Automatic Diagram Drawing based on Natural Language Text Understanding

ANIRBAN MUKHERJEE

Thesis submitted for the fulfillment of
Degree of Doctor of Philosophy in Engineering
in the Faculty of Engineering and Technology

Indian Institute of Engineering, Science and Technology, Shibpur
2014

Purabi Das School of Information Technology
Indian Institute of Engineering, Science and Technology, Shibpur
(Formerly Bengal Engineering and Science University, Shibpur)
Shibpur, Howrah, West Bengal, India

CERTIFICATE FROM THE SUPERVISORS

This is to certify that the thesis entitled **Automatic Diagram Drawing based on Natural Language Text Understanding**, submitted by Mr. Anirban Mukherjee who got his name registered on November 2009 for the award of PhD (Engineering) degree of Indian Institute of Engineering, Science and Technology, Shibpur is absolutely based upon his own work under our supervision and that neither this thesis nor any part of it has been submitted for any degree/diploma or any other academic award anywhere before.

(Dr. Utpal Garain) (Dr. Arindam Biswas)

Dedicated to my beloved father-

to fulfill his last wishes...

Acknowledgment

At the onset, I must take this opportunity to express my respect and deep sense of gratitude to my supervisor Dr. Utpal Garain (Associate Professor, ISI) who accepted me as a PhD student at my very first approach and guided and inspired me throughout as a teacher and an elder brother. It was he who gave the brilliant idea of automating text to diagram conversion which is the basic theme of this research work. Without his constant monitoring, uncompromising concern for quality of work, competent advices, and patient consideration of all deviations owing to my preoccupation in service, this thesis would not have materialized.

I am extremely thankful to Dr. Arindam Biswas (HOD, Dept. of IT and Director, PDSIT, IIEST) who so very kindly accepted to act as my supervisor at a difficult juncture of my research career. His role as a stupendous mentor in detecting and rectifying faults in my efforts should never remain unappraised. I gratefully acknowledge the contribution of Prof. S S Barat (former Director, PDSIT, IIEST), Dr. H Rahman (Professor, Dept. of IT, IIEST), Dr. Jaya Deb Roy (founder Principal, RCCIIT), Dr. Tirtha Sankar Das (Associate Professor, Dept. of ECE, RCCIIT), Late Mr. P K Sen (former HOD, Dept. of IT, RCCIIT), Mrs. Ritabari Roychowdhury (former HOD, Dept. of IT, RCCIIT), Dr. Siddhartha Bhattacharyya (HOD, Dept. of IT, RCCIIT) and my friend Thakurprasad Bhattacharyya (former TPO, RCCIIT) for their support, encouragement, and helpful suggestions.

I owe a lot to my students of RCCIIT, namely Pratiti, Anutosh, Srikanta, Kiran, Souvik, Dipanjan, Sarbartha, and Anirban who worked with me for developing and testing different modules of the geometry prototype.

The invitation received by Dr. Utpal Garain and myself from Microsoft Research India (MSR), Bangalore to present our research work there and the interest shown in our system by Dr. A Kumaran, Monojit Choudhury, and Kalika Bali of the Multilingual Systems (MLS) Group encouraged me a lot. The discussions hinted future research possibilities in collaboration with MSR. I must convey my special thanks to Dr. Garain and Monojit for providing me this opportunity. I should express my gratitude to Mr. R N Lahiri (formerly Principal Consultant, TCS) *and Mr. Ajoyendra Mukherjee (Executive VP and Global Head, HR, TCS) for kindly approving and facilitating TCS sponsorship of my travel to Herrsching, Germany in 2008 to present one of my research papers.*

I must also mention the names of some of my colleagues of RCCIIT namely, Hrishikesh Bhaumik, Biswanath Chakraborty, Arindam Mondal, Dipankar Santra, Indrajit Pan, and Avijit Das for the very useful and timely help that I have received from them at different stages of my PhD work.

Eloquence always falls short in expressing my humble submission to my mother (Mrs. Sunanda Mukherjee) and father (Late Mr. Kalyan Kumar Mukherjee), and elder brother (Dr. Amlan Mukherjee). As like always, they have shared my moments of glory as well as failure in this journey. My father always inspired me for PhD work - this thesis would have made him happy had he not left us some months ago.

Very special thanks are due to my wife, Attreyee for her evergreen encouragement and the painstaking moral support she rendered in helping me out of all adverse situations. She has shared with me her vaulable research experiences and helped me to hold my head high and keep my heart open even in conditions of dejection. I must also thank my loving son Ritam for his invaluable cooperation and sacrifice during the entire period of my research work.

Last but not the least, I am grateful to my near and dear ones who are really concerned about me and have bestowed their blessings and best wishes for successful completion of my PhD work.

Department of Information Technology
RCC Institute of Information Technology, Kolkata
August 2014

[Anirban Mukherjee]

Abstract

In teaching-learning of many science and engineering subjects (like geometry, physics (mechanics), chemistry, civil, mechanical, electrical, electronics, etc.), often one has to understand/explain relations and configurations of different objects through diagrams representing scientific descriptions. Such descriptions are usually in natural language. A student of science can draw a scientific diagram using concepts of a given subject. No standard method exists whereby diagram can be automatically generated from text. This thesis focuses on finding a solution framework for this generic problem taking school-level (VIIIth to Xth grade) geometry word problems as a case study.

The present research has been successful in proposing a working system that accepts a geometric statement (in English) as an input and draws the representative figure as the output. This involves machine understanding of the problem-text and then automatically creating the corresponding diagram using standard graphics functions. A novel information representation scheme and a lexical resource (comprising classified knowledge about objects and concepts related to school-level geometry) have been developed to this end. The thesis reports the design, implementation, efficiency of these methods quite elaborately. The thesis presents some performance evaluation schemes that can evaluate the accuracy of the proposed system in terms of correctly interpreting a geometry word problem and converting it into a representative diagram. The evaluation finally helps in indicating the future improvement areas.

The text-to-diagram conversion framework proposed in the thesis has potential application in computer based teaching (CBT) where the system can play the role of a geometry tutor who reads a problem statement and shows how to draw the corresponding figure in a step by step manner. Another very interesting application of the automated drawing framework, in tactile medium is reported in the thesis. Tactile representation of text-converted digital diagrams are done using Braille text and are tested to be quite understandable by the blind students. Besides reporting the methodology of digital-to-Braille mapping and user evaluation at a Blind school, this thesis also provides a quantitative evaluation of the tactile ouput.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 Introduction

1.1 Introduction

A piece of text that basically describes a diagram comprising basic geometric entities appears in many branches of science like geometry, physics (mechanics), chemistry, and also in engineering branches like civil, mechanical, electrical, electronics, etc. Such texts are best understood with the help of illustrative diagrams. While solving a problem, usually, a problem statement (text) is first translated into a sketch (diagram) which visually articulates the essential problem parts; mechanical models, free-body diagrams, electrical/electronic circuits, geometry diagrams, and chemical structures are instances of such transformation. The diagram is then transformed into a set of mathematical formulae (symbols), which drive the problem solution. As far as machine drawing of scientific or engineering diagrams is concerned we find many Computer Aided Design (CAD) tools that allow us to draw through step-by-step commands and parameter values given as input.

But these tools are not intelligent enough to read and understand textual description of a diagram and automatically generate it in the computer. An automatic text-to-diagram conversion would be possible in any domain if a suitable approach can be employed that aptly analyzes and understands a piece of text describing a diagram to draw the same with basic geometric entities. This thesis focuses on finding a feasible solution of this generic problem taking school level (8th to 10th standard) geometry word problems as a case study. Such geometric problems typically describe a particular configuration of a set of basic geometric entities like line, circle, triangle, quadrilateral, etc. and respective diagrams are often needed to be drawn to illustrate and solve those problems.

The research work reported in this thesis attempts to develop a system following the human approach for drawing a geometric figure implied by a school-level geometric problem stated in natural language English. This involves machine understanding of the problem-text and then automatically creating the representative diagram by triggering relevant computer graphics functions. The proposed system uses a language comprehension module for syntactic and semantic analysis and formal representation of a problem-text. It also uses a custom knowledge base termed as *GeometryNet* [11], comprising classified knowledge about objects and concepts related to school-level geometry, for geometric interpretation of a problem.

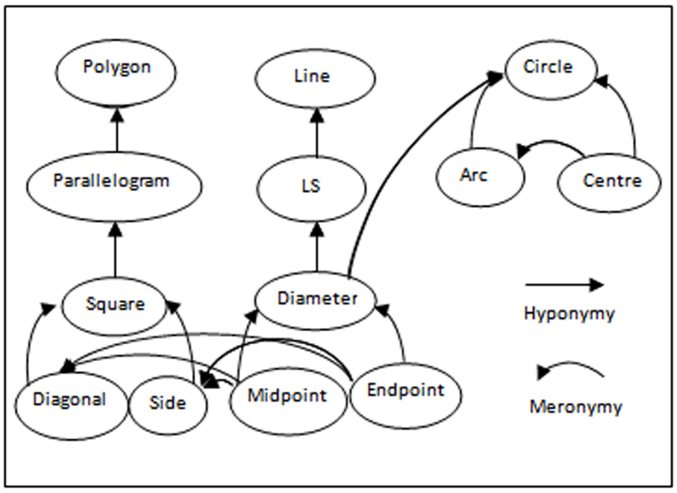
Interestingly, our research is only a focused subset of a larger research domain of automated solution of natural language scientific problems. Right from the early sixties or even earlier this had been a distinct subject of research resulting in many intelligent systems that can analyze, understand, and solve scientific problems stated in natural language. Though our system does not attempt to completely solve a natural language problem, the basic challenges of our research are same as the earlier ones. One is the issue of correct interpretation of a problem statement filtering out any lexical complexity and ambiguity while the other is the challenge of constructing a complete knowledge base - a combination of database and rule base for making correct inference or solution from the linguistic summary.

1.2 Scope of the Thesis

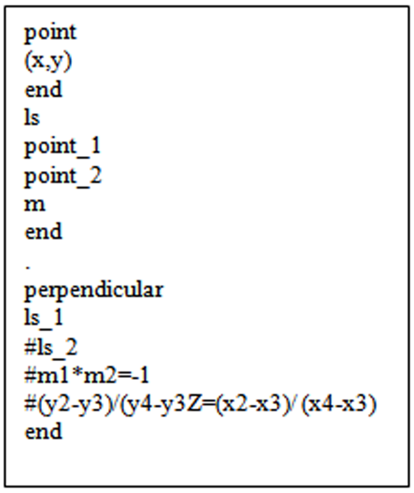
Chapter 2 contains the literature survey carried out to get a reasonable account of past and recent research in the area of automated solution of natural language scientific problems. Thirteen different research prototypes in the subject domain of algebra, arithmetic, mensuration, probability, calculus, mechanics, physics, and chemistry have been studied and analyzed to figure out different techniques used by these systems to extract information from natural language description and represent such information. It is depicted how different schemes like *paradigm*, *schema*, *proposition*, *atom*, etc. or structures like *tree*, *frame*, *meta-structure*, *ATN*, etc. or *database* or *files* are used to store information, inference rules, and knowledge. In many cases, ad-hoc and heuristic programming approach or rudimentary pattern matching technique are adopted for machine understanding that are quite useful for typical cases but fails to deal with problems of diverse nature. However, the basic architectures of all these systems are found to be grossly same. In two prototypes only, namely LIM-G [106] and ISAAC [45] we have found the utility of automatically generating representative diagrams of word problems. We have shown that researches are still continuing in the direction of translating mathematics problems in natural language to formal or specification language, using lexical resource as knowledge base in designing natural language understanding model, parsing natural language sentences for hierarchical representation of meaning, etc.

The survey also includes a comprehensive study of available devices, tools, and research prototypes for effectively communicating graphics to the blind students. Five categories of systems are discussed namely, *tactile*, *audio-tactile*, *audio-haptic*, *3D audio*, and *multimodal*. Finally it shows that most of the sophisticated computer-aided systems for communicating diagrams to the blind students are prohibitively expensive for use in the resource-constrained blind schools of developing countries like India and not so effective for learning diagram-based subjects in a classroom. The survey, however, does not find any intelligent system that can produce geometry diagrams automatically from textual description either for the use of sighted user or for the blind users.

Chapter 3 presents the details regarding the knowledge base *GeometryNet* [11] which is the backbone of the text-to-diagram (hereinafter, at times referred as t2d) conversion system described in this thesis. Basically it builds a domain specific ontology for geometrical terminology, where the semantics of certain terms are finally expressed in terms of equations involving their arguments. *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. While discussing the construction of *GeometryNet* the reference to the standard lexical resource *WordNet* [38] is drawn to explain the similarity in organization of different syntactic categories. For example, *hyponymy* and *meronymy* relations used for nouns in *WordNet* are used for depicting hierarchical and part-whole relations of geometric entities in *GeometryNet* as shown in Figure 1.1(a).



(a)



```
point
(x,y)
end
ls
point_1
point_2
m
end
.
perpendicular
ls_1
#ls_2
#m1*m2=-1
#(y2-y3)/(y4-y3Z=(x2-x3)/ (x4-x3)
end
```

(b)

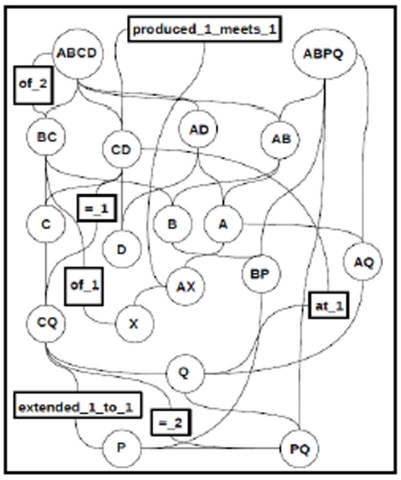
Figure 1.1: (a) Network representation of a set of hyponymy and meronymy relation in *GeometryNet*, (b) Listing of nouns *point, ls* and adjective *perpendicular* in *GeometryNet*

One of the salient features of *GeometryNet* structure is that there is natural inheritance of properties of the *subordinate* entities from the *superordinate* entities. Instead of listing common properties redundantly with both *superordinate* and *subordinate* they are listed only with the *superordinate*. Any geometrical term can be completely defined by some parameters (e.g. point coordinates, slope, radius, etc.) and some mathematical relations or equations involving the parameters (refer Figure 1.1(b)). Also, default numerical values are stored against each defining parameter of all entities. Examples of many *GeometryNet* terms under each category of *noun*, *adjective*, and *verb* are explained with illustrations in this chapter. Other issues behind design/construction of *GeometryNet* and its usage are also discussed. Finally, hint is given on how the knowledge base can be extended in domains other than geometry.

Chapter 4 describes the working of the *parser* belonging to the *Natural Language Processing (NLP)* module [8] which was most difficult to generalize and hence is the most important part of our system. The algorithmic approach and execution steps of syntactic and semantic analysis of input problem text and subsequent language-independent representation of useful information extracted from the problem is explained in details with suitable illustration. The input to this module is a school level geometry word problem stated in English. A *parts-of-speech (POS) tagger* is first used to tag the problem statements. A *parser* works on the tagged output and lists the proper nouns (NNP), corresponding common nouns (NN), and attributes if any of the NNPs in a tabular form. NN types are also assigned to those NNPs for which no NN is mentioned in the problem text. All the NNPs are then decomposed into basic entities, e.g. parallelograms are decomposed into their sides and the sides into the vertices. A *parse graph* is generated with the NNPs (parent and decomposed) as nodes. Nodes are connected using parent-child relationship. Each node contains the NN of each NNP and other special attributes (if any). The *parse graph* formed at this stage can be disconnected as it still does not contain information about relationships among entities, (e.g. X is the midpoint of BC).

So a rule based approach is followed to find the *connectors* (e.g. produced, meet, has, midpoint, etc.) establishing relations between a pair of entities and include the *connectors* in the graph. Thus a single *connected graph* (e.g., Figure 1.2(a)) is formed with new nodes and appropriate edges being created from the *connectors*. A *translator* is designed that works on the *connected parse graph* to translate it into a *structured summary* or *formal representation* (e.g. Figure 1.2(b)) as the final output of the NLP module. The experimental results on a representative problem set are cited here. Finally the evaluation of the module is done and an error analysis is included that shows the typical cases where the module fails.

It is important to mention here that full-fledged NL analysis has not been done as it was not an essential requirement in the context of extracting only relevant geometrical information from a word problem. Though a standard *POS tagger* has been used, no formal parsing technique is applied. A rule based language parsing algorithm has been developed by us that works fine in the limited domain of school-level geometry word problems. Again a rule based approach is made to resolve minimal lexical, structural and reference ambiguities occurring while handling geometry statements. Adoption of canonical form of entity names proved to be useful in resolving reference ambiguities. However, word-sense disambiguation in its true NLP sense could be avoided without any significant impact on the final output. In our approach, the knowledge base has a greater role to play than the NLP module in interpreting a problem geometrically correctly. So the NLP module has been kept simple yet effective to optimally support the purpose of automated diagram drawing from a word problem. That is why, rather than evaluating the language phenomena the module has correctly handled, it is more important to judge whether the entity connector table and for that matter the connected parse graph is built properly that would result in correct/incorrect drawing. Thus the module is evaluated by manually checking the correctness of parse graph and intermediate representation for 40 test problems.



