach for translating the parse graph into an intermediate representation is explained in a separate section.

A. The GeometryNet

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. About 300 school level geometry problems are selected from the books taught at Grades 8-10 in India. These problems are used to develop the GeometryNet. The GeometryNet consists of several hierarchical relations like subordinatesuperordinate is represented as hypernymhyponym relation, partwhole relation is shown by MeronymHolonym relations [5]. Fig. 2 shows a small portion of the Net (*LS* denotes Line Segment). Note that the hierarchical relations in pairs of geometric entities are different from geometric relations. For example, there exists a Meronym relation between entries *circle* and *diameter* and this relation is different from geometric relations like *produced* or *extend*. From implementation point of view GeometryNet is a set of text files 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:

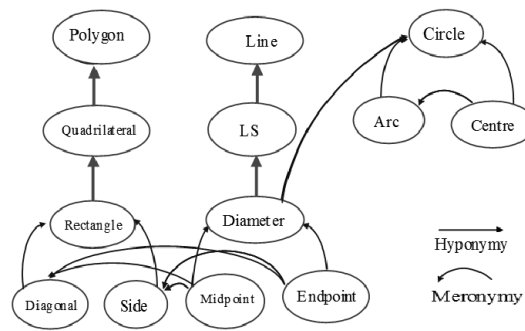


Fig. 2. A small portion of GeometryNet: a graphical view.

```

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 information that a *vertex* is basically a point having a coordinate pair as parameter is not listed explicitly against *vertex*. Similarly, the information like each *side* has two points and a slope m is also not available in the entry for *Parallelogram*. These facts are implied as in the GeometryNet structure there is inheritance of properties of the subordinate entities from the superordinate entities. Parameters of *vertex_1*, *vertex_2*, *vertex_3*, *vertex_4* are inherited from the generic superordinate entity *point* which is separately listed as:

```

point
(x, y)
end

```

The inheritance occurs because the Net defines vertex as a point.

```

vertex
point
end

```

By default, *vertex_1* implies *point_1* with parameters (x_1, y_1) , *vertex_2* implies *point_2* with (x_2, y_2) and so on. Similarly, from the following generic definition of *side*, *line* and *m* the parameters of *side_1*, *side_2*, *side_3*, *side_4* are inherited. *side_1* is a line and has *point_1*, *point_2*, *m1* where $\text{point}_1 = (x_1, y_1)$, $\text{point}_2 = (x_2, y_2)$ and $m_1 = (y_2 - y_1) / (x_2 - x_1)$; *side_2* is a line and has *point_2*, *point_3*, *m2* where $\text{point}_2 = (x_2, y_2)$, $\text{point}_3 = (x_3, y_3)$, $m_2 = (y_3 - y_2) / (x_3 - x_2)$ and so on. The entry associated to *side* is as follows.

```
side
line
end
```

The entity *line* is as follows.

```
line
point_1
point_2
m
end
```

The entity *line* refers to its slope, *m* which appears as a separate entity as shown below.

```
m
# (y2-y1) / (x2-x1)
end
```

The attributes and relations are also represented in a similar way. For example, consider an attribute *concentric*. This attribute is coded as below.

```
concentric
circle_1
circle_2
#r1 ! r2
#xc1=xc2
#yc1=yc2
end
```

Similarly, a geometric relation *extend* is represented as:

```
extend
produced
end
```

The relation *produced* is coded as follows.