(a)

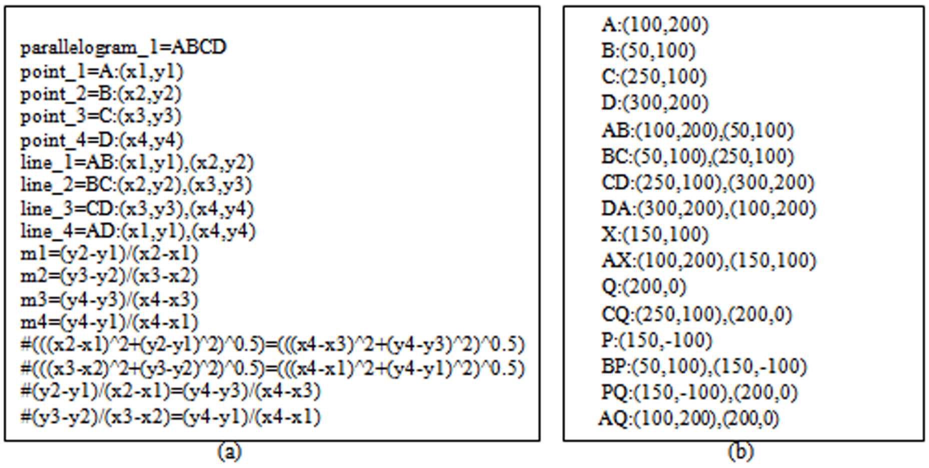
parallelogram_1=ABCD
line_1=AB
line_2=BC
line_3=CD
line_4=AD
point_1=A
point_2=B
point_3=C
point_4=D
midpoint_1=X
line_5=AX
point_5=Q
line_6=CQ
point_6=P

parallelogram_2=ABPQ
line_7=BP
line_8=PQ
line_9=AQ
(midpoint_1=X).(line_2=BC)
(line_2=BC).(parallelogram_1= ABCD)
produced_1_meets_1(line_3=CD,
line_5=AX)
at_1(line_3=CD,point_5=Q)
extended_1_to_1(line_6=CQ,
point_6=P)
=_1(line_3=CD,line_6=CQ)
=_2(line_6=CQ,line_8=PQ)

(b)

Figure 1.2: (a) The *connected parse graph*, (b) The formal *intermediate representation* for the word problem: *ABCD is a parallelogram and X is the midpoint of BC. The line AX produced meets DC produced at Q. CQ is extended to P and the parallelogram ABPQ is completed. Prove that DC = CQ = QP*

Chapter 5 explains how a geometry problem is converted to the representative diagram through intermediate steps of machine understanding of the NLP output [8] with the help of *GeometryNet* [11] and then forming graphics-friendly representation of the problem. All the steps are elaborated through an example. A second *parser* program [7] mines relevant knowledge from *GeometryNet* about the terms featuring in the final output of the NLP module (as shown in Figure 1.2(b)). It scans the NLP output line by line - whenever a statement in the form `NN1_1=NNP1` (e.g. `parallelogram_1=ABCD`) is found, it passes the basic NN (e.g. `parallelogram`) as argument of a recursive function which then returns all the child entities and expressions listed against that NN in *GeometryNet*. If the returned value is anything other than point or an expression e.g. vertex or side, those are subsequently sent as arguments of the function. Thus search for *parallelogram* triggers sequential searching of knowledge about *vertex*, *point*, *side*, and *line* in *GeometryNet*. Dynamic correlation is made between the generic information (e.g. `point_1=(x1,y1)`) extracted from *GeometryNet* for each constituent point corresponding to the NN (e.g. `parallelogram_1`) searched for and the relationship already stored in the *connected graph* for the respective NNP (e.g. `ABCD`). This results in assignments like `point_1=A:(x1,y1)`. Thus we get the generic information (geometric meaning) about all the entities and relations pertaining to a problem in terms of variables and equations as shown in Figure 1.3(a) (for parallelogram).



```
parallelogram_1=ABCD
point_1=A:(x1,y1)
point_2=B:(x2,y2)
point_3=C:(x3,y3)
point_4=D:(x4,y4)
line_1=AB:(x1,y1),(x2,y2)
line_2=BC:(x2,y2),(x3,y3)
line_3=CD:(x3,y3),(x4,y4)
line_4=AD:(x1,y1),(x4,y4)
m1=(y2-y1)/(x2-x1)
m2=(y3-y2)/(x3-x2)
m3=(y4-y3)/(x4-x3)
m4=(y4-y1)/(x4-x1)
#(((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)
#(y2-y1)/(x2-x1)=(y4-y3)/(x4-x3)
#(y3-y2)/(x3-x2)=(y4-y1)/(x4-x1)
```
(a)

```
A:(100,200)
B:(50,100)
C:(250,100)
D:(300,200)
AB:(100,200),(50,100)
BC:(50,100),(250,100)
CD:(250,100),(300,200)
DA:(300,200),(100,200)
X:(150,100)
AX:(100,200),(150,100)
Q:(200,0)
CQ:(250,100),(200,0)
P:(150,-100)
BP:(50,100),(150,-100)
PQ:(150,-100),(200,0)
AQ:(100,200),(200,0)
```
(b)

Figure 1.3: (a) Instantiated form of the information mined against *parallelogram ABCD* from *GeometryNet*, (b) Draw-able representation of the problem cited in Figure 1.2

Next, these variables are suitably instantiated with numerical values to form a drawable representation [7] of the problem as shown in Figure 1.3(b). In the process either default numerical values stored in *GeometryNet* are used or constraint equations are solved to evaluate the variables. A graphic module then invokes standard line draw and circle draw functions for all the lines and circles in the said representation and passes respective set of coordinate values as arguments of the functions. As a result, the diagrams representing the problem takes shape (Figure 1.4) in computer display as all the entities are drawn sequentially.

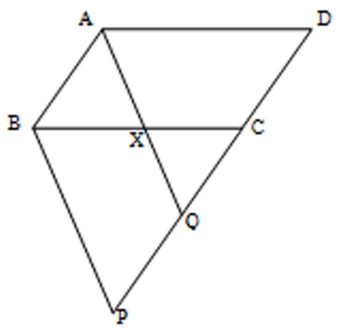


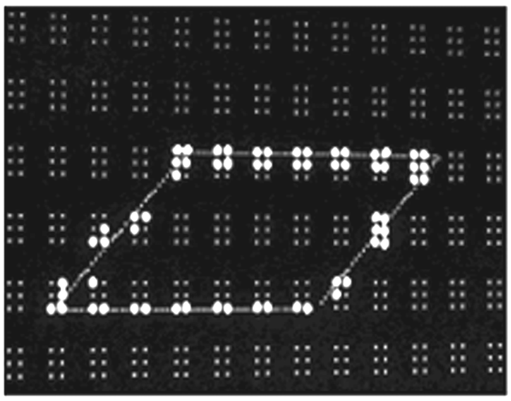
Figure 1.4: The computer-generated diagram of the parallelogram problem in Figure 1.2

The correctness of the diagrams automatically drawn from a test set of around 40 school-level geometry problems are evaluated algorithmically by matching with corresponding groundtruth diagrams created using CAD software. Two methods are described - one based on physical attribute matching considering base entities drawn in CAD with same default parameter values as stored in *GeometryNet*; the evaluation program counts the number of correctly drawn (perfectly matched) entities, the number of missed entities and the number of false entities drawn and formulates the performance factor for a given diagram. The other method [10] is based on matching geometric properties of system-drawn and groundtruth diagrams considering the similarity of adjacency matrices and number and values of nodes, internal angles, and edges. It then calculates the accuracy percentage with some degree of probability. Experimental results show that if the linguistic analysis is correct, the diagrams are mostly drawn correctly.

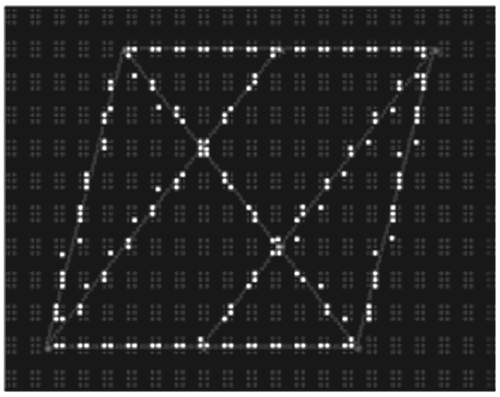
The evaluation of the diagram drawing module gives a measure of the quantum of geometry phenomena our system is able to handle and represent properly. The present scope of work encompasses the geometrical concepts pertaining to the domain of school level Euclidean geometry. Considering many possible ways of graphically representing a particular geometry phenomena described through a word problem, the second method of evaluation more realistically tracks the geometrical correctness of a diagram. Though the drawing module can handle straight line, circle and for that matter any possible combination of straight line and circle making complex geometric configurations, the second method of evaluation cannot properly handle diagrams containing circles. The scope of evaluating our work is thus limited to only straight line geometry figures in 2 dimension.

Chapter 6 is about our system's extension to a Braille based application. This application is meant to draw geometric figures automatically from textual description of geometry problems in the tactile form understood by the blind people. This chapter discusses the Braille NUMBRL system [15] and presents how the existing algorithms for digitization of basic geometric shapes are modified to work for a low-cost Braille printer (meant to print text only) commonly used in resource-constrained Blind schools in countries like India.

To demonstrate the working of the Braille module, a computer simulation of Braille character mapping of geometric entities has been done [9]. An array of Braille cells (each having 6 dots) is displayed in the computer to represent a page of printed Braille characters. The numerical values of parameters of all drawing entities (lines, circles, etc.) are read sequentially from the draw-able representation (as shown in Figure 1.3(b)) of a geometry word problem and passed on as input to the Braille line and circle drawing functions. These functions find Braille dots (pixels) closest to the respective line or circle path. All such selected points are marked with thicker dots to emulate embossing by Braille text printer. By additive combination of position values, the NUMBRL code is evaluated for a group of selected dots which belong to a single cell. The NUMBRL codes are stored row-wise in a text file. One row of NUMBRL codes in the file correspond to one print line by Braille text printer. As the printer prints all the coded characters of the output file row-wise, the diagram implied by the input geometry problem takes shape as shown in Figure 1.5



(a)



(b)

Figure 1.5: (a) Emulated Braille diagram of a parallelogram - the first row of Braille data in the format (column no., row no., NUMBRL code) corresponding to the top line of the parallelogram is: (5,3,73),(6,3,33),(7,3,33),(8,3,33),(9,3,33),(10,3,33),(11,3,77) (b) Emulated Braille diagram for a Xth grade geometry problem

An empirical method has been proposed to evaluate line and circle represented in Braille by quantifying the approximation errors. Comparison of diagram errors and accuracy of the digital to Braille mapping system with similar other systems have also been discussed in this module.

Probably the most interesting part of the thesis is the section of chapter 6 that describes the user study and evaluation (of the Braille drawing utility) [12] carried out with blind students at a blind school in Kolkata, India. Different aspects of user training and experiments, analysis of pre and post experiment feedback and results have been elaborately presented in this chapter.

Chapter 7 is the concluding chapter of our thesis. It reviews the work done so far and attempts to estimate our achievement against our initial goals. Our journey for last 6-7 years through different research experiences is briefly recollected. Finally an incisive discussion is included on the possible future extension of this research work.

1.3 Motivation

Developing intelligent systems for understanding and solving natural language scientific problems is quite challenging. It focuses on a problem which is different from one that is handled while developing a menu or command driven software to solve computation-oriented scientific problems. Such software have a control over the user-input through pre-designed structured input mechanism whereas the intelligent systems are supposed to accept natural language description of a problem that can be stated in many different ways by different users. A natural language based system [24] should use Artificial Intelligence to understand the problem out of all lexical complexity, interpret it in terms of known concepts and then apply the most appropriate solution logic. On the other hand, software packages usually follow a processing pattern which is mostly mechanical and hence, do not require any language comprehension module or domain knowledge to solve a problem. The question of misinterpreting a problem does not arise for such programs whereas the natural language problem solvers run the risk of giving erroneous results for misinterpretation or lack of knowledge for a particular problem or case.

Ideally a natural language scientific problem solver should use the information presented in the problem itself to figure out the mathematical relationships among the elements and use some intelligence to propose the solution. The challenge lies in avoiding the rather easier approach of finding potential answers stored for matching cases. One more difficulty in developing such intelligent systems is to map an optimum set of rules out of a wide variety of available rules or techniques to solve all types of problems pertaining to any scientific domain and thus make the system robust for a given domain.

In the literature survey presented in Chapter 2 it is shown how different natural language problem solvers have negotiated the above challenges to different extent. Some have used pattern matching techniques, some used heuristic programming approach, some were limited within predefined knowledge or input or rule base, etc. But hardly any system is found that is ready to accept all types of problems within its domain and deal reasonably with linguistic ambiguity and also extraneous or missing information naturally carried by a word problem. As such, no system is found that can automatically generate geometry diagrams (with any combination of basic geometric entities) from natural language description.[1]. Software capable of dynamically generating diagrams to represent any scientific description or word problem can make computer-aided teaching-learning far more interesting than now is.

The above stated facts and challenges of developing such an intelligent system have been our motivation to undertake the research. We always believed if diagrams can be generated from school level geometry word problems, then diagrams (comprising basic geometric entities) in other subject domain can also be generated from corresponding descriptions. Now that there has been significant development in the field of natural language processing and knowledge engineering, we dared to attempt this problem. The popular lexical database *WordNet* [38] often used for language processing task, provided us with a model for developing the geometry knowledge base *GeometryNet* - a major component of our proposed system.

The literature does not report any standard mechanism for evaluation of output (mathematical solution or diagrams) produced by different natural language problem solvers. These systems are reported to be performing reasonably good with limited test set. But it is not tested how those systems perform with more complicated problems involving more number of objects. There is no ready reference of accuracy rate of text-to-diagram conversion - a conceptually correct diagram (produced from a problem) can be detected by a subject expert but partially correct output cannot be quantified by observation only. This gap in research generated our motivation for proposing a suitable scheme for quantitative evaluation of geometry diagrams produced by our proposed system.

Many Computer-Aided Instruction (CAI) systems or tutoring software that are used for Computer Based Teaching (CBT) or learning have predefined strategy of instructing the learner on solution methods of model problems stored in the database. But those are not intelligent enough to automatically comprehend, solve, and demonstrate new

[1]LIM-G and ISAAC can generate diagrams but that too only few predefined simple figures. They cannot deal with problems which involve complicated geometric shapes.

problems given by the learner. Intelligent tutoring demands such learner-initiated CAI systems that can deal with unknown problems stated in natural language. It should ideally be able to analyze the language and comprehend the problem in technical terms and accordingly provide the learner necessary information, diagrams, and incremental guidance for solution of a problem. The proposed t2d conversion framework has the potential to meet some of these requirements. This motivated us towards applying the framework in student self-control CBT software (Geometry Tutor).

For blind students, understanding of diagram-based subjects is often difficult if they cannot perceive diagrams along with the text. As a matter of fact, it is a problem for teaching any science subject to the blind students in India. Illustrative figures for elementary topics or problems may be available in embossed form in text books or study materials. But it has some obvious limitations. Firstly, producing tactile versions of large number of figures for books is a time consuming, labour-intensive, and costly process. Secondly, such static text book figures cannot be as effective (for the blind students) as the diagrams which are drawn on blackboard/paper by a teacher while explaining any subject/problem to the sighted students. There are few sophisticated interfaces and graphics programs for instant desktop production of tactile diagrams but such high-end costly systems are not widely available in developing countries like India.

Thus blind students in such countries often grow up without any exposure to modern learning aids for diagram-based subjects like geometry. Historically, teaching geometry to these students is limited to giving basic theoretical definitions only and application oriented exercises requiring frequent diagram drawing are deliberately avoided. The lack of access to figures results in students memorizing facts as verbal assertions, without developing true scientific skill. Motivated by the need of developing an assistive technology to support science education for the blind pupil at school level at affordable cost, we have further extended our research work. An automatic t2d conversion framework that we developed for the sighted users is improvised for the blind users - here the final form (hardcopy) of drawing is made using low-cost braille printer (meant only for printing braille text) that is available in most blind schools of India.

1.4 Contribution

Though several efforts have previously been reported in the literature on understanding natural language scientific problems, focused research on automatic t2d conversion as reported in this thesis could be considered as a novel study.

The system proposed in the thesis is unlike usual CAD software that produces geometric diagrams upon feeding parameter values of geometric entities against structured input prompts generated by the software according to predefined formulae/algorithm. It is intelligent enough to figure out the parameters of the drawing elements from the problem description upon consulting the knowledge base *GeometryNet* [11]. No such geometry knowledge base has been developed before. It has been able to overcome the limitation of rudimentary pattern matching technique which had been the basis of working of most of the natural language problem solvers developed before. Thus the proposed system is flexible enough to accept any school level geometry word problem as input and draw any complex combination of geometric entities described in the problem.

Unlike earlier developed natural language problem solvers which had no provision for measurement of correctness of output, we have successfully applied two methods of quantitative evaluation of diagrams produced by our prototype. One of these methods is an algorithmic approach - mostly an adoption of some recent work on evaluating graphics recognition systems (GRS). Though evaluation of the present system is much more critical as it measures how much correct a diagram is that is created from natural language description and knowledge base rather than merely been reproduced from one existing format to another (as done by GRS). Here the system may fail to draw correctly for any of the following reasons - linguistic ambiguity, insufficient/ redundant information in the problem description, inadequate information in the knowledgebase or incorrect derivation of variables pertaining to certain geometric relation(s).

The second evaluation approach reported in the thesis is a novel one - though it has some distinct limitations, it is more fundamental and can judge the geometric correctness of a diagram just the way a teacher checks a diagram drawn (practically at any size and orientation) on paper by a student. The method is based on graph adjacency matrices and geometric properties of basic entities unlike physical properties that are considered in the first method. This problem of evaluating conceptual correctness of diagrams drawn by text-to-graphics conversion systems is a problem never dealt earlier. Even though our solution is not robust, it puts a step forward in the future direction of research in this field.

Upon analysis we find that inaccuracy of output of our proposed system is mostly due to error in handling the language of the geometry problems in some typical cases. No standard parsing approach has been adopted, but still one of the salient contributions of our research is the module developed for semantic analysis of a problem description (in English) and its subsequent formal representation in the form of a graph and summary. In designing the NLP module we were inspired by the idea that an intelligent system should process information in a way which resembles human perception closely. It should not keep itself confined to the parsing and comprehension of a few built-in text forms as the earlier system used to do. We think this research is reasonably successful in this respect.

The novel text-to-diagram conversion mechanism proposed in the thesis establishes a basic framework towards developing intelligent Computer Based Teaching (CBT) tools for school-level geometry problems [6]. The tool can work on new problems given by the user unlike existing tutoring software that can demonstrate only model problems/questions stored in the database; it can help in step-by-step drawing of geometric elements corresponding to the input problem statements so as to guide a student in comprehending a problem. This is same as the manual approach of drawing parts of a diagram while one reads a problem line-by-line and tries to understand or explain it.

The same utility for intelligent geometry tutoring is extended for the blind users [12]. Keeping the basic mechanism same we have suitably modified the graphics routines to map the automated drawing output to the commonest braille printing system [9] so as to make the drawing accessible and affordable to the blind users. Here the main challenge was the improvisation of the Braille printer, which is usually used to print alphanumeric characters, to represent geometry diagrams comprising many shapes and configurations. Earlier, only drawing of triangular shapes was attempted [3] in Braille.

With adequate training to the users, the proposed system can be used as a self-learning aid whereby blind students themselves, without assistance of any sighted person, can create and access the geometry diagrams through tactile interleaved text and graphics mode. Though an empirical evaluation of Braille-mapped diagrams suggest scope of further improvement of the system, such automated text-to-diagram representation using Braille text is the first attempt ever.

One very important contribution of this research is the series of trainings and field tests conducted [12] with blind students and teachers belonging to a blind school and also working professionals (who are blind) on the Braille drawing system. Introducing a new learning methodology to the blind students, who are hardly conversant in diagram-based learning in schools, was really challenging. The usefulness of the methodology is clearly indicated in the overwhelming performance of the subjects within a short learning time.

In light of the above it can be concluded that the present study has established a working framework for text-to-graphics conversion in scientific domain by implementing a prototype that can draw geometry diagrams directly from geometry word problems. The design of such a framework is significant in the context of several current research areas like natural language text understanding, machine learning, artificial intelligence, etc. Moreover, as an useful extension of this work, an working scheme is proposed that is able to represent such machine-drawn diagrams using Braille text. If developed on full-scale, both the applications can be used as effective learning aids and are expected to benefit the sighted and blind student community in their science education.

CHAPTER 2 Literature Survey

2.1 State of the Art

As far as computer science is concerned the subject of our study falls in the classic domain of Artificial Intelligence (AI). The ultimate objective of research in AI is to explore the capability of a computer to act as a replacement of human intelligence and reasoning ability backed with multitude of domain knowledge. However, in comparison to the total research effort in the above direction little work has been seen in automating scientific or more specifically mathematical problem solving in a natural language environment in a way that follows usual problem solving approach by human. By the term mathematical problem we largely mean problem that involves some kind of scientific theory/rule or logic or formulae and basically requires mathematics to solve it - be it a problem of mechanics, physics, geometry, or algebra. This chapter provides a state-of-the-art technical review of earlier systems and approaches proposed by different research groups in the said area [5]. The systems are studied here in two aspects - one is language comprehension technique and the other is the problem-solving scheme. Some of the latest research in natural language understanding and knowledge representation, relevant to the context of our study, have also been included here.

2.1.1 Mathematics Problems

Algebra and Arithmetic Problems

The **STUDENT** program [27] written by Daniel G. Bobrow is the first program that reads, understands and solves an algebraic problem stated in a restricted set of English language and answers the questions pertinent to the problem in English language. For example a problem input to the program is:

```
If the number of customer Tom gets is twice the square of 20
percent of the number of advertisement he runs, and the number
of advertisements he runs is 45, what is the number of customer
Tom gets?
```

The output given by the program for this problem is:

```
The number of customer Tom gets is 162.
```

The information storage structure used in STUDENT system follows a *relational model*. The objects in the model are variables, which are nothing but the words and phrases of the input sentences around the key words. For example, `the number of customer Tom gets` is a variable where the key word is `number`. The relationships between the objects are the mathematical relations of sum, difference, product, quotient, exponentiation and equality. The media for exhibiting such relationship are a set of equations expressed in a parenthesized prefix notation, e.g. `(EQUAL (NUMBER OF ADVERTISEMENTS HE RUNS) 45)`. Using recursive use of format matching complex sentences and coordinate sentences in the problem are resolved into simple sentences free of *if*, *comma*, *and*, etc. Stage-wise transformations are then carried out with necessary substitution (e.g. *2 times* for *twice*), truncation (e.g. *square of* to *square*), etc. until these sentences themselves become variables and are related with basic operators (e.g. *plus, times, percent,* etc.) to form a set of equations representing the problem. After such transformation the above problem becomes:

```
(EQUAL X00001(NUMBER OF CUSTOMER TOM GETS))
(EQUAL (NUMBER OF ADVERTISEMENTS HE RUNS) 45)
(EQUAL (NUMBER OF CUSTOMERS TOM GETS) (TIMES 2 (EXPT
(TIMES 0.2 (NUMBER OF ADVERTISEMENTS HE RUNS)) 2)))
```

Using standard techniques of solving simultaneous equations answers to the problems are finally found. Apart from some general information like *twice always means 2 times* or *12 inches equals 1 foot*, no information is made permanent store of knowledge in the system.

Another system called **WORDPRO** [23] was developed by Charles R Fletcher for understanding and solving arithmetic word problems. WORDPRO uses a set of propositions to represent the meaning of a problem text. For example, `Joe had 3 marbles` is represented as:

```
(((P1 (EQUAL X JOE))) ((P2 (PAST P3)) ((P3 (HAVE X P4)) ((P4 (3
MARBLE))).
```

The concept of *set schema* [94] is applied as the basis of construction of problem model. Structurally, a *change schema* holds relation amongst start-set (`Joe had 3 marbles`), transfer-set (`Then Tom gave him 5 marbles`) and result-set (`How many marbles does Joe have now?`). Table 2.1 lists the instantiations of four *schemas* used in WORDPRO.

The program solves a problem guided by certain rules (thirteen *meaning postulates*, twelve *arithmetic strategies*, and eleven *problem solving procedures*) which are applied sequentially for add/change/delete actions based on the content of the short term memory (STM) of the program; the rules add information to the problem model, add new *schema* to the STM, add request for *schema* with certain specification to complete a higher-level *schemata*, and eventually finds solution value for the problem. Two other programs, namely - CHIPS [28] and ARITHPRO [25] could also solve one-step arithmetic word problems with only one possible operation addition/subtraction. The models used by CHIPS and ARITHPRO categorize simple word problems into the same three categories: *compare*, *combine* and *change*. But for both these models rigid limitations were imposed on the *change* verb (`to give`) and on the order of the problem sentences (the first sentence of the problem must describe the number of objects the owner had to begin with, whereas the second sentence must contain the verb `gave`).

Table 2.1: Schemas in WORDPRO

Change - In	`Joe had 3 marbles. Then Tom gave him 5 marbles. How many marbles does Joe have now?`
Change-Out	`Fred had 3 boxes. Then he gave 2 boxes to Susan. How many boxes does Fred have now?`
Combine	`Lucy has 3 dimes. Sarah has 6 dimes. How many dimes do they have altogether?`
Compare	`Dan has 6 books. Jill has 2 books. How many books does Dan have more than Jill?`

Yefim Bakman developed a more advanced simulation program **ROBUST** [110] that could understand free-format multi-step arithmetic word problems and that too with extraneous information [31]. An example having complex sense and also extraneous information is:

```
David gave 3 candies to Ruth, and John gave 2 candies to David.
Now David has 4 candies more than Ruth has.  How many candies
does David have now, if Ruth had 7 candies in the beginning?
```

The concept of *schema* as used in the earlier works is adopted with introduction of *change formulae* and further expansion of *change schemas* into six distinct categories (namely, *Transfer-In-Ownership*, *Transfer-Out-Ownership*, *Transfer-In-Place*, *Transfer-Out-Place*, *Creation* and *Termination*) as against only two used earlier (namely, *Change-In* and *Change-Out*). Such *change formulae* exhaustively describes a change situation, i.e. the initial number of items, number of items transferred, and final number of items depending on the type of change (i.e. transfer of items to or out from a place or person).

The ROBUST simulation works by first parsing the problem text to split all sentences into propositions or simple sentences like `David gave 3 candies to Ruth`, `David` `has ? candies` for the above problem. The propositions having complex change verbs like *give* are further split into elementary ones (like `David gave 3 candies to Ruth` is split into `David forfeited 3 candies` and `Ruth got 3 candies`). Then the *change formulae* are applied to the elementary propositions by substituting the constant values for the corresponding variables. The formula propositions are matched to their instantiations in the problem text and the relevant *schema* is attached to the *list of schema instantiations (LSI)*. The process of understanding results from the representation of the problem through the *schema* instantiations (listed in the *LSI*) of all *compare*, *combine* and *change schemas*.

Kushman et al. very recently reported an approach [78] for automatically solving algebra word problems by focusing on analyzing multiple sentences simultaneously unlike most other approaches. They have defined a two step process to map word problems to equations. First, a template is selected (using a probabilistic model) to define the structure of the equation system pertaining to a problem. Next, the equation template is instantiated with numbers and nouns (variables) from the text. This output system of equations is then automatically solved to generate the final answer. The set of equation templates are induced from the training set under semi-supervised or supervised learning scenario. The evaluation of the approach is done in terms of two metrices - equation accuracy and answer accuracy. The average accuracy measured on a corpus of 514 algebra word problems is around 70% which is a remarkably good result considering the inherent challenge of the problem dealt with.

Many computer aided instruction (CAI) systems are developed to help students learn how to solve algebra and arithmetic word problems. As for example, **Pump Algebra Tutor (PAT)** [61, 62], **WORDMATH** [22], **DISCOVER** [76], **WPS Tutor** [55] and **Math-CAL** [60]. All these systems have predefined strategy of instructing the learner on solution methods of model problems stored in the database. Such systems are not intelligent enough to comprehend and solve previously unseen problems given by the learner.

Mensuration Problems

A Learner-initiating Instruction Model, **LIM-G** [106] is a proposed CAI model that can deal with unknown mensuration problems of certain types. After comprehending any problem it can provide necessary information, diagrams and incremental guidance to a learner to solve the problem. The domain of the system is guided by problems taken from Taiwans elementary school level textbooks and are textual descriptions (without any graphical illustration) involving seven classes of geometric shapes including squares, rectangles, circles, triangles, trapezoids, parallelograms, and sectors. For example:

```
There is a rectangular land with length 1km and width 500m.
What is its area?
```

LIM-G can comprehend a new mensuration problem by extracting from it relevant information and representing it in a tree-type cognitive knowledge structure that contains hierarchical nodes of schematic knowledge, problem concepts, and linguistic knowledge. For the above problem the knowledge structure is shown in Figure 2.1 where the *problem category* is the schematic knowledge and *attributes* and *values* are linguistic knowledge.

To solve a problem, semantic knowledge like *1km = 1000m*, *area of rectangle = length * width* is also used. Using *Scalable Vector Graphics (SVG)*, a tool for graphical applications in XML, simple diagrammatic representation of a problem is made out of the *concept-attribute* content of the problem. Also the content is used by the comprehension diagnosis module to ask relevant questions to the learner to assess his comprehension level and approach for solution.

LIM-G has a cognitive knowledge base constructed with an ontology-based knowledge engineering tool called *InfoMap* [107] and containing generic template nodes for problem class, problem-concept, lexical knowledge, lexicon, etc. Using template matching mechanism the exact category of a given problem and most fitting problem-concept set is determined to finally retrieve the concepts attributes and values from a problem.

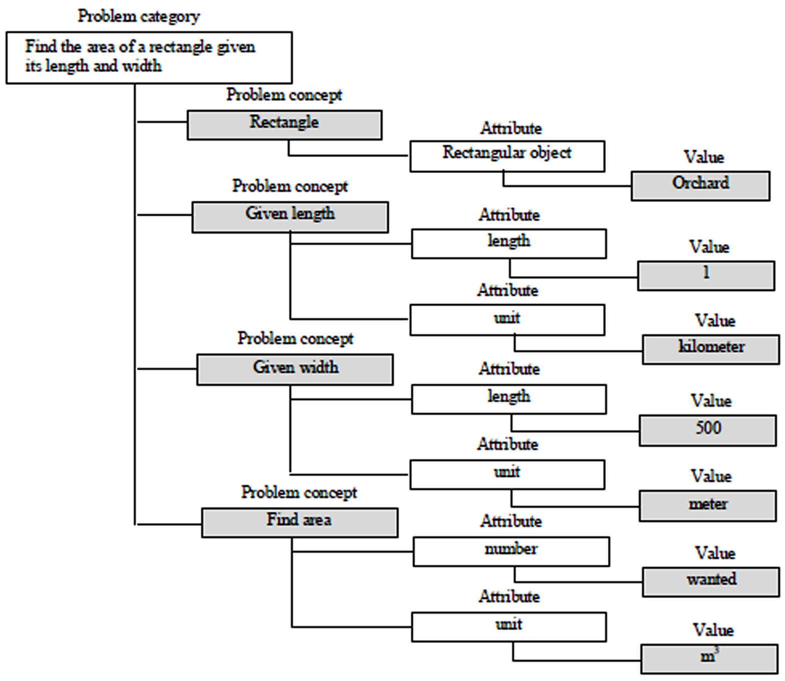


Figure 2.1: Knowledge structure used in LIM-G

Rate Problems

The program, **CARPS**, i.e. Calculus Rate Problem Solver [34, 35] written by Eugene Charniak reads, understands and solves rate problems stated in English. The program is similar to STUDENT [27], the primary difference being introduction of *structures* as the internal model of representing information extracted from a problem. The *structure* is basically a *tree* which has its head the name of an object, and at various levels beneath the head all the information the program could abstract from the problem.

The program has solved 14 rate problems pertaining to two types - distance and volume, mostly taken from standard calculus texts. For example, consider the following rate problem.

```
WATER IS FLOWING INTO A CONICAL FILTER AT THE RATE OF 15.0 CUBIC
INCHES PER SECOND. IF THE RADIUS OF THE BASE OF THE FILTER IS
5.0 INCHES AND THE ALTITUDE IS 10.0 INCHES FIND THE RATE AT
```

```
WHICH THE WATER LEVEL IS RISING WHEN THE VOLUME IS 100.0 CUBIC
INCHES.
```

Upon reading this problem the program ascertains the type (distance or volume) it belongs to and the relevant formulae stored in memory by searching for certain clues and keywords (e.g. `LEVEL` and `CONICAL`) in the sentences after parts of speech tagging of the words. Next each sentence is decomposed into simpler sentences by finding match with stored patterns of sentences. For example the second sentence of the problem is matching with the pattern `IF-ANYTHING-, -QUESTION WORD-ANYTHING` where the question word is `FIND`; the associated operation is to break the sentence into two simpler sentences namely `IF THE RADIUS....10.0 INCHES` and `FIND THE RATE....100.0 CUBIC INCHES`.

Once the sentence decomposition process is over, obtaining six simple sentences of the like of `((THE RADIUS OF THE BASE OF THE FILTER (IS VERB) 5.0 (IN UNIT))`, the transformation module translates each of these simple sentences into sub-structures by identifying the basic form of a sentence (e.g. `NVP-IS-NUMBER-UNIT`) and/or its noun phrase (e.g. `ANYTHING-OF-A / THE /PRONOUN-ANYTHING`) from a stored list. The sub-structure corresponding to the simple sentence mentioned above is the second branch from right `(FILTER-BASE:G0016-RADIUS:G0017-VALU:(5IN))` in the complete *tree structure* (Figure 2.2) constructed by coherent integration of all the sub-structures. Here `G0017`, `G0018` are symbols generated by LISP system and are known as *GENSYMS*.