```
produced
ls_1
point_3
#y3-m1x3=y1-m1x1
#m2=m1
end
```

B. 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*. More on how lines are determined is given next where this step is described in detail.

- Step 2. Build an entity table listing all the proper nouns (NNPs), the corresponding Common Nouns (NNs) and the attributes (e.g. isosceles, equilateral, etc.) of the NNPs (if any).
- Step 3. Use GeometryNet to complete the entity table that remained incomplete in step 2.
- Step 4. Construct a parse graph from the entity table. The NNPs are represented by nodes and they 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.

1) *Description of 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 the case 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, $X, Y, \text{ and } Z$ are the midpoints of sides $AB, BC, \text{ and } AC$ 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 this kind of a situation 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, $AB = BC \text{ and } CD = DF$ will call for a division at ‘and’.

2) *Description of 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. 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.

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

3) *Description of 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 them. Table IV shows a partial information extracted from the GeometryNet.

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 the 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.

Sometimes we may encounter *NNs* without *NNPs* associated with them in the problem statement. For example, The *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.

Note that the NNP for *bisector* is blank. 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

¹In GeometryNet, generalized entries occur before specialized entries in the knowledge-base. So *parallelogram* occurs before *square*, etc.

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

TABLE VI
EXAMPLE OF AN *NNPless* table

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

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 $\angle BCA$ meet at $O \dots$), 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, \dots 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.

4) *Description of 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. eg. 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 $\angle ABC$ and $\angle BAC$ 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. 3. Note that only the names of the entities are shown and the attributes are omitted, to avoid cluttering.

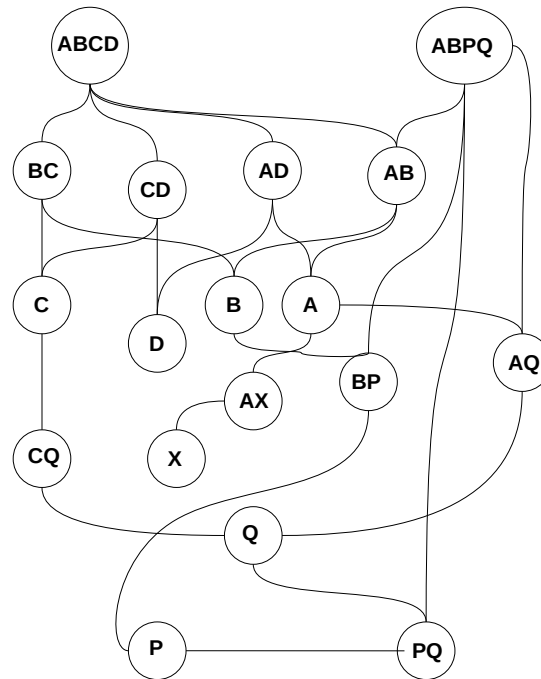


Fig. 3. The graph at Step 4 corresponding to the entity Table V.

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.

5) *Description of 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 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*, *VCN* 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 (eg. *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 *NNPless* cases; for example, 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 statement. 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

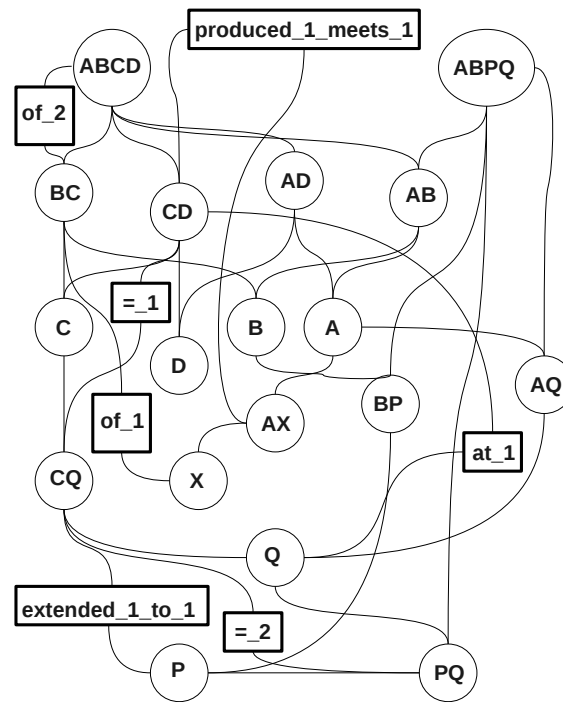


Fig. 4. The final graph corresponding to the problem in Table I.

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 *NNPl_{ess}* STATEMENT.

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

Once the *connectors* are identified, they are included in the 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 as in Fig. 4 (the *connectors* are shown in boxes).

Finally, note that while parsing a statement like $\dots\alpha$ of β of γ meets ϵ at $\delta\dots$, 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 \dots 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*.

III. TRANSLATION OF THE PARSE GRAPH INTO AN 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. eg. midpoint = B. Next the entities are numbered distinctly according to their appearance. eg. midpoint.1 = B. If an entity has an attribute *attr* it is represented as $attr(NN_I=NNP)$. 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: $conn_I(attr1(NN1_I=NNP1), attr2(NN2_I=NNP2))$. For instance, the summary of the statement The line CQ is extended to P is: $extended_1_to_1(line_1=CQ, point_1=P)$.

Sense of Belonging: If a relation shows a sense of belonging of one entity to another then it is represented as $Attr1(NN1_I=NNP1).Attr2(NN2_I=NNP2)$, 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. 4 is Table X.

TABLE X

THE FORMAL DESCRIPTION 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)