Using the information gathered in the tree structure and the knowledge it stores about the word `conical`, the program finally forms three equations relevant to the problem. For instance, the equation it forms for the question sentence of the example problem is:

```
((EQUAL (G0020) (DERIV (G0019 WATER FILTER)))
```

The usual notation of this equation is:

$$\mathtt{G0020} = \frac{\mathtt{d(G0019)}}{\mathtt{dt}}$$

Here value of `G0020` will be the answer of the problem. The basic formulae retrieved from storage to form the other two equations are:

$\texttt{volume} = \texttt{1/3}\pi\texttt{(RC)}^2\texttt{(AC)}^2$

Here `RC` and `AC` are radius and altitude of water content in the conical at any time, `R` and `A` are radius and altitude of the conical.

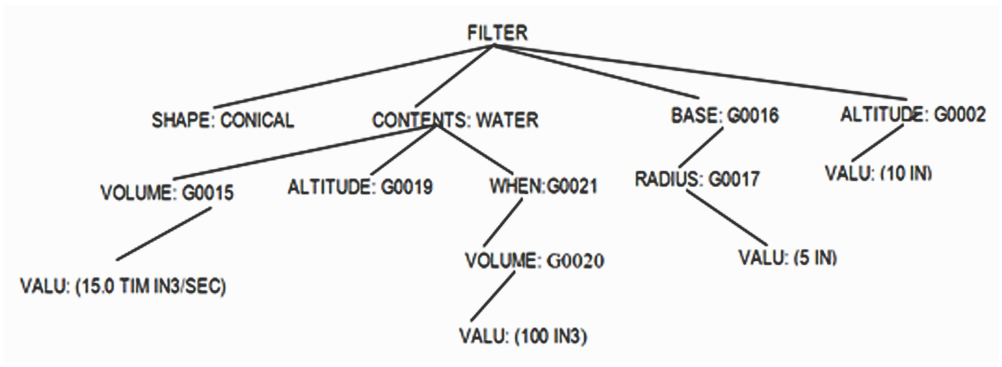


Figure 2.2: Tree structure used by CARPS for representing information extracted from a problem

Probability Problems

A program which solves basic probability problems stated in English is the **HAPPINESS** program [56, 57] written by Jack P Gelb. HAPPINESS stands for *Heuristic Analysis of Probability Problems in a Natural-language Environment with Symbolic Solutions*. Given an input problem,

`WHAT IS THE PROBABITY OF GETTING TWO OR MORE HEADS OR EXACTLY 3 TAILS WHEN FOUR COINS ARE TOSSED ONCE?`,

the output given by the program is: $\frac{15}{16}$ or 0.9375.

HAPPINESS is similar to Charniaks CARPS; it builds a tree structure or descriptor-list (*desclist*) representing a problem and selects a solution method based on the occurrence of keywords (e.g. `dice`, `cards`, `coins`) in the problem statement. The language analyzer module decomposes the input sentences into simple clauses and phrases, e.g. `((TOSS/PASSIVE) (4 COINS) (1 TIMES))` by some pattern matching. Next, semantic scan (finding keywords as `coin`, `distribute`, etc. to ascertain the type of problem) and subsequently syntax analysis is done on these simple clauses using a context free grammar. The result is a symbolic description of the simple probability events described in the problem, together with a specification of the combination of these events required to solve the problem. This information, combined with the results of the semantic scan, constitute the *desclist* of the problem as shown in Figure 2.3.

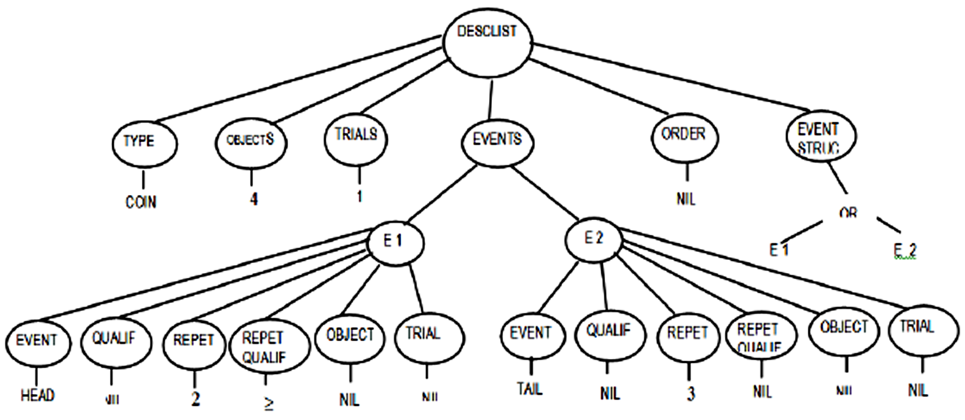


Figure 2.3: *Desclist* as used by HAPPINESS for the example problem

Using the *desclist* information the solution generator module then produces the solution, first by producing symbolic result (e.g. P(E1) + P(E2), E1: 2 heads, E2: 3 tails) followed by expansion of symbolic probability terms entailing permutation, combination, fraction and arithmetic operation. For example, P(E2) is expressed as:$(\frac{4}{3})(\frac{1}{2})^3(\frac{1}{2})^1$ and P(E1) is expressed as: $(\frac{4}{2})(\frac{1}{2})^2(\frac{1}{2})^2 + (\frac{4}{3})(\frac{1}{2})^3(\frac{1}{2})^1 + (\frac{4}{4})(\frac{1}{2})^4(\frac{1}{2})^0$, and finally the numeric result is $\frac{15}{16}$ or 0.9375. Heuristics are applied when the *desclist* information is insufficient or incomplete to expand the symbolic result.

2.1.2 Physics Problems

Mechanics Problems

The **NEWTON** program [52, 53] written by Johan de Kleer is an expert problem solver in the domain of mechanics, specifically relating to kinematics of objects moving on surface. By employing different representation and reasoning techniques NEWTON demonstrates how to solve simpler problems through simpler steps or techniques than those required for complex problems.

To solve a kinematics problem NEWTON first employs qualitative knowledge or arguments to predict the sequence of possible path/position of object movement and also different states of the object at different times. This prediction process is termed as *envisioning*which, for example, tells without much calculation that an unsupported object will fall. The system extracts the gross kinematics features from a problem and represents the envisioned path/positions in a tree-type structure, each node of the tree being a possible position of the object. For instance, consider the *envisioning tree* (Figure 2.4(b)) that may be constructed to qualitatively represent the following problem.

```
Whether the ball rolling along the frictionless curved surface
could reach point B starting from point A?
```

The problem is represented in Figure 2.4(a). The tree describes that the object starts at point A, slides through segment S1, reaches point C, slides through segment S2, either slides back on segment S2 or reaches point B and so forth. So the possible answers to the above problem can be predicted by tracing the tree only and without necessitating any numerical analysis by using rules of conservation of energy, etc. But in order to answer more detailed questions (referring the same problem) like `What will be the velocity of the ball at point B?` quantitative knowledge like heights and distances of points `A` and `B`, other relevant variables and constants and basic kinematics rules/formulae are required.

Quantitative knowledge is organized in terms of *FRAME* structure, which packages together mathematical equations involving the variables, like the following one.

```
FRAME energy OF object surface t1 t2
VARIABLES:
(vi: VELOCITY OF object AT TIME t1,
 vf: VELOCITY OF object AT TIME t2,
  h: HEIGHT OF surface)
vf^2 - vi^2 = 2Gh
```

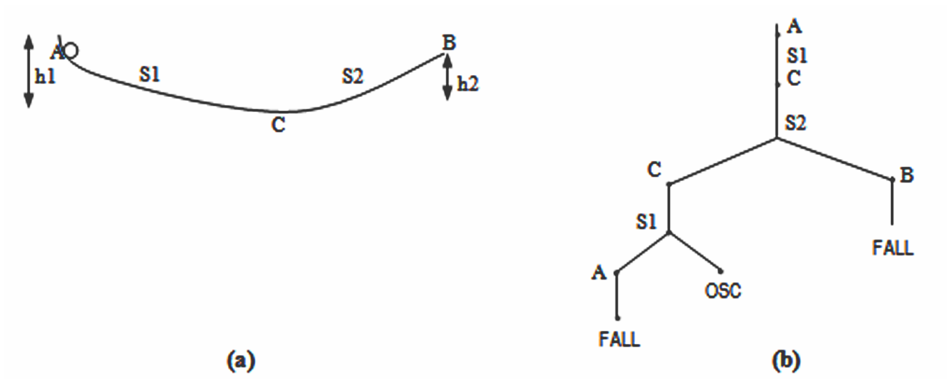


Figure 2.4: (a) The ball rolls along path A-C-B, (b) Envisioning tree for the rolling ball problem

NEWTON maintains a database to represent the semantics of variables [30]. On carefully analyzing the *envisioning tree* the relevant *FRAMES* are invoked, a subset of which is instantiated with variables and values taken from the problem and subsequently examined for solution.

The program **MECHO** developed in Prolog by Alan Bundy and group [1] solves mechanics problems stated in English in the areas of pulley problems, statics problems, motion on smooth complex paths and motion under constant acceleration. The underlying focus is to get a formal representation of a problem from an English statement and to use this representation as well as an expert inferential system to solve the problem. From input text to the final solution, the program works through the following stages (shown in boxes in Figure 2.5) using various types of knowledge input (shown as labels of arrows in Figure 2.5).

Upon parsing a sentence, a set of assertions are produced (by instantiating out the referents) about the objects in the problem. Consider an example problem statement:

```
Two particles of mass m1 and m2 are connected by a light string
passing over a smooth pulley.
```

Some of the assertions produced for the above statement are:

```
isa(particle,p1), isa(pulley,pull), coeff(pull,zero),
mass(p1,mass1,period1), measure(mass1,m1), given(mass1)
```

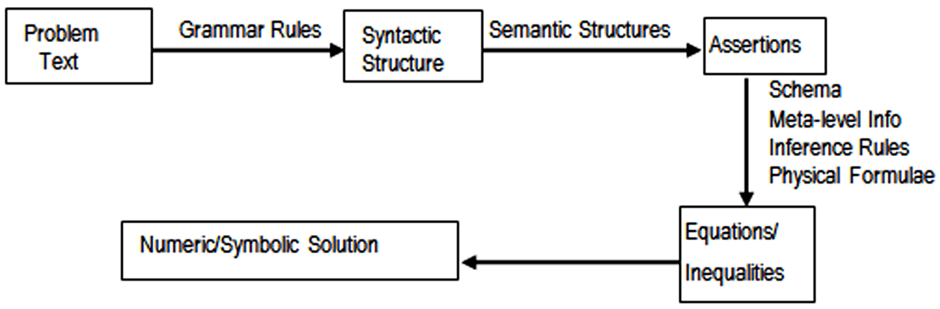


Figure 2.5: Schematic diagram of the processing stages of MECHO

Schema is a structure of certain key words and object configuration and is required to supplement for default information (*house rules* in the domain) which are not given explicitly in the problem. For example, a typical pulley system *schema* where the objects satisfy ideal-type constraints is defined as:

```
sysinfo(pullsys,
[pull,str,p1,p2],
[pulley,string,solid,solid],
[supports(pull,str),
 attached(str,p1),
 attached(str,p2)])
```

The input assertions provide meta-level information about whether certain quantities are given or sought. Moreover, before using an equation its applicability in the context of the problem is checked and/or the situation is created whereby an equation can be applied using meta-level information/reasoning. The following is one of the meta-level structures used for the above problem statement.

```
Isform (resolve,situation(Obj,Set,Dir,Time),
F = M * A)
   <--mass(Obj, M, Time) &
accel(Obj,A,Dir,Time) &
sumforces(Obj,Set,Dir,Time,F)
```

A general knowledge of mechanics which is required to solve a problem is formalized in a set of inference rules (object-level). For example:

```
Relaccel(p1,p2,zero,Dir,Period)
     <--constrelvel(p1,p2,Period)
```

This rule states that the relative acceleration between two points p1,p2 is zero if there is a constant relative velocity between them over a certain period.

MECHO demonstrates how the technique of using and controlling knowledge about the domain by inference at the meta-level can be applied to a wide range of mechanics problems. It clearly exploits the advantage that meta-level knowledge representation has over object-level knowledge representation.

General Physics Problems

The **ISAAC** program [45] written by Gordon S. Novak can read, understand, solve and draw pictures of physics problems stated in English. It has been successfully tested with 20 problems taken from high school and college physics text. One such problem is:

```
The foot of a ladder rests against a vertical wall and on a
horizontal floor.  The top of the ladder is supported from the
wall by a horizontal rope 30 FT long.  The ladder is 50 FT long,
weighs 100 LB with its center of gravity 20 FT from the foot and
a 150 LB man is 10 FT from the top.  Determine the tension in
the rope.
```

Along with numerical answer (120.00000 LB) a picture (Figure 2.6) is also drawn by ISAAC for the above problem.

A parsing program transforms the linear string of words into a structure that explicitly expresses the syntactic relationships of the words and phrases. The *parser* programs written in pure LISP actually implements the grammar of a phrase as an *augmented transition network (ATN)* influenced by the ATN formalism of W. A. Woods [105]; the grammar functions parse the phrases or clauses into a network of interlinked nodes which bear

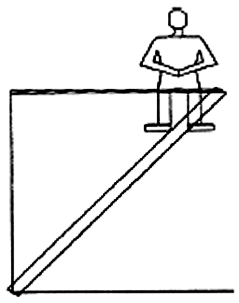


Figure 2.6: Picture output of ISAAC for the example problem

strong resemblance to the semantic networks of R. F. Simmons [89]. Each node in the network is a *GENSYSM* atom whose name is `TOK` followed by a number (e.g. `TOK290`) and represents the *main* word of a phrase (e.g. `TOK290` can represent `ladder`). Features of the nodes are stored on its property list under named *indicators* like `LFRAME` (indicating type of *linguistic frame* like noun phrase - NP, verb phrase - VB, etc.), `MODS` (modifier), etc. Then the semantic routines associated mostly with the verbs and prepositions of a phrase are invoked to determine the proper sense-meaning of the words just like the phrase `top of the ladder` having the preposition `of` belongs to the sense-meaning class `<location>OF<object>`. Some semantic routines identify the referents (e.g. `top of`) and modifiers (e.g. `a 150 LB man`) of the noun phrases. Finally each parsed structure is transformed into *semantic frame* (`SFRAME`) form whereby more *indicators* are added to the property list of the nodes or totally new frames created. Some of these *indicators* explicitly denote the link of a node or a phrase to other objects (`SEMOBJ`) or referents (`RFNT`) in the problem to which the node or phrase refers. Thus `SFRAME` are designed to reflect the attributes and relationship of objects as described in the problem.

Completion of syntactic and semantic processing of all the problem statements ultimately results in formation of the *internal model* of a problem which is nothing but a collection of `SFRAME`s like the one shown below.

```
LADDER291((TOK.LADDER) (ENTITY.PHYSENT) (LOCS LOC297 LOC309
LOC317 LOC322) (ATTACH ATTACH298 ATTACH299 ATTACH310 ATTACH323)
(SUPPORTBY FLOOR296 WALL294 ROPE305) (COFG LOC317) (LENGTH 50
FT) (WEIGHT 100 LB))
```

Here `LOC297, LOC309`, etc. under the *indicator* `LOCS` are list of all locations like `foot,top`, etc. of the ladder, `ATTACH298, ATTACH299`, etc. are list of all `ATTACH`ments concerning ladder like the one between the foot of ladder and floor, another between the foot of ladder and wall, etc. `COFG` denotes the location of the center of gravity of the ladder by pointing to the `SFRAME LOC317`. `LOC317` describes a location of the ladder 20 ft from its foot as shown below.

```
LOC317 ((FRAME.LOCATION) (ENTITY.LOCATION) (OBJECT.LADDER291)
(LOCNAME.FOOT) (REFLOC.LOC297)(RELPOS FROMLOC (20 FT)))
```

There is provision in ISAAC for representing some common-sense knowledge about usual features and relationships of particular type of objects programmatically (e.g. by a procedure to decide which object is referred to as the `force` in the problem) or in the form of data structure (e.g. typical geometry and location names of a `lever`). Such knowledge is required in addition to the information extracted from the problem to make the *internal model* complete for understanding a problem.To convert the *internal model* into a model suitable for writing equations describing the interaction of the objects of a problem *canonical object frames* (idealized objects) are chosen for all the objects in the model. A person, for example, might be modeled as a `WEIGHT` (having no geometric size) when sitting on a pole or as a `PIVOT` when carrying the pole. A total of seven *canonical object* types are used in ISAAC: `LEVER, WEIGHT, SPRING, PIVOT, ROPE, SURFACE, and FORCE`. After a *canonical object frame* is made for an object, its geometric size and its absolute rotations are found or assumed and absolute location names and coordinates are assigned to all of its locations. This helps construct the *geometric model* of the problem which basically represents the spatial position and orientation of each object.

Next, the *frame* completion routine is invoked by ISAAC for completing the attachment relations by associating appropriate force vectors for each object. The forces exerted by an object and the geometric position of the point at which each force is exerted are collected and put in the *attachment frame* under the *indicator* `FORCE`s. The next functional module called by ISAAC is the *problem-solving module*; the function `ATTDRIVER` write equations for each attachment relation according to the physical laws like sum of X-forces and sum of Y-forces must each be zero for an object in static equilibrium using LISP format alike those used by STUDENT program. For example:

```
(EQUALS 0 FORCE179)
(EQUALS 0 (PLUS (TIMES TENSION327 -1.00000) FORCE 332))
```

Similarly the `SOLVELEVER` routine calculates the moments of the forces (identified as acting on the lever) about the `PIVOT` point, sums them and sets the sum to zero. The equations so formed are solved by `SOLVEQ` and `PSOLVER` routines.

Lastly a *picture model* of a problem is generated to allow a diagram of the problem to be drawn. This model is similar to the *geometric model*, except that all dimensions of objects are numerical. Sizes are chosen for objects which have zero size in the *geometric model* (e.g. a person represented as a point mass). The different models generated in the processing stages of ISAAC are illustrated in Figure 2.7.

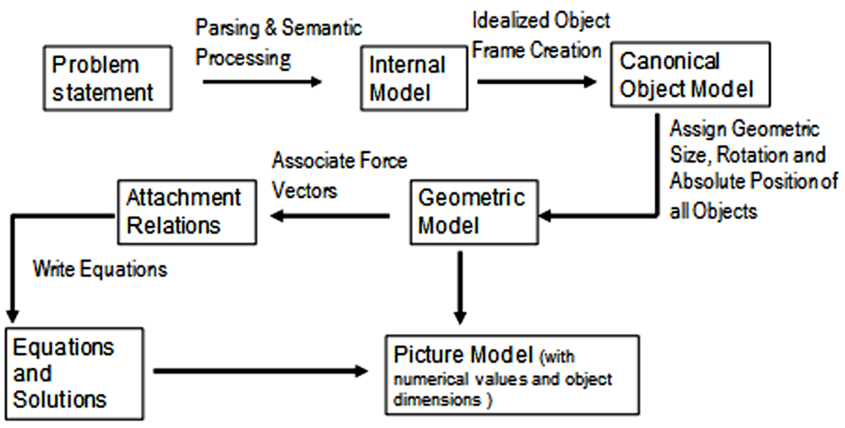

Problem statement
Parsing & Semantic Processing
Internal Model
Idealized Object Frame Creation
Canonical Object Model
Assign Geometric Size, Rotation and Absolute Position of all Objects
Geometric Model
Associate Force Vectors
Attachment Relations
Write Equations
Equations and Solutions
Picture Model (with numerical values and object dimensions)

Figure 2.7: Schematic diagram of the processing stages of ISAAC

The program **BEATRIX** [44] is an advancement over ISAAC. It understands not only text but also diagrams and can thus solve physics problems described in terms of statements and illustrations. It uses a single unified model that incorporates information from both text and diagram and establishes correspondence between parts of the text and diagram that refers to the same object or feature. To replicate the natural human approach of reading and understanding a problem with illustrations, BEATRIX is organized to allow co-parsing of the two input modalities using the BB1 blackboard system.

ALBERT [41] developed by Graham E Oberem is an intelligent tutoring (CAI) system that not only understands and solves physics (more specifically kinematics) problems but can also teach a student how to solve them. An intelligent discussion of a kinematics problem with a student is a more complex task than solving a textbook physics problem but ALBERT accomplishes both using separate natural language systems for each task. A lexical database of words grouped into 25 categories based on their semantic function is maintained. Traditional linguistic groupings such as noun and verbs are not used. In the first pass (lexical analysis), the words in the input text are parsed into numerical string where each number in the string corresponds to the lexical category of the associated word. Regular patterns occur in such numerical strings and around 100 commonly occurring syntactic patterns are maintained in ALBERTs knowledge base. In the second phase of the parsing, the numerical string derived earlier is searched for matching syntactic patterns. When a particular pattern is identified, a parameter-driven semantic routine is called to extract information from the corresponding text string. A small number of data elements are associated with each syntactic pattern to act as parameters of the appropriate semantic routines. When semantic processing is complete, all the information in the input will have been used to instantiate variables in the computational model of the solver.

FREEBODY [40] is an adoption of ALBERT [41] and a speciality software that understands the free-body diagram of a physics problem drawn by a student in a graphic user interface; it assesses whether or not a diagram is reasonable in terms of situation described in the problem statement and dynamically guides the student while each force is drawn by him. More or less the same lexicon and knowledge base as used in ALBERT is used by this program. The semantic routines are restructured to deal with more qualitative data, since there is no numerical problem solving in FREEBODY.

2.1.3 Chemistry Problems

ECLIPS [42] is a program designed to understand and solve chemistry problems of molarity stated in English. The program closely resembles ALBERT [41] as it uses both the language system and generic problem solver of ALBERT with minor modifications in the semantic routines. One additional semantic routine was needed to identify chemical formulae and to extract information from them. An example of a problem solved by ECLIPS is:

```
How many moles of $H_2SO_4$ are required to make a 0.1 M solution
with a volume of 375 ml?
```

Information extracted from the problem and relevant knowledge gathered from knowledge base by ECLIPS are: the solute is H_2SO_4, atomic mass = 98.08 a.m.u, the mass of a mole is 98.08 g, the volume of the solution (= v) = 0.375 litres, solution strength (= c) = 0.1M. Number of moles (= n) is required to be calculated. The solution steps are:

```
 Identify the relevant formulae:  $c = n/v$
Rearrange the formula (to keep the unknown variable on the left
hand side):  $n = cv$
Put the values of $v = 0.375$ and $c = 0.1$ in the above formula
Find $n = 0.037$ mol
```

2.1.4 General Numerical Problems

A language based problem solver was designed and implemented by S Ramani [95, 96] that can respond to a problem stated in natural language based on analogy with similar problem, solution steps and answer formats stored and learnt by the system previously. The system uses an analogy directed language processor that works irrespective of any specific language. The response of the processor to any situation is based on pattern setting examples called *paradigms* given earlier. A *paradigm* consists of a *context*(C), an *input* (Q) and a *response* or output (A) and is stored in a form called *schema*. A database storing sentential strings (questions, commands, and informational sentences, etc.) is consulted and the *schema* is extrapolated to create responses to new situation or *stimulus*. Considering a simple paradigm as:

```
C: The density of lead is 8 gm per cc
Q:  Find the weight of a lead sphere 6 cm in diameter
A:  Find the volume of the sphere
The volume is 0.75*PI*(0.5*6)^3
Multiply the volume by 8
The result is the weight of the sphere
```

given a new situation as:

```
C:  The density of a metal is 10 gm per cc
Q:  Find the weight of that metal sphere 4 cm in diameter
```

the processor would logically work as following to find the weight.

```
A:  Find the volume of the sphere
The volume is 0.75*PI*(0.5*4)^3
Multiply the volume by 10
The weight of the sphere =
```

Schematization of paradigm serves essentially to assign a structure to the *paradigm* facilitating extrapolation. Any input *stimulus* S, a string of words and symbols, triggers off a search activity in the processor. The search is for the *schema* of a *paradigm* p[C,Q,A] which has an input component Q maximally resembling S. If no suitable *schema* is found for a given *stimulus* S, it is treated as informative input to be directly sent to the sentential store.

2.1.5 Comparative Summary

Among several systems studied (summarized in Table 2.2) STUDENT [27], ISAAC [45], MECHO [1] and LIM-G [106] are of special interest to us as these are closer to our research. Therefore their relative merits and limitations are discussed more carefully.

Bobrow used the concept of *relational model* (object-relationship-media) quite effectively in STUDENT as was used by R. K. Lindsay in his SAD-SAM program [90] for internal representation of the input problem. This model proved to be quite useful for algebra problems as standard techniques are there for finding numerical values that satisfy sets of simultaneous equations, thus taking full advantage of the relational language media. The main challenge of Bobrows approach was to map the English sentences directly into the *relational model*. Though it worked for a number of algebra story problems it has definite limitations. The technique works only when the sentences express algebraic relationships among quantities. The variable phrases must be similar in each occurrence so that they can be matched properly and the key words must not be used in multiple ways which might confuse the pattern matcher. Moreover, the method adopted for extracting simple sentences from complex and compound input statements is adhoc (heuristic) and primitive and might not work for sentences of great grammatical complexity. It may be noted that STUDENT didnt require storing knowledge except having a glossary of few global facts.

ISAAC [45], though tested on only 20 static problems, seem to be a more complete natural language scientific problem solver with well defined functional modules and generic structure of the internal representation model. Since the model makes explicit all of the features and relationships of the objects, it is many times larger than the original problem statement and comprehensive enough to answer any question related to a problem. It has the potential to draw force diagrams for a problem if required (in addition to its capability of drawing illustrations) and can even facilitate machine translation. The *SFRAME (semantic frame)* concept used by Novak for semantic interpretation of problem statements is likely to be important in any domain as far as the process of understanding is concerned. The application of *canonical object frame* [75] is unique. It can be really powerful if extended to handle dynamic attributes (e.g. velocity and acceleration as function of time) or model more complex objects in engineering domain. In addition, since such a system could handle symbolic as well as numeric calculation it could perform analysis task which most of the specialized programs cannot handle at all. ISAAC can be used as a test bed for investigating different strategies for solving a given problem. However, like all the other systems, ISAAC has its own limitations. It cannot handle new type of problems accurately without further adjustment in the program or addition of vocabularies. Secondly, the procedures used for referent identification are fairly rudimentary and deal only with extensionally specified referents. Finally, the number of equations generated per problem is much more (10 on an average) than what humans usually generate. There remains scope to apply some redundancy check and generate only the critical equations more directly.

We find MECHO [1] developed in Prolog quite interesting in the domain of Mechanics problem solving. While many of the earlier programs including STUDENT [27] and CARPS [34] used lot of heuristics and rudimentary parsing to collect useful information from natural language problems, MECHO exploited the advances on NLP techniques; the NLP module of MECHO produced a set of predicate calculus *assertions* about the objects (of a problem) to facilitate problem solving. It uses *schema* and *meta-level structure* similar to the *frame* structure used by Kleer [52] for representing input-supplementary yet conceptual information and inference rules. David Marpless search algorithm [29] is applied strategically to extract equations from the combination of input *assertions*, *schemas*, and inference rules. The philosophy underlying the operation of MECHO is the application of *meta-information - meta-terms*, *meta-assertions* and *meta-rules* [86] in semantic processing, inference mechanism, and knowledge representation which is principally different from object-level representation used in other natural language problem solvers. Many researchers argue that *meta-level inference* is superior to other strategies as it results in more transparent and modular representation of knowledge that forms the core of an expert problem solving system. It is hypothesized that human subjects solving mechanics problems employ similar model building techniques as applied in MECHO. Thus it could serve as a cognitive model as is evident from the work of Luger [43]. MECHO is especially significant as it led to the development of *meta-level reasoning*, a distinct subfield of AI. For example, MECHO led to PRESS [66], a seminal piece of work in automated equation solving using *meta-level inference* as its basis.

Following an instructor-initiated strategy most CAI systems use a pre-designed database of systems and do not address the issue of understanding or solving unknown and non-standard problems. In contrast, LIM-G [106] is an intelligent tutor that doesnt use any database of model problems or standard solution steps. Rather it uses a powerful knowledge base and can not only comprehend standard problems but can also deal with those having missing or extraneous information. The system is tested with fairly large number of school-level elementary mensuration problems and yielded an impressive success rate of about 85%. However, it cannot fully comprehend problems that involve more than one shape and cannot deal with problems which involve complicated geometric shapes other than the seven classes of original shapes defined in the knowledge base. Any geometric condition or constraint satisfaction is also not within the scope of the system which limits its utility to picking the correct formulae for a geometric shape and substituting numeric values for variables. It is yet to be explored whether LIM-G model is extendable to other domain as claimed by the authors.

Contemporary systems which we could not include in this review might differ in the style and low-level stages of functioning from the ones we have discussed here. But, significantly, the basic architectures of all these systems are found grossly same (refer Figure 2.8) containing four important logical parts: the language processor and analyzer, the knowledge base, the inference engine and the user interface. The knowledge base is a combination of database and rule base that the inference engine uses to make inferences or solution. The language processor and analyzer extracts the data or information expressed through the text by syntactic and semantic processing of the problem statements. The user interface simply passes data or information back and forth between the user and the expert system. We see some of the earlier systems (e.g. STUDENT, ISAAC) had to use tailor-made language processing routines mostly using format matching techniques while the later systems (e.g. LIM-G) benefited from the parallel developments in the area of NLP. We have seen how different schemes like *paradigm*, *schema*, *proposition*, *atom*, etc. or structures like *tree*, *frame*, *meta-structure*, *ATN*, etc. or *database* or *files* are used to store information, inference rules, and knowledge. In many cases, adhoc and heuristic programming approach or weak search procedure is adopted that seems quite useful for typical cases but renders too narrow a scope for proper understanding and representation of problems of diverse nature.

Table 2.2: Summary of the Natural Language Systems reviewed in Chapter 2

Solver	**Author**	**Year**	**Domain**	**Basic Feature**
STUDENT	Daniel G Bobrow	1964	Algebra	Keyword-based pattern matching, relational model (object-relationship-media)
WORDPRO	Charles R Fletcher	1985	One-step arithmetic problems	Set schema and rule base
ROBUST	Yefim Bakman	2007	Multi-step arithmetic problems with extraneous information	Schema instantiation, application of change formulae to elementary propositions
LIM-G	Wing K Wong et al.	2007	Elementary Geometry (Mensuration)	Knowledge base using InfoMap, frame-template structure, problem representation thru tree structure
CARPS	Eugene Charniak	1968	Distance and volume rate problems	Tree structure, keyword-based pattern matching, knowledge base of formulae, etc.
HAPPINESS	Jack P Gelb	1971	Probability	Tree structure or descriptor-list, keyword-based solution method, pattern matching, tailored solution function
NEWTON	Johan De Kleer	1975	Mechanics (kinematics)	Tree structure (qualitative information), Frame structure (quantitative information)
MECHO	Alan Bundy et al.	1979	Mechanics (statics and kinematics)	Input assertions, schema (house-rules), meta-level structure (information and inference rules)
ISAAC	Gordon S Novak	1976	Physics	Augmented Transition Network, semantic frame, canonical object frame, knowledge representation thru programs data structure, geometric model
ALBERT	Graham E Oberem	1987	Physics (CAI for kinematics prob.)	Lexical database, knowledge base of syntactic patterns, syntactic pattern matching
FREEBODY	Graham E Oberem et al.	1993	Physics (CAI for FBD drawing)	NLP and problem solving scheme basically same as that of ALBERT
ECLIPS	Graham E Oberem et al.	1994	Chemistry (molarity problems)	NLP and problem solving scheme basically same as that of ALBERT
—	S Ramani	1969	General Numerical Problem	Analogy-directed language processor, schema/paradigm (knowledge), extrapolation algorithms for analogizing

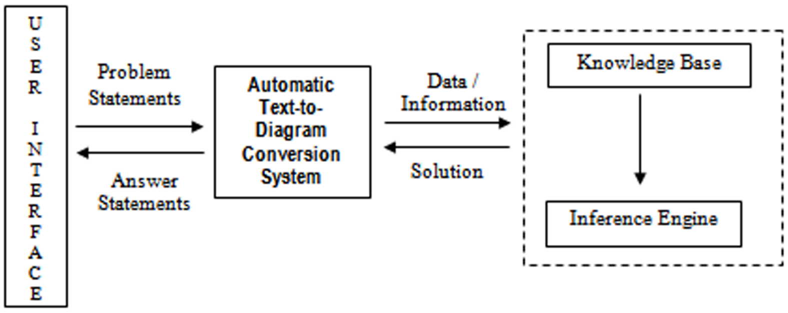


Figure 2.8: Interaction of the logical parts of a natural language problem solver

The review paper [5] contains significant works till 2007. However, researchers are still continuing to contribute in this field [77]. The idea of using lexical resource in representing knowledge has been exploited in many contexts. For example, the paper [59] describes the role of a knowledge base in designing a natural language understanding model. In the context of query answering, the paper [103] examines the task of identifying a knowledge base and using it to answer questions on a wide variety of topics. Again the study in [108] attempts the 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 representation with hierarchical structures. Of late, Baral et al. [18] 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. The idea to extract knowledge from text and represent it in a graphical form was attempted [71] in the context of representing textual knowledge in the form of a UML.

2.2 Graphics for the Blind People

In the context of possible extension of the present research work for the blind people, a substantial survey is also made on the existing systems for conveying graphics (particularly geometry figures) to the blind students. There had been lot of research on this subject and several sophisticated interfaces and programs have been developed to that end in developed countries. Here we only touch upon the existing systems without going for an in-depth technical comparison. This discussion brings out the fact that there exists hardly any low-cost and easily available assistive technology to support classroom teaching/learning of diagram-based subjects (particularly geometry) in the resource-constrained blind schools in countries like India. Therein lies the utility of the survey which underlines the need and relevance of developing an online text-to-diagram conversion system in a widely accessible medium for the blind people.

Upon examining the technologies used by the existing systems, we observe five modes of graphic communication for the blind people: (1) *tactile*, (2) *audio-tactile*, (3) *audio-haptic*, (4) *3D audio* and, (5) *multimodal*.