```

A. Utility of the Intermediate Representation

In this section, we discuss about the utility of the translator in the context of the text-to-diagram conversion problem. The primary aim of this paper is to generate a language-independent, intermediate form (from the problem statement stated in general English), which could be used as an unambiguous representation of the input statement. The summary generated by the translator has quite a few attributes which makes it suitable to be used in text-to-diagram conversion. Let us look at the properties of the summary generated above in Table X.

- The summary generated by the system is natural language-free. The summary 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. When we read a problem statement in general, we mentally use the language constructs, words, phrases, punctuation, etc. for the sake of understanding. However, there should be no doubt in accepting that once the meaning of the problem is clearly understood, we retain only the useful information, which is used further 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 as detailed in section V. 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, generates the same summary ultimately.
- The summary is 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. From the example provided above, we can see that the information content is completely unambiguous. The system also attaches unique keys to each of the geometric entities, in case of repetition. So, any standard diagram generating software should be able to identify each of these entities uniquely.
- The summary is generated in a way so that it can be used to call specific graphic functions to generate a diagram that is implied by the problem statement. It can be fed to any standard graphics module to generate accurate (geometrically correct) diagrams. For this purpose, the summary may need to be parsed (refer to *Parser 2* in Fig. 1) and converted to the required input form, compatible with the module.
- The language independent nature of the summary makes it suitable for inter-natural-language translation and for problem solving purpose. More in this regard has been discussed in section VI.

IV. TIME COMPLEXITY ANALYSIS OF THE PARSER AND THE TRANSLATOR

Here we attempt to derive a complexity analysis of our parsing approach. We take up each step as described in section II-B and discuss the complexity. The generation of Table V broadly involves the following steps:

- The first step involves dividing the problem into *lines*. In this step each word from the problem statement is read and appended to a table (Table II). When a comma (,), a full stop(.) or the word *and* (which doesn't separate the same kind of grammatical entities) is encountered then we switch to a new row in the table and continue. This requires scanning the problem statement once. So the complexity of this step is linear in terms of the number of words (say, n) in the problem statement.
- The second step is to identify the words which can be classified as geometric entities, along with their types and attributes and put them into the entity table. Hence, the complexity of this step is also linear on the number of words in the problem statement.
- The next step is to identify the types (NNs) of some of the entities which are not directly mentioned in the problem. This is done by simply counting the number of vertices used to represent the entity and matching it against a look-up table containing entity names and number of vertices. The look-up table is implemented using hashing technique and the look-up is done in constant time. The counting process is bounded by the number of vertices in the entity. In order to determine names of the nameless entities, the GeometryNet [3] is consulted. The number of vertices required to correctly identify such an entity is obtained and assigned default values. This is done in constant time as the GeometryNet is implemented as a set of files and pointers. Things become more complex if the entity is plural. Then we have to scan the other words in the sentence to know the actual number of entities implied by the plural entity. This requires multiple scan of the input statement but the overall complexity remains linear in terms of n .

Generation of the intermediate graph as in Fig. 3 involves:

- Decomposing each entity into atomic entities requires processing of each vertex of each entity in Table V. This step is therefore linear in terms of the number of vertices in each entity in Table V, which will be less than the number of words in the problem statement in most cases.
- Next, each entity is converted to its canonical form. This step too is linear in terms of the number of entities in Table V, which is bounded by the number of words in the problem statement.
- Now each canonical entity must be checked against existing nodes to prevent duplication. This is done in constant time using a hash technique.
- Next, the nodes are put into parent-child relationships to form the intermediate graph as in Fig. 3. This step is linear in terms of the number of nodes in the graph in Fig. 3. In our implementation, the graph is realized using an adjacency matrix. This is in turn bounded by the number of words in the problem statement. So this step is linear in terms of n .

The generation of the entity-connector Table VIII and including the connectors in the parse graph broadly involves the following steps:

- *Storing the geometrical entities (with tags NNP, NN, and CD)*: In this phase, the problem statement (broken into lines) is scanned on a line-by-line basis. Whenever a geometrical entity or a constant term is encountered, the program stores it in a table (Table VIII in this case). So, the parsing complexity varies linearly with the number of words, n .
- *Storing the connectors (with tags VBZ, VBN, JJ, SYM)*: In this phase too, the problem statement is scanned on a line by line basis. Each word in the problem is checked for the tags corresponding to the connectors considered. On encountering a connector, the same is stored in the table. So, the complexity varies linearly with the number of words in the problem statement.
- *Finding common nouns for the entities stored in the table*: As all the proper nouns mentioned in the problem statement have been stored in the table, those which do not have a corresponding common noun mentioned explicitly in the problem are to be associated with their corresponding entity types (common nouns). This phase basically scans the entity Table V generated in Step 4 to find the corresponding common nouns. It requires scanning every row of Table VIII to find such proper nouns and then for each one of them, scanning every row of Table V to obtain its matching common noun. Considering the number of rows in Table V to be much less than n , we find that the algorithm's complexity varies linearly with n . Same is true for finding proper nouns (NNPs) for the common nouns stored in the table.
- *Assigning a unique key to each of the repetitive connectors*: This phase assigns a unique key value to each of those connectors (with an attempt to identify them uniquely). The procedure uses a one-dimensional array to store all connectors stored in Table VIII. Each row of Table VIII is parsed and entities on the left and right of a connector are stored in queues. Depending on the number of entities in the queues the cardinality of the relation is determined and that many nodes are created in the graph. For each of these connectors, a key is generated which is incremented with every repetition. Once all of the connectors have been appended with their respective keys, the old connectors are replaced with them. This approach clearly has to scan every word stored in the entity table, which has an upper bound of the number of words in the problem.
- The step to include the connectors in the graph requires parsing of entity-connector table, Table VIII as many as the number of connectors present in the table. This step is therefore linear in terms of the number of entries in Table VIII, which is again bounded by the number of words in the original problem statement. So this step is linear in terms of n .

The translator traverses the parse graph in order to generate the summary. The time complexity of translator is therefore bounded by the number of nodes in the graph, which is bounded by the number of words in the problem statement. Hence, both the parser and the translator works in $\Theta(n)$ time where n is the number of words in the problem statement. It can be noted that the linearity of the time complexity is achieved by sacrificing the space complexity. Many intermediate tables, files and pointers as mentioned

above are maintained in order to keep the time complexity of our method linear on the number of word in the input problem statement.

V. EVALUATION

In representing an input natural language problem into a formal description, the parser generates a graph and the translator generates an intermediate structured description. The graph itself is a formal description of the input statement but as the final goal is to use this graph in some application drawing the underlying diagram the graph is further translated into another description which is easy to convert into the required graphic functions. Therefore, 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.

A. 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. By doing this, it is true that the evaluation scheme becomes 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 (how much correct is the parse graph or the translated representation) in future. However, using our binary evaluation scheme we found that the present method perfectly parses and translates 42 out of 51 problem. Some of these results and related discussions are presented in the next section. Errors for processing the remaining 9 problems are also analyzed and presented in section V-C.