2.2.1 Tactile Mode

In tactile mode tactile images with raised lines or dots are produced on Braille paper by Braille image embossers. These embossers with variety of capabilities and spanning a significant price range[1] (as cited in Table 2.3) usually come with compatible Braille graphics software tools for creating and editing tactile graphics. *IVEO Viewer* by ViewPlus [50], *Picturebraille* by Pentronics [84], *Tactile Graphics Designer Pro* (*TGD Pro*) by Duxbury Systems [33] are examples of such software. The *IVEO Viewer* uses accessible *Scalable Vector Graphic* (*SVG*) technology that can provide visual, audio, and tactile (audio-tactile) access to graphical information (maps, charts, graphs, diagrams). One can get a tactile copy of any SVG image [91] created on screen by printing from *IVEO Viewer* to a ViewPlus embosser. The tactile copy is then placed on the IVEO touchpad and the blind user can trace the figure by moving fingers on it - when he presses on an object or text label the corresponding information is spoken. *Picturebraille* for Windows is a Braille graphics program by Pentronics designed to produce maps, floor plans, science diagrams, graphs of equations, or other images for individuals who are blind or have low vision. Images are either drawn free-hand on the screen, using the simple drawing

[1]Popular and lesser cost models of image embossers cost twice as much as low-cost Braille text printer

program supplied, or scanned from a document such as a textbook or magazine. Once the image is created or imported, the program converts it to a standard Braille file. *Tactile Graphics Designer Pro* (*TGD Pro*) is a product combining two Braille graphics programs, *QikTac* and *TraceME*. In *QikTac* [33] a drawing is first made using a mouse or by tracing and editing a picture using a *TagPad*. The drawing can then be embossed using a Braille printer. *TraceME* is specifically designed for freehand sketching, tracing, and/or transforming of color photographs, line drawings, clip art, or scanned images into a tactile form for output using a Braille printer.

There is another category of automated tools that can translate any normal image to tactile form and then print using the image embossers. ViewPluss *IVEO Creator* and *IVEO Creator Pro* [50] allows graphics created in any popular software like Adobe Illustrator, CorelDraw, MS-Word, PDF writer, etc. to be opened in SVG format and get printed in Braille format. *Tiger Software Suite* (*TSS*) [104], *TACTICS* or *TACTile Image Creation System* [101], *TeDUB* or *Technical Drawings Understanding for the Blind* [70] and *TGA* or *Tactile Graphics Assistant* [87] are some other examples. Besides these, *Tactile Image Enhancer* [17] is a specialized device by Repro-Tronics that converts an image drawn on FlexiPaper with ink-jet printer into raised line drawing when passed through the device. Two most traditional, universally used tools to help the blind users get primary idea about geometry shapes through tactile perception are the nail-board-elastic band system and the slate-stylus system. Other modern toolkits include- *APH Geometry Tactile Graphics Kit* [14], and *APH Draftsman Tactile Drawing Board* [14] and *Math Window* [16].

2.2.2 Audio-Tactile Mode

In this category we find several research prototypes that provide interactive interfaces to draw and trace the drawing with the help of audio feedback. *IC2D* [72] is one such that provides the user selection access of center points of square screen regions (cells) through recursive decomposition of screen or a cell into 3x3 grid; objects can be drawn by moving the cursor between cells using specified directional keys and number keys. *TDraw* [73] uses a thermo pen to draw on a swell paper fixed on a digitizer tablet which gives the state (pressed/released) and position of pen during drawing. Each object is

Table 2.3: Comparison of price and features of Braille printers/embossers(courtesy [68] with price updated)

Name of Braille Embosser	**Key Features**	**Cost (US$)**
Basic-D by Index Braille	Cannot emboss images; Tractor fed paper; Z-Folding	3,295
Everest by Index Braille	Cannot emboss images; Embosses documents, Braille books, labels and visiting cards; Can use ink and Braille in same document	4,195
4x4 Pro by Index Braille	Can emboss images with WinBraille; Newspaper format	4,795
Emprint SpotDot by ViewPlus	Print the original ink text together with Braille allowing sighted reader to follow along; embosses images; New easy-to-use operator panel with tactile buttons; Uses the same paper and ink cartridges as an HP Inkjet printer	6,995
Premier 80 by ViewPlus	High-speed Braille along with Tiger super-high-resolution graphics ; Automatic double-sided embossingno flipping the paper; Production-strength hardware made for running long hours; Compact desktop sizesmaller than most production embossers	9,995
4Waves Pro by Index Braille	Can emboss images with WinBraille; High embossing speed; 4 modules + one service module; Low noise level	25,995

immediately tangible as one draws it and recorded by a computer along with an identifying name as the user speaks it. Minagawa Ohnishi [46] has prototyped tactile-audio diagram (for maps and circuits) represented by a tactile pattern and linguistic information. The system consists of a PC, a specially designed tactile display with a digitizer, an audio input/output device and a command keyboard. Tactile patterns (for typical diagram-elements) are created by adjusting the height of pins on the tactile display and recording input auditory information linking to each pin. Users perceive the tactile pattern (spatial information) by touching the pins of different heights on the display surface. The linguistic information (position, shape, type, etc.) is simultaneously output as voice previously recorded. *NOMAD* [32] developed by Parkes is a well-known audio-tactile drawing system consisting of a touch pad connected to a computer and a speech synthesizer for reading a tactile diagram. Apart from getting a rough layout of the diagram by touch, when a blind reader places a tactile diagram on the pad and presses on a point of interest on the diagram, detailed information is spoken by the computer. The *Talking Tactile Tablet* [93] and *IVEO touchpad* [91] are similar examples of interactive audio-tactile interface. Watanabe et al. [102] pursued a different approach, combining a tactile pin array device, a 3D digitizer, and a tablet PC to allow students to draw and erase lines by moving a stylus along the tactile surface, which they would then explore with their fingertips.

2.2.3 Audio-Haptic Mode

In the audio-haptic category we find systems that primarily relies on haptic (usually force rather than tactile) feedback to help convey graphs and shapes to visually impaired students [79]. One approach is to incorporate audio with force feedback devices such as the *SensAble Phantom* or *Logitech WingMan Force Feedback Mouse* to explore and create graphs [64, 63]. Another haptic-based approach [98] used a stylus with an embedded vibration motor in conjunction with a touchscreen to convey a graphical trace to a user. In a recent report [54], we find a specially designed haptic or vibratory touchscreen (with actuators) on which a blind user can draw a figure by touching and moving fingertips on and across the screen as a software tracks the touch events/trajectory. While tracing a diagram on the touchscreen the user receives vibratory feedback to the finger (in haptic mode) and tones with specific frequencies and durations (in audio mode) that help perceive point locations, line extent and simple geometric shapes.

2.2.4 3D Audio

With advances in 3D audio technology we find some system that uses 3D immersive auditory cues which are really useful for the blind people to mentally represent the topology of a graphical environment. In this area, *From Dots to Shape* created by Roth et al. [83] is an experimental game platform whose goal is to help educators in their teaching blind and visually impaired students basic Euclidean geometry. The sytem making use of a graphical tablet, SoundBlaster Live sound card and headphone shows in some initial experiments how the students can locate points and line path and identify geometric shapes by listening to sound cues on a sound screen.

2.2.5 Multimodal

In the multimodal category *SALOME* [97, 99, 21] is a research prototype that uses specialized force feedback device which is used as a pen to make command gestures for drawing on a virtual plane. A gesture recognition engine interprets those gestures as different geometric objects like point, lines, triangle, circle, etc. The figures so drawn are actually coded as haptic magnetized rails that attract the pen towards different drawing elements when the users try to trace the figures using the pen. Each drawing element also has an audio description that enhances perception of an element. One can refer [51, 2, 37] for better overview of multimodal systems for making graphical information accessible to blind people.

In all the audio supported tactile drawing systems, if the audio information was sufficient most blind users could quickly learn well-authored graphics but still complex graphics were not easily understood by them. Despite some specific advantages the research outcomes on audio-tactile and audio-haptic method of communicating graphics to the blind people didnt become broadly popular primarily because the components are specialized and expensive and secondly not suitable for communicating graphics to a number of students at a time in a classroom. For the haptic based and 3D audio based systems, the learnability of gestural elements and sound cues respectively by blind pupils is also questionable.

As far as classroom teaching of geometry to the blind students is concerned, Indian researchers [92] have designed an innovative PC-network based group teaching system. The blind students can trace diagrams (created in teachers machine) in their monitors by using an optoelectronic sensor and perceive it with the help of a device producing vibrotactile output. However, in this system, the blind students themselves cannot create the diagrams. Borges et al. have conducted some experiments with blind students of Brazil using the prototype *DESENVOX* [49] to explore how simple geometric elements (like points, straight line segments, curved lines, rectangles, etc.) and shapes can be generated in a computer through a specially designed menu system and voice guidance utility and finally embossed by a Braille printer. This scheme, widely used in Brazilian blind schools, is a good geometry teaching tool for the blind people at moderate cost.

2.2.6 Diagram Drawing using Braille Text

As one of our research objectives is to develop an utility of automatic diagram drawing (from text input) using low-cost Braille text printer, we have searched the literature for existing tools or techniques used for that purpose. But, very limited research can be found on this mode of drawing. Rahman et al. [68] have investigated the potential use of a software solution for converting regular print image into tactile image using low cost Braille text printers. The software is designed for sighted users who are required to select images on a printed page, run the software with the printed image as input, and then print the corresponding tactile image for the blind readers.

Using an iterative thresholding technique, the software first obtains a thresholded binary image from an original image after resizing it to fit to an 84x84 pixel frame. Now, each 3x2 block of pixels in the frame is considered equivalent to a Braille cell (3x2 dots) - there are as many pixel blocks (28x42) in the pixel frame as there are printable Braille cells (28x42) in a Braille sheet. The pattern of the marked pixels in each block is mapped to yield same pattern of raised dots in the corresponding cell of the Braille grid. All the mapped cells (or Braille characters), when printed, reproduce the original image in tactile format. Figure 2.9 illustrates this method of Braille mapping of a digital line.

This system, though, do not produce figures automatically from textual description, draws our interest as it can represent tactile figures in terms of embossed Braille dots that usually imply Braille text. The basic approach taken by this system is linear mapping from an evenly spaced pixel-grid to an unevenly spaced dot-grid which naturally causes some distortion in the shapes. The process is simple but does not attempt to optimize the number or position of dots to be embossed. Quite expectedly, it causes redundant dots being selected at positions other than the best possible positions with respect to the actual entity (line or circle).

BrlGraphEditor [69] and *Sparsha Chitra* system [4] are two other software tools (with

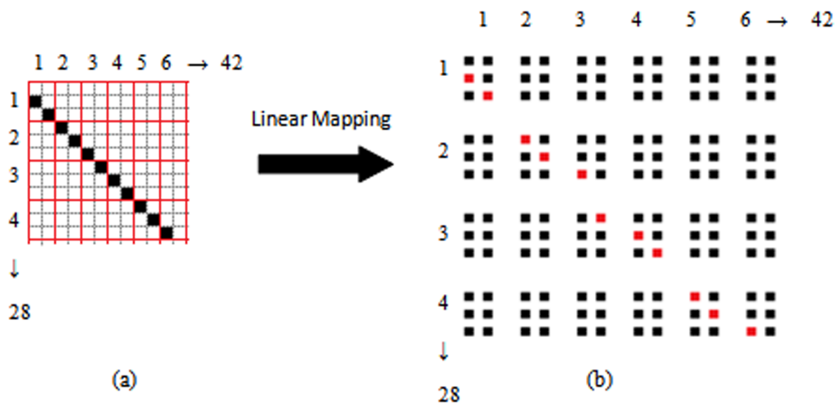


Figure 2.9: (a) The pixel frame comprises 42 columns and 28 rows of pixel blocks, each containing 3x2 pixels. (b) The Braille grid comprises 42 columns and 28 rows of Braille cells, each containing 3x2 Braille dots. The pixel blocks with black pixels in (a) are mapped to the Braille cells at similar positions in the Braille grid of dots (small black squares) in (b). This ultimately results in a series of raised Braille dots (faded squares) representing the tactile version of the digital line in (a). The pattern of raised dots in a Braille cell represents a character of Braille text.

design similar to the one developed by Rahman et al. [68]) that convert images into suitable form for printing on a Braille text printer. Both the systems provide a GUI editor that displays graphics in exactly the same resolution as used by the Braille printer. The tactile graphics viewed in the editor can be edited before finally being embossed. However, no optimization attempt is made by the systems to improve the graphics.

It may be noted that the tools discussed above for Braille-text based figure drawing, basically work on existing printed images. Their capability in generating complex geometry diagrams is not readily understood as those were not employed to do so. Moreover, no technique for numerical evaluation of the Braille figures produced by these systems has been reported.

CHAPTER 3

GeometryNet

3.1 Introduction

GeometryNet [11] can be considered as a simple knowledge repository containing the mathematical definition of different geometric entities and concepts found in school level geometry text books. The idea is borrowed from *WordNet* [38], a popular lexical database often used for natural language processing (NLP) tasks. The basic objective behind constructing a *GeometryNet*(hereinafter at times referred as *Net*) is to provide our system with the domain or lexical knowledge required for machine understanding of school level geometry problems stated in English.

Upon reading a geometry problem, prior to solving it or drawing the corresponding diagram the logical human approach is to analyze the problem description first to extract the following information: (i) basic entities (like lines, circles, triangles, etc.) it refers to, (ii) entity parameters (like vertex, center, etc.) and entity attributes (like horizontal, in-circle, equilateral, etc.), if any, and (iii) entity relations (like produce, intersect, bisect, etc.) or relational attributes (like intersection point, in-centre, etc.), if any. This extracted information immediately prompts our generic knowledge about the respective geometric entities and their relations. This generic knowledge is then mapped to the problem instances. In effect, the geometric phenomena of the problem is completely understood and we attempt to draw or solve it. Ideally, *GeometryNet* is designed to enable machine to follow the above approach in interpreting a geometry problem. It contributes to the intelligence of our system as an application of AI and is thus a major contribution of our research. To the best of our knowledge, no such knowledge base exists in geometry domain.

Before proposing the structure of *GeometryNet* more than 300 geometric problems ranging from grade VIII standard to grade X standard of ICSE, CBSE and WBBSE school-education Boards of India were studied and analyzed. It helped identify all the varieties of geometric entities (nouns), attributes (adjectives) and relations (verbs) occurring in geometry problems. The *WordNet* model of organization of nouns is almost totally adopted in *GeometryNet* while that of verbs is partly followed and that of adverbs and adjectives is hardly followed. Unlike in *WordNet* where the words *synonym sets* do not carry the underlying semantics, in *GeometryNet* use of mathematical expressions with word forms represent the underlying geometric concept. However, the structure of *GeometryNet* data types and relations are implemented using inheritance and interlinking similar to that in *WordNet*.

This chapter outlines some key concepts of *WordNet* [109] that are used in *GeometryNet*. It presents the detailed description of organization of three different categories of geometric terms and concepts with illustration for better understanding. Additional features and issues behind constructing the *Net* are also discussed here.

3.2 Construction Details

Upon analyzing geometry word problems of school level text books we identified 51 different types of geometric entities and entity parameters, 50 types of entity attributes, and 35 interaction types or relations between the entities. From linguistic point of view the entity and entity parameters are basically nouns while the attributes are adjectives and the relations mostly represented by verbs. This syntactic categorization of geometric terms is required because the linguistic analysis of the problem statements will extract only parts-of-speech (POS) out of which the nouns, adjectives and verbs will be separated to find their meaning in *GeometryNet*. The geometric properties of the identified entities and relations were further studied to find out what generic information is necessary to store the meaning, i.e. the mathematical interpretation of each term in *GeometryNet*. It is found that a term can be completely defined by some coordinate variables and some mathematical relations and equations involving the variables. Accordingly the *Net* is built as a domain specific ontology for geometrical terminology, where the semantics of certain terms are finally expressed in terms of equations involving their arguments. Also, default numerical values are stored against each defining parameter of all entities (e.g. two endpoint coordinates of a line segment, coordinates of vertices of a triangle, parallelogram etc, center and radius of a circle and likewise). The reason being geometry problems seldom explicitly mention or imply the parameter values of entities or relations required to draw representative figures of such problems. But even then our system must be able to draw a standard (default) figure in keeping with the basic geometric definition (in *GeometryNet*) as we do intuitively while working out a geometry problem manually.

3.2.1 Terminologies

In building *GeometryNet* some concepts of hierarchical data structure are borrowed from *WordNet*. These are illustrated below with reference to *WordNet*.

- *Hypernymy / Hyponymy*: Between two nouns x and y, y is a *hyponym* of x if it can be said that *y is a (kind of) x*. In that case, x is the *hypernym* of y. Such relation is also termed as *IS A* relation. For example, *tree* is a *hyponym* of *plant* and *plant* is a *hypernym* of *tree*.

- *Superordinate / Subordinate*: *Superordinate* refers to *hypernym* whereas *subordinate* means *hyponym*. In the above example, *plant* is a *superordinate* of *tree* and *tree* is a *subordinate* of *plant*.