B. Experimental Results

In this subsection we discuss some of the experimental results.

- Problem 1. BP(NNP) bisects(VBZ) angle(NN) ABC(NNP) and(CC) PB(NNP) is(VBZ) produced(VBN) to(TO) Q(NNP) .(.) Prove(VBZ) angle(NN) ABQ(NNP) =(SYM) angle(NN) CBQ(NNP) .(.)

The parse graph for this problem is in Fig. 5.

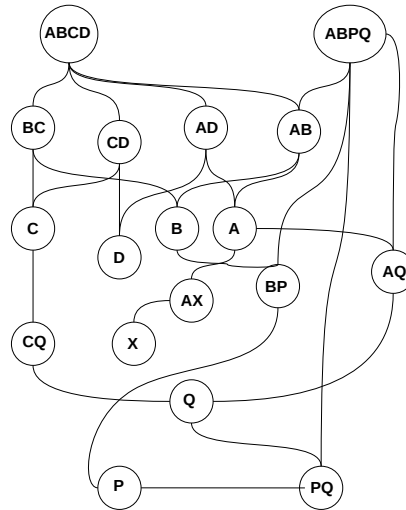


Fig. 5. Output parse graph for Problem 1.

The intermediate representation generated by the translator is:

```

line_1=BP
point_1=B
point_2=P
angle_1=ABC
line_2=AB
line_3=BC
point_3=A
point_4=C
point_5=Q
angle_2=ABQ
line_4=BQ
angle_3=CBQ
bisects_1(line_1=BP, angle_1=ABC)
produced_1_to_1(line_1=BP, point_5=Q)
=_1(angle_2=ABQ, angle_3=CBQ)
    
```

- Problem 2. In(IN) triangle(NN) PQR(NNP) ,(,) angle(NN) PQR(NNP) =(SYM) angle(NN) PRQ(NNP) .(.) QR(NNP) is(VBZ) produced(VBN) to(TO) M(NNP) and(CC) N(NNP) .(.) Prove(VBP) that(IN) angle(NN) MQP(NNP) =(SYM) angle(NN) NRP(NNP) .(.)

The graph returned by the parser for this problem is given in Fig. 6 and the intermediate representation out of the parse graph is:

```

triangle_1=PQR
line_1=PQ
line_2=QR
line_3=PR
point_1=P
point_2=Q
point_3=R
    
```

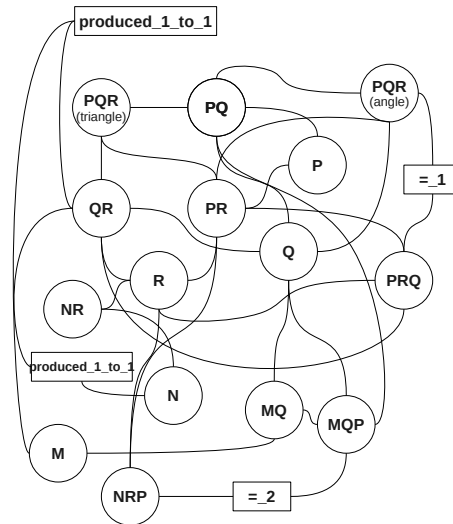


Fig. 6. Output parse graph for Problem 2.

```

angle_1=PQR
angle_2=PRQ
point_4=M
point_5=N
angle_3=MQP
line_4=MQ
angle_4=NRP
line_5=NR
=_1 (angle_1=PQR, angle_2=PRQ)
produced_1_to_1 (line_2=QR, point_4=M)
produced_1_to_1 (line_2=QR, point_5=N)
=_2 (angle_3=MQP, angle_4=NRP)

```

- Problem 3. In(IN) triangle(NN) ABC(NNP) ,(,) P(NNP) ,(,) Q(NNP) ,(,) R(NNP) are(VBP) the(DT) midpoints(NN) of(IN) AB(NNP) ,(,) AC(NNP) and(CC) BC(NNP) respectively(RB) and(CC) AQ(NNP) =(SYM) 3(CD) cm(DG) ,(,) BC(NNP) =(SYM) 7(CD) cm(DG) .(.) Find(VBP) PR(NNP) ,(,) PQ(NNP) .(.)