- *Meronymy / Holonymy*: If two nouns x and y are such that a y has an x (as a part) or an x is a part of y then x is a *meronym* of y while y will be termed as *holonym* of

x. Such relations are also called *HAS A* relation. For example, *wing* is a *meronym* of *bird* (because *bird* has a *wing*) and *bird* is a *holonym* of *wing*. *Meronym* can be of different types. For example:
x #p$\rightarrow$ y indicates x is a component part of y
x #s$\rightarrow$ y indicates x is the stuff that y is made from

- *Troponymy*: A verb x is a *troponym* of verb y if it can be said that *to x is to y in a particular manner*. For example, *limp* is a *troponym* of *walk* because *to limp is to walk in a particular manner*.

- *Polysemy*: A word exhibits *polysemy* and is said to be *polysemous* if it bears more than one meaning. For example, the word *case* can mean *cupboard, box, example, legal case,* etc.

- *Antonymy*: If a word x is semantically and lexically opposite to another word y then x is said to be an *antonym* of y and vice versa. For example, *man* is an *antonym* of *woman*.

- *Synset*: Similar meaning word forms expressed within brackets {}, separated by commas form a *synonym set* or *synset*. For example, {board, plank} and {board, committee} are two different *synsets*.

3.2.2 Nouns

Looking at the organization of nouns in *WordNet* [39] we find that they are related by *hyponymy/hypernymy* relation that is also termed as *subordinate/superordinate* or the *ISA* relation. In geometry we can find many such semantic relations between geometric entities which are basically nouns. For example, the diameter of a circle is a line segment, the median of a triangle is a line segment, a vertex of a triangle is a point. So diameter and median are *hyponyms* of line segment whereas vertex is a *hyponym* of point. Basic entities like line segment and point are called *superordinates* while diameter and vertex are *subordinates* of line segment and point respectively. Other than line segment which is a line of finite length there is another variety of line (like the coordinate axes) that has fixed orientation but no finite length. Therefore such lines are more generic compared to a line segment and we can say line segment (hereinafter referred as *ls*) is a *hyponym* of line. Here, by the term line we mean straight line. We could have added another level in this hierarchy of line entity if we introduced the concept of curved line. Then curved line and straight line would have been the *hyponyms* of line and *ls* a *hyponym* of straight line. But considering little relevance of curved lines in school-level geometry we have ignored it in the *GeometryNet*.

The structure of organization of nouns in *GeometryNet* is marked by the entity hierarchy (Figure 3.2) which is implemented after identifying all *hyponymy* relations that exist among the 52 geometric entities that we identified and analyzed. The structure becomes meaningful as we include in the *Net* entity parameters or attributes to characterize the entities. It becomes an inheritance system in the sense all the characteristics of the *superordinate* are assumed to be characteristics (be it active or passive) of the *subordinate* as well. Instead of listing common properties redundantly with both *superordinate* and *subordinate* they are listed only with the *superordinate*.

Mathematical feature or *parameter* is one such characteristics. For example, a *line*s feature is its slope, mathematically expressed as $\frac{dy}{dx}$ or $tan\theta$ or m. But actually drawing a *line* would also require a point coordinate (x, y) through which the *line* passes thus specifying the exact position of the *line* in a 2D space. So a *line* can be listed with basic features m and (x, y). An *ls*, which is a *hyponym* of *line* is not fully defined with m and *(x,y)* inherited from its *superordinate line*. Additionally it requires two end point coordinates, say *(x1,y1)* and *(x2,y2)* for its generation in a 2D space. So it is necessary and sufficient to include parameters *(x1,y1)*, *(x2,y2)* only in the definition of *ls*[1]. Again *diameter*, a *hyponym* of *ls*, requires in its definition the coordinate *(xc,yc)* of the *centre* and *radius (r)* of the *circle* of which it is a *diameter*. Although for drawing the *diameter*, its end points *(x1,y1)* and *(x2,y2)* are essential and are inherited from its immediate *superordinate ls*. From geometric point of view these two end point coordinates cannot be arbitrary; it should satisfy mathematical relations as:

[1]We have kept *(x1,y1), (x2,y2)* and m in the definition of *ls* as because we have not listed *line* in the initial implementation of *GeometryNet* for its limited use.

```
(x1-xc)^2+(y1-yc)^2=r^2
(x2-xc)^2+(y2-yc)^2=r^2
 xc=(x1+x2)/2, yc=(y1+y2)/2
```

Herein lies the necessity of a *parameter condition* listing along with the parameters as characteristics of an entity. For an *ls* the parameter condition required is $(x1, y1) \neq (x2, y2)$, i.e. the two end points should not be identical. But there may not be parameter condition as such for some entities. We differentiate the parameter condition statements with a # symbol prefixing the statement. Given below is an example of entity (*parallelogram*) listing in the *Net* showing how constituent entities, parameters, and parameter conditions are stored against the entity.

```
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 it shows 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 information that the sides (of a *parallelogram*) have two *point*s and a slope *m* or that the vertices are basically points having a coordinate pair as parameter is not listed explicitly against each *side* or each *vertex* respectively. This is implied. Because according to the *GeometryNet* structure there is natural inheritance of properties of the *subordinate* entities from the *superordinate* entities. Parameters of vertices of the parallelogram, i.e. `vertex_1, vertex_2, vertex_3, vertex_4` are inherited from generic *superordinate* entity *point* which is separately listed as:

```
point
(x,y)
end
```

So, `vertex_1` implies `point_1` with parameter `(x1,y1)`, `vertex_2` implies `point_2` with `(x2,y2)` and so on. The inheritance occurs because we define *vertex* as a *point*.

```
vertex
point
end
```

Similarly, from the following generic definition of *side, ls* and *m* the parameters `side_1, side_2, side_3, side_4` are inherited. `side_1` is a *ls* and has `point_1, point_2, m1`, where, `point_1=(x1,y1), point_2=(x2,y2)` and `m1=(y2-y1)/(x2-x1)`; `side_2` is a *ls* and `has point_2, point_3, m2`, where, `point_2=(x2,y2), point_3=(x3,y3)`, and `m2=(y3-y2)/(x3-x2)`, and so on.

```
side
ls
end
ls
point_1
point_2
m
end
m
#(y2-y1)/(x2-x1)
end
```

Thus repetition of information is avoided in *GeometryNet*. When a program searches for knowledge about *parallelogram* it triggers sequential searching of knowledge about *vertex, side, point, ls* and *m*.

The parameter conditions of *parallelogram* are the four mathematical relations prefixed with #. The first two relations,

```
#((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
```

depict opposite sides of a *parallelogram* are equal in length. The other two relations,

```
#m1=m3
#m2=m4
```

depict slope of the opposite sides are same, i.e. they are parallel. Figure 3.1 may be referred here.

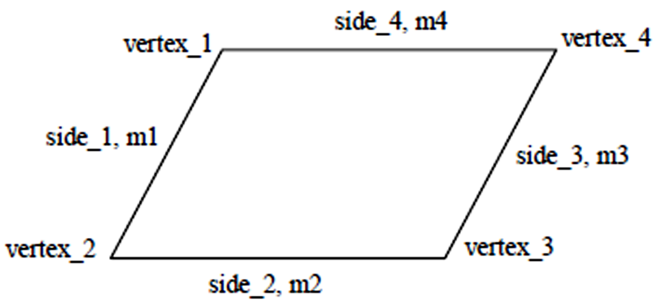


Figure 3.1: Sides and vertices of a parallelogram

Parts or *component* is another characteristics of *GeometryNet* nouns. In the above example, four *vertices* and four *sides* are parts of *parallelogram*. Referring to the concept of *meronymy* or part-whole relation (i.e. *HAS* relation) applied in *WordNet*, we can say `vertex_1, side_1` are *meronyms* of *parallelogram* or in other words, a *parallelogram* has `vertex_1, side_1`. Similarly, *midpoint* is a part of a *ls*, *arc* is a part of a *circle*; so *midpoint* is a *meronym* of *ls* and *arc* is a *meronym* of *circle*. *Meronyms* are distinguishing characteristics that *hyponyms* can inherit. For example, *midpoint* is a *meronym* of *ls* and *diameter* is a *hyponym* of *ls*, then by inheritance *midpoint* is also a *meronym* of *diameter*. Similarly, `point_1, point_2, m` are *meronyms* of *ls* and *side* is a *hyponym* of *ls* therefore `point_1, point_2, m` are also *meronyms* of *side*. A figurative representation of some *meronymy-hyponymy* relations in *GeometryNet* is shown in Figure 3.2.

Hyponymy relations (↑) shown in Figure 3.2 implies that a *parallelogram* is a *superordinate* of *square* and *subordinate* of *polygon*. There are, however, other *subordinates* of *polygon* (e.g. *triangle, pentagon*) and *parallelogram* (e.g. *rectangle, rhombus)* which are not depicted in Figure 3.2. The citation of *square* and *triangle* in the *Net* are given below.

```
square
parallelogram
```

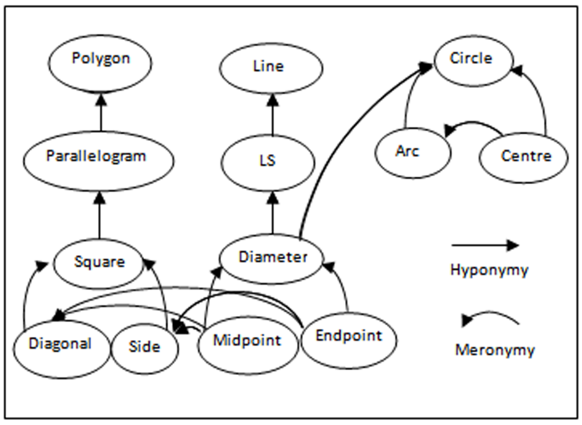


Figure 3.2: Network Representation of a set of *hyponymy-meronymy* relations in *GeometryNet*

```
#((x2-x1)^2+(y2-y1)^2)^0.5 =((x3-x2)^2+(y3-y2)^2)^0.5
#m1*m2=-1
#m2*m3=-1
#m3*m4=-1
#m4*m1=-1
end
```

This implies that a *square* is a *parallelogram* with length of `side_1` equal to that of `side_2` and the angle between every two adjacent *side* is 90° i.e. they are mutually perpendicular. This in turn implies that all the *sides* of a *square* are equal in length as from the definition of parallelogram we already get `side_1=side_3` and `side_2=side_4`. Figure 3.3 may be referred here.

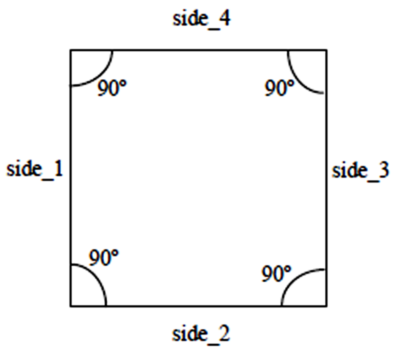


Figure 3.3: Square is a kind of parallelogram

For *triangle* we get:

```
triangle
n=3
vertex_1
vertex_2
vertex_3
side_1
side_2
side_3
#((((x2-x1)^2+(y2-y1)^2)^0.5)+(((x3-x2)^2+(y3-y2)^2)^0.5))>
(((x3-x1)^2+(y3-y1)^2)^0.5)
#((((x3-x1)^2+(y3-y1)^2)^0.5)+(((x3-x2)^2+(y3-y2)^2)^0.5))>
(((x2-x1)^2+(y2-y1)^2)^0.5)
#((((x2-x1)^2+(y2-y1)^2)^0.5)+(((x3-x1)^2+(y3-y1)^2)^0.5))>
(((x3-x2)^2+(y3-y2)^2)^0.5)
#m1!m2
#m1!m3
#m2!m3
end
```

Here it implies a *triangle* is formed by three non parallel line segments the sum of any two of which is always greater than the third side.

There are some nouns which basically denote an entity but also imply a particular relation between two entities. *Bisector* and *perpendicular* are examples of such nouns occurring in statements like - `A perpendicular AD is dropped from A on BC`; `M is the midpoint on the bisector XY of AB`. The *bisector* as defined in the *Net* is:

```
bisectorl
ls_1
#ls_2
#x2=(x3+x4)/2
#y2=(y3+y4)/2
end
```

Bisector can be related to a *line* or an *angle*. For geometric statements as above, `bisector XY of AB` indicates `XY` is the bisector of *ls* `AB`. An *l* is appended after the word *bisector* (*bisectorl*) to differentiate line bisector with angle bisector (*bisectora*). *bisectorl* is defined as being a line segment `ls_1` related to another line segment `ls_2` prefixed with #, such that the parameter condition prefixed with # holds i.e. `ls_2` is bisected at point_2. Figure 3.4 represents this definition of *bisectorl*. The more generalized definition of line bisection is, however, listed under verb *bisectl* where ls_1 bisects ls_2 at a point other than the one endpoint ((x2,y2) of ls_1.

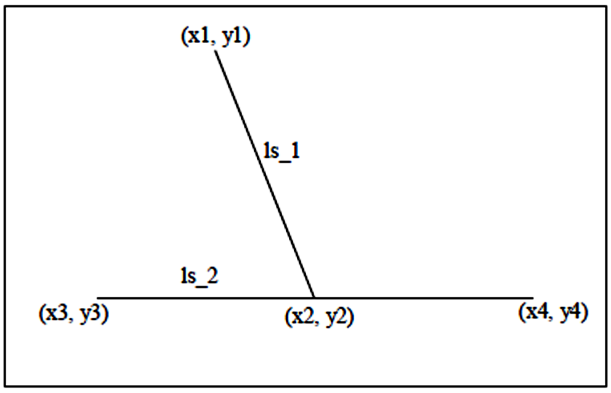


Figure 3.4: bisectorl: ls_1 bisects ls_2 at (x2,y2)

Similar to *bisectorl, perpendicular* is listed as noun as:

```
perpendicular
ls_1
#ls_2
#m1*m2=-1
#(y2-y3)/(y4-y3)=(x2-x3)/(x4-x3)
end
```

Here the expression #(y2-y3)/(y4-y3)=(x2-x3)/(x4-x3) implies one of the endpoint (x2,y2) of ls_1 is on ls_2 to which ls_1 is perpendicular. Figure 3.5 illustrates this.

There is no provision in *GeometryNet* for storing names of the entities or its parameters. Names are dynamically assigned according to the problem statement. When a statement says: ABCD is a parallelogram, a *parser* program correlates the linguistic information extracted from the statement and generic information stored in *GeometryNet* (for *parallelogram*) to make assignments as: vertex_1=point_1=(x1,y1)=A. Again corresponding to statements like - AB is the diameter of a circle having centre C, the interpretation is: one endpoint of the diameter is: A, the other endpoint is B , and the centre of the circle is C. This results into final assignments like A=(x1,y1), B=(x2,y2) and C=(xc,yc).

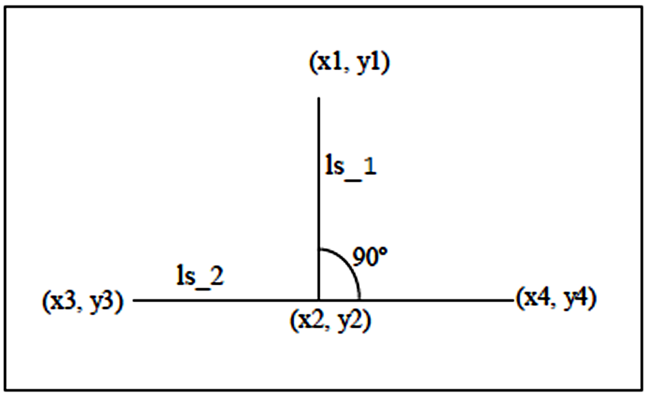


Figure 3.5: ls_1 is a perpendicular on ls_2 at (x2,y2)

We have discussed so far about the intrinsic features of geometric entities which are required to draw them and are related to their independent existence irrespective of other entities except few typical ones like *bisectorl, perpendicular,* etc. But geometric problems in general concern the interrelations and interactions between multiple entities described by adjectives and verbs. *GeometryNet* keeps the definitions of all such adjectives and verbs examples of which are illustrated in the following sections.

3.2.3 Adjectives

Adjectives are there to express qualification of a noun entity. For example, *common tangent, parallel line, equilateral triangle, collinear points,* etc. Words belonging to other syntactic categories can function as adjectives, such as present and past participles of verbs (e.g. *intersecting lines, inscribed polygon*), nouns (e.g. *right-angle triangle*), prepositional phrases (e.g. *triangles with common base*), and noun phrases (e.g. *the circles center*). Entire clauses can also qualify nouns as: *...the chord that passes through the center*. It can be observed that prepositional phrases and clausal noun modifiers follow the noun (entity) while noun phrases and single word adjectives precede the noun (entity).

Adjectives in geometric statements typically qualify a particular type of entity. For example the relevant adjectives for *triangle* are *right-angle, isosceles, equilateral, right-angle isosceles, similar, congruent, equiangular,* etc. These adjectives do not make any sense if applied to qualify entities other than *triangle*. Similarly adjectives *collinear, non-collinear,* *cyclic, concyclic, intersection* all are used to specify status of *points*. However some adjectives (e.g. *common*) are equally applicable for different types of entities. For example: *common chord, common base, common tangent*. Again an adjective applicable to an entity is by inheritance applicable to its *hyponym*, e.g. *parallel ls, parallel chords*; *chord* is a *hyponym* of *ls*.

Pure adjectives can be divided into two categories, namely *descriptive* and *relational*. Consider the following two geometric statements.