The parse graph generated for this problem is given in Fig. 7 and the intermediate representation is:

```

triangle_1=ABC
line_1=AB
line_2=BC
line_3=AC
point_1=A
point_2=B
point_3=C
midpoint_1=P
midpoint_2=Q
midpoint_3=R
line_4=AQ

```

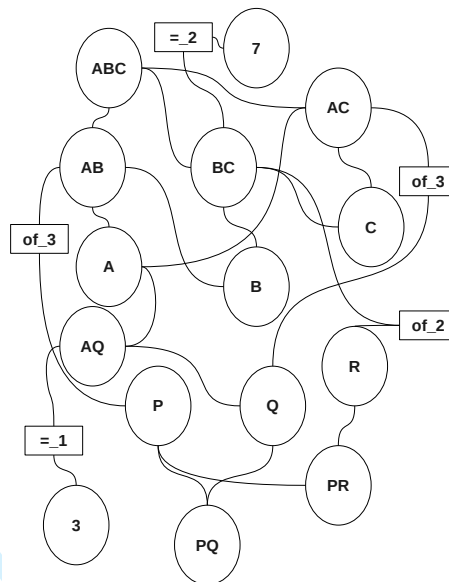


Fig. 7. Output parse graph for Problem 3.

```

line_5=PR
line_6=PQ
const_1=3
const_2=7
(line_1=AB) .(midpoint_1=P)
(line_2=BC) .(midpoint_3=R)
(line_3=AC) .(midpoint_2=Q)
=_1 (line_4=AQ, const_1=3)
=_2 (line_2=BC, const_2=7)
    
```

- Problem 4. PQ(NNP) is(VBZ) perpendicular(JJ) to(TO) RS(NNP) .(.) PT(NNP) parallel(JJ) to(TO) RS(NNP) .(.) Prove(VBP) that(IN) angle(NN) RQT(NNP) =(SYM) 128(CD) degree(DG) .(.)

The graph produced is in Fig. 8 and the translation of parse graph into the intermediate representation is:

```

perpendicular (line_1=PQ)
point_1=P
point_2=Q
line_2=RS
point_3=R
point_4=S
line_3=PT
point_5=T
angle_1=RQT
line_4=QR
line_5=QT
const_1=128
perpendicular_1_to_1 (perpendicular
                      (line_1=PQ), line_2=RS)
parallel_1_to_2 (line_2=RS, line_3=PT)
    
```

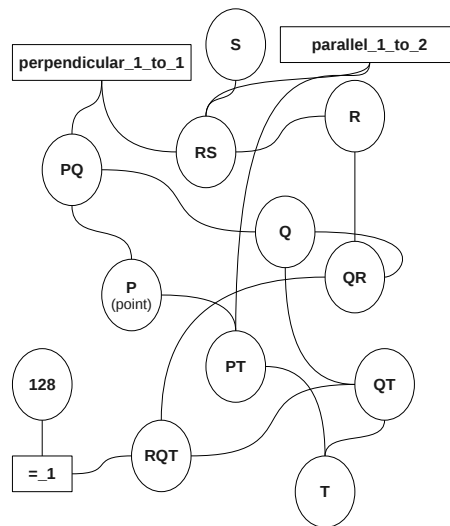


Fig. 8. Output parse graph for Problem 4.

=_1 (angle_1=RQT, const_1=128)

- Problem 5. The(DT) bisectors(NN) of(IN) angle(NN) B(NNP) and(CC) angle(NN) C(NNP) of(IN) an(DT) equilateral(JJ) triangle(NN) ABC(NNP) meet(VBP) at(IN) O(NNP) .(.) Find(VBP) angle(NN) BOC(NNP) .(.)

The parse graph for Problem 5 is shown in Fig. 9 and the intermediate representation is:

```

bisector_1=GH
point_1=G
point_2=H
bisector_2=IJ
point_3=I
point_4=J
angle_1=B
angle_2=C
equilateral(triangle_1=ABC)
line_1=AB
line_2=BC
line_3=AC
point_5=A
point_6=O
angle_3=BOC
line_4=BO
line_5=CO
(bisector_1=GH) . (angle_1=B)
(bisector_2=IJ) . (angle_2=C)
(angle_1=B) . (equilateral(triangle_1
                           =ABC))
(angle_2=C) . (equilateral(triangle_1
                           =ABC))

```

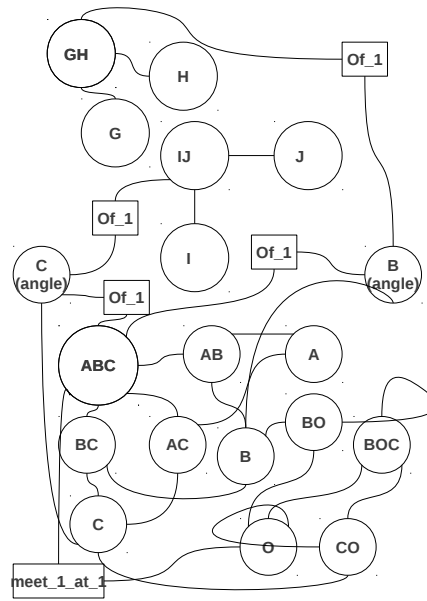


Fig. 9. Graph for the problem

```
meet_1_at_1 (bisector_1=GH, point_6=O)
meet_1_at_1 (bisector_2=IJ, point_6=O)
```

C. Error Analysis

It is noted that out of 51 test problems 42 problems were perfectly parsed and translated into intermediate representation. However, for 9 cases, the method fails. We analyze these errors and find that all errors are due to problems in parsing. The output parse trees are wrong. 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. Analysis of the parsing errors reveals that the reasons behind the errors are as follows:

- Type-I. If the number of occurrences of a plural entity depends on the nature of another entity, then our model cannot identify the actual number of occurrences of that plural entity.

For example, in the following problem, the number of bisectors depends on the properties of the entity *triangle*: The bisectors of the angles of triangle ABC meet at O. The intended entity table is as in Table XI.

TABLE XI
EXPECTED TABLE

DE	bisector	none
FG	bisector	none
HI	bisector	none
A	angle	none
B	angle	none
C	angle	none
ABC	triangle	none

But the entity table actually produced by our approach is in Table XII.

TABLE XII
ACTUAL OUTPUT

DE	bisector	none
F	angle	none
ABC	triangle	none

The problem could be solved if we incorporate the property of a triangle to determine the number of occurrences of the entity angle. But at this moment our parser cannot resolve this problem.

- Type-II. When an entity is denoted by more number of letters than such entities usually are denoted by.

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 $\text{angle } AOB + \text{angle } BOC = \text{angle } COD + \text{angle } DOA$, prove that AOC is a straight line.

- Type-III. If a problem statement contains relations like *same side of* which has a common noun (in this case *side*) then our model will identify the common noun as a separate entity, not as a relation.

For example, the following problem should have produced the entity table in Table XIII but it actually produced Table XIV: AB is parallel to DE and $\text{angle } ABC$ and $\text{angle } DEF$ are on the same side of AB such that $\text{angle } ABC = \text{angle } DEF$. Prove that BC is parallel to EF .

TABLE XIII
EXPECTED OUTPUT

AB	line	none
DE	line	none
ABC	angle	same_side_of
DEF	angle	same_side_of
AB	line	none
ABC	angle	none
DEF	angle	none
BC	line	none
EF	line	none

TABLE XIV
ACTUAL OUTPUT

AB	line	none
DE	side	same
ABC	angle	none
DEF	angle	none
AB	line	none
ABC	angle	none
DEF	angle	none
BC	line	none
EF	line	none

- Type-IV. If a problem statement contains an entity and it is referred to by a pronoun at some other part of the problem, then our approach fails to relate both.

For example, in the following problem statement, the last sentence refers to the $\triangle ABC$. But our model treats it as a separate triangle: ABC is a triangle where $\text{angle } ABC = \text{angle } BCA$. Prove that it is an equilateral triangle.

VI. 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 in details. 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. The Fig. 1 shows that the intermediate representation is next converted (with the help of the GeometryNet) into a set graphic functions for drawing the underlying diagram. 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] Anirban Mukherjee and Utpal Garain. *A review of methods for automatic understanding of natural language mathematical problems*. Artificial Intelligence Review, 29(2):93–122, April 2008.
- [2] Anirban Mukherjee and Utpal Garain. *Automatic diagram drawing based on natural language text understanding*. In: Proc. of the 5th international conference on Diagrammatic Representation and Inference (DIAGRAM), 398–400, 2008.
- [3] Anirban Mukherjee, Utpal Garain, and Mita Nasipuri. *On construction of a GeometryNet*. In: Proc. of 25th IASTED International Multi-Conference: artificial intelligence and applications (AIAP), 530-536, 2007.
- [4] C. Baral, M. Gonzalez, J. Dzifcak, and J. Zhou. *Using inverse lambda and generalization to translate english to formal languages*. In: Proc. of Int. Conf. on Computational Semantics, Oxford, England, January 2011.
- [5] C. Fellbaum. *WordNet: an electronic lexical database*. MIT Press, 1998.
- [6] Ch. Kop and H.C. Mayr. *Mapping Functional Requirements: From Natural Language to Conceptual Schemata*, In: Proc. of 6th Int. Conf. on Software Engineering Advances (ICSEA), Cambridge, USA, 2002.
- [7] Daniel G. Bobrow. *Natural Language Input for a Computer Problem Solving System*. PhD thesis, Department of Mathematics, MIT, Cambridge, 1964.
- [8] Eric Brill. *A simple rule-based part of speech tagger*. HLT '91: Proceedings of the workshop on Speech and Natural Language, Morristown, NJ, USA: Association for Computational Linguistics, 112116, 1992.
- [9] Gordon 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 and R.P. van de Riet. *Analyzing Informal Requirements Specifications: A First Step towards Conceptual Modeling*, In: 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, and L. Ureel. *Integrating Natural Language, Knowledge Representation and Reasoning, and Analogical Processing to Learn by Reading*. In: Proc. of AAAI Conf. on Artificial Intelligence (AAAI), Vancouver, BC, 2007.
- [12] Nhon Do, Hoai Phan Truong, and Tuyen Trong Tranan. *Approach for translating mathematics problems in natural language to specification language COKB of intelligent education software*. In: Int. Conf. on Artificial Intelligence and Education (ICAIE), 324 - 330, 2010.
- [13] Magda Ilieva and Harold Boley. *Representing Textual Requirements as Graphical Natural Language for UML Diagram Generation*. In: Proc. of 20th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE), San Francisco, CA, USA, 478-483 , 2008.
- [14] Vinay K. Chaudhri , Bert Bredeweg , Richard Fikes , Sheila McIlraith, and Michael P. Wellman. *A Categorization of KR&R Methods for Requirement Analysis of a Query Answering Knowledge Base*. In: Proc. of 6th Int. Conf. on Formal Ontology in Information Systems (FOIS), 158-171, 2010.
- [15] W. Lu, H.T. Ng, W.S. Lee, and L.S. Zettlemoyer. *A generative model for parsing natural language to meaning representations*. In: Proc. of Empirical Methods in Natural Language Processing (EMNLP), 2008.
- [16] Wing-Kwong Wong, Sheng-Cheng Hsu, Shih-Hung Wu, Cheng-Wei Lee, and Wen-Lian Hsu. *Lim-g: Learner-initiating instruction model based on cognitive knowledge for geometry word problem comprehension*. Computers and Education., 48(4):582–601, May 2007.