```
AB is parallel to CD
Two equilateral triangles ABC and BCD have a common base BC
```

Here the adjective `parallel`, qualifies *ls* `AB` and describes a particular geometric relation of `AB` with another *ls* `CD`. Similarly `common` qualifies `base` (i.e. *ls*) `BC` by refering to two *triangles* `ABC` and `BCD`. Such adjectives are termed as *relational adjective* in *GeometryNet* nomenclature. The adjective `equilateral`, on the other hand, describes a particular configuration of each of the *triangles* it qualifies. It does not point to other entities for the description. Such adjectives are termed as *descriptive adjective*.

Unlike in *WordNet* [19], in *GeometryNet* neither of these adjectives is organized in clusters of *synsets* related by anotonymy pointers (mentioned in Section 3.2.5). This is because the number of adjectives pertaining to *GeometryNet* is very low compared to that of *WordNet* and also because *antonyms* of entity adjectives (used in Geometry) are rarely found or used. Rather in *GeometryNet* both *relational* and *descriptive adjectives* refer to a mathematical condition (involving the relevant parameters), the principal entity (head noun) it qualifies, and the related entities if any. The *relational adjective parallel* is listed as:

```
parallel
ls_1
ls_2
#m1=m2
#(y2-y3)/(y4-y3)!(x2-x3)/(x4-x3)
end
```

Here `ls_1` is the principal entity described by *parallel*. The entity it relates to is `ls_2`. While `#m1=m2` depicts that the lines are *parallel*, the other # expression depicts that the lines are not *collinear*. Only *slopes (m)* being same does not explicitly represent the case (refer Figure 3.6(a)) we commonly mean by the term *parallel* because *slopes* are also same for two lines which are *collinear* (refer Figure 3.6(b)). In a statement like - `AB and CD is parallel`, there is no principal entity and `parallel` behaves as a *relational adjective* but it refers to the same definition in the *Net*.

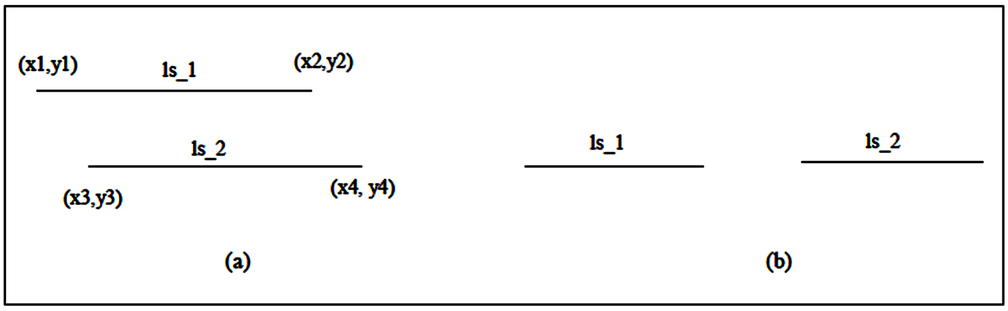


Figure 3.6: Parallel lines: (a) Non-collinear, (b) Collinear

Another similar example of *relational adjective* is '='. Though after initial processing of geometric statements '=' is tagged as symbol, it is interpreted as adjective *equal* - there may be noun arguments on both side of '=', e.g. `DC=CQ`. This implies `DC` is equal to `CQ` and thus the adjective *equal* qualifies the *ls* `DC` and relates it to the other *ls* `CQ`. Like *ls* two *angles* (or *points*) can also be equal, e.g. $\angle ABC = \angle DEF$. A differentiator suffix the adjective *equal* to indicate which nouns it describes. So *equall* concerns two *ls*, *equala* concerns two *angles* whereas *equalp* relates two *points*.

```
equall
ls_1
ls_2
#((x2-x1)^2+(y2-y1)^2)^0.5=((x4-x3)^2+(y4-y3)^2)^0.5
end
```

Here the # statement implies that the length of `ls_1` and `ls_2` are equal.

An example of *relational adjective* whereby more than two entities are connected with the principal entity is *common*. When it qualifies a *tangent* (refer Figure 3.7) the syntax for listing *common* is:

```
commont
tangent
circle_1
circle_2
#(x1-xc1)^2+(y1-yc1)^2=(r1)^2
#(x2-xc2)^2+(y2-yc2)^2=(r2)^2
end
```

It is evident from the above quadratic # expressions that for given two *circles* there may be four possible *common tangents*.

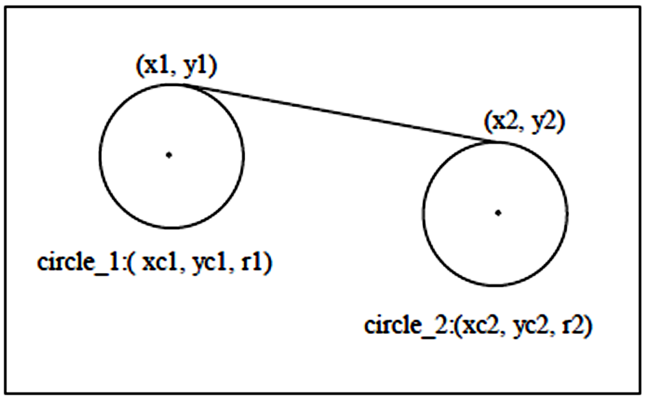


Figure 3.7: Common tangent of two circles

When *common* qualifies a *chord* (refer Figure 3.8) the listing is different under *commonc*.

```
commonc
chord
circle_1
circle_2
#(x1-xc1)^2+(y1-yc1)^2=(r1)^2
#(x1-xc2)^2+(y1-yc2)^2=(r2)^2
#(x2-xc1)^2+(y2-yc1)^2=r1^2
#(x2-xc2)^2+(y2-yc2)^2=r2^2
end
```

In both the cases shown above for *relational adjective common*, the basic entity (i.e. the *superordinate*) it refers to is *ls* which is retrieved with parameters `(x1,y1)` and `(x2,y2)` and other # conditions while searching for information about *tangent* or *chord* separately listed as nouns.

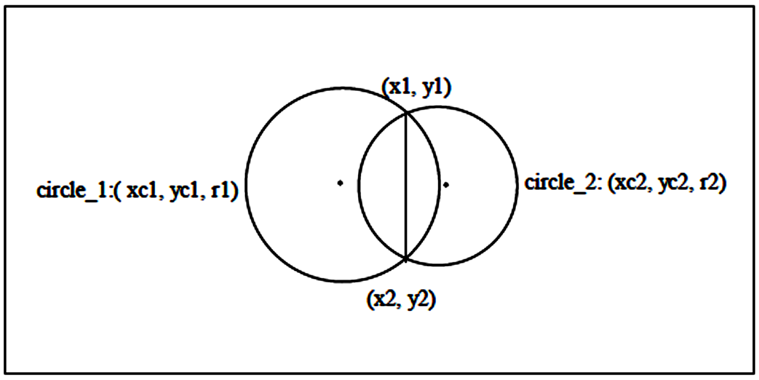


Figure 3.8: Common chord of two circles

equilateral is an example of a *descriptive adjective* that describe a *triangle* necessitating no reference to any other entity.

```
equilateral
triangle
#(x1-x2)^2+(y1-y2)^2 =(x1-x3)^2+(y1-y3)^2
#(x1-x2)^2+(y1-y2)^2 =(x2-x3)^2+(y2-y3)^2
end
```

3.2.4 Verbs

Verbs are arguably the most important lexical and syntactic category of a language. Hardly any sentence can be found where there is not a single verb; at least there should be an auxiliary verb like *is, are, be, has* and *have*. In geometric problems, verbs generally express relationship between two entities. For example:

```
AX when produced meets CD at Q; The bisectors of AB and AC
intersect at O.
```

While language processing, mostly verbs are used as *connectors* to connect different isolated graphs representing many entities along with their attributes. Thereby the different geometrical parts are related to each other from which the complete meaning or configuration of a problem is understood.

Verbs usually exhibit higher *polysemy* than the other POS thereby suggesting that same verb has different meanings depending on the noun arguments with which they co-occur. Accordingly in *WordNet*, verbs are divided into 15 files each representing different semantic domain. But as far as geometry problems are concerned *polysemy* is hardly displayed by any verb. In *WordNet* [20] similar meaning verbs are organized into *synsets* denoted by { }. In *GeometryNet*, there are verbs having similar meaning like *meet-cut-intersect, join-connect, expand-extend-produce, draw-describe-drop,* etc.; only generic information is stored against any one verb in a set, the other verbs inherit those information by simply referring that verb. Some of the verbs occurring in geometric statements do not have any synonyms, e.g. *stand, lie, touch, pass, divide, subtend, center,* etc. There are few verbs that do not contain any geometric information as such, e.g. *is, was, prove, find, show, calculate, completed,* etc. and hence are ignored in *GeometryNet*.

Like the adjectives, the listing of verbs in the *Net* contain principal entity (noun) along with other related entities. Such entities or their *hypernyms* are used as arguments of the verbs (*connectors*). An *ls* can *cut* (or *meet* or *intersect*) another *ls* or *circle* usually. Two *circles* can also *cut* (or *meet* or *intersect*) each other. To specify the *meets* verb as *connector*, the noun arguments, i.e. the *ls* or *circle* should be used. Depending on the argument type the *GeometryNet* listing of *meets* is done in the form of *meetsll* (implying *ls* meets *ls* as illustrated in Figure 3.9), *meetslc* (implying *ls* meets *circle*) and *meetscc* (implying *circle* meets *circle* as illustrated in Figure 3.10).

```
meetsll
ls_1
ls_2
#m1!m2
#x2=x4
#y2=y4
end
```

Here it implies that two intersecting *ls* should not be parallel plus the intersection point must satisfy the equation of both the *ls*. Similarly, *meetscc* is defined as:

```
meetscc
circle_1
circle_2
point_1
point_2
#((xc1-xc2)^2+(yc1-yc2)^2)^0.5=(r1+r2)
```

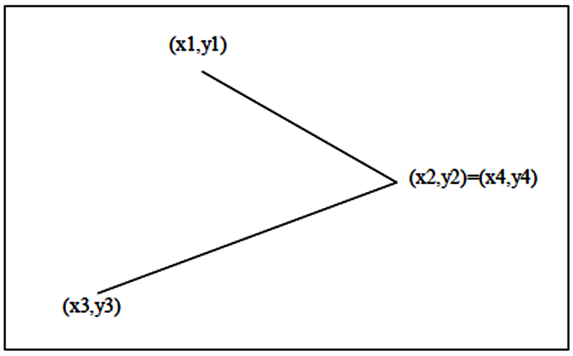


Figure 3.9: Intersecting lines

```
#(x1-xc1)^2+(y1-yc1)^2=r1^2
#(x2-xc1)^2+(y2-yc1)^2=r1^2
#(x1-xc2)^2+(y1-yc2)^2=r2^2
#(x2-xc2)^2+(y2-yc2)^2=r2^2
end
```

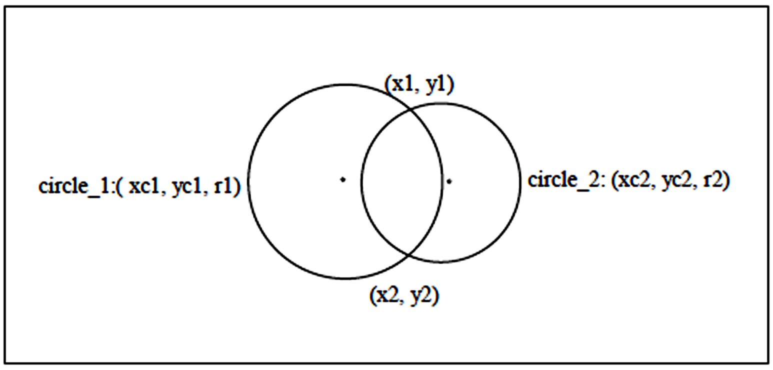


Figure 3.10: Intersecting circles

touch is another verb for which the way how the entities interact (touch) with each other is important unlike the *meets* verb where which two entities interact (meet) is important. *touch* is usually applied for circles that can touch each other *externally* (Figure 3.11(a)) or *internally* (Figure 3.11(b)). The parametric conditions are different for these two cases which are separately listed in *GeometryNet* under *touche* and *touchi* and referred when the qualifying adverb of *touch* in geometric statements is *externally* and *internally* respectively.

```
touche
circle1
circle2
point_1
#((xc1-xc2)^2+(yc1-yc2)^2)^0.5=(r1+r2)
#(x1-xc1)^2+(y1-yc1)^2=r1^2
#(x1-xc2)^2+(y1-yc2)^2=r2^2
end

touchi
circle1
circle2
point_1
#((xc1-xc2)^2+(yc1-yc2)^2)^0.5=|r1-r2|
#(x1-xc1)^2+(y1-yc1)^2=r1^2
#(x1-xc2)^2+(y1-yc2)^2=r2^2
end
```

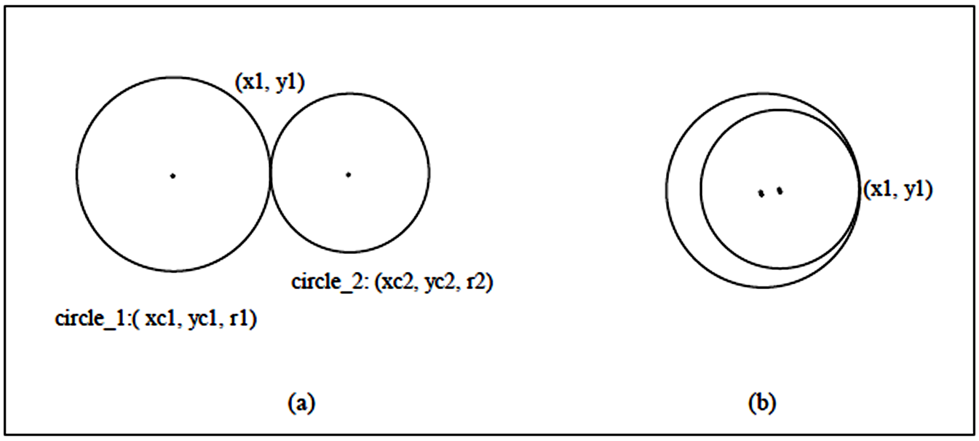


Figure 3.11: Two circles touching each other: (a) externally, (b) internally

As like when used as a noun (*bisector*), the concept of bisection is also referred to when used in the geometric statements as verb *bisects*. When the argument of *bisect* is *ls* then it means *intersects* but when the argument is *angle* it means *divide*. So verb *polysemy* is displayed here. Again looking in the other way *troponymy* is also displayed by *bisect* because *to bisect (ls)* is *to intersect (ls)* in a particular manner. Figure 3.12 can be referred here. Again *to bisect (angle)* is *to divide (angle)* in a particular manner. Figure 3.13 illustrates this. The particular manner is expressed by specific parameter condition (#

expressions) under *bisectl* and *bisecta* respectively, The basic concept of line intersection is inherited by *bisectl* by referring to verb *intersectll* from within *bisectl*.

```
bisectl
intersectll
#x5=(x1+x2)/2
#x5=(x3+x4)/2
#y5=(y1+y2)/2
#y5=(y3+y4)/2
end

intersectll
ls_1
ls_2
point_5
#y5-m1*x5=y1-m1*x1
#y5-m2*x5=y3-m2*x3
end
```

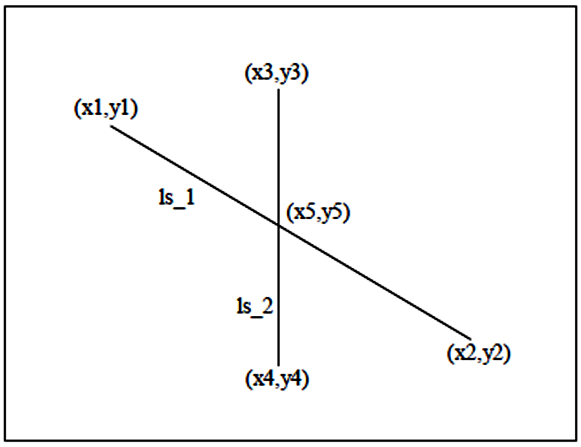


Figure 3.12: Line bisection as a special case of line intersection

The concept expressed by the verb *bisectl* is different from that expressed by noun *bisectorl* - the former concerns division of both the *ls* into two equal parts while the later divides only one *ls* into two equal halves. But the verb *bisecta* and the noun *bisectora* both expresses the same concept of dividing an *angle* into equal two smaller *angles*. The information stored under *bisecta* is:

```
bisecta
angle_1
ls_3
angle_2
angle_3
#angle_2.angle=angle_1.angle/2
#angle_3.angle=angle_1.angle/2
#point_2=point_4
#point_2=point_7
#point_2=point_10
#point_1=point_6
#point_5=point_8
#point_5=point_9
#point_3=point_11
end
```

Here the # statements stating that two *points* are equal automatically results in equality of the corresponding x and y coordinates, e.g. `#point_2=point_4` causes `#x2=x4` and `#y2=y4`.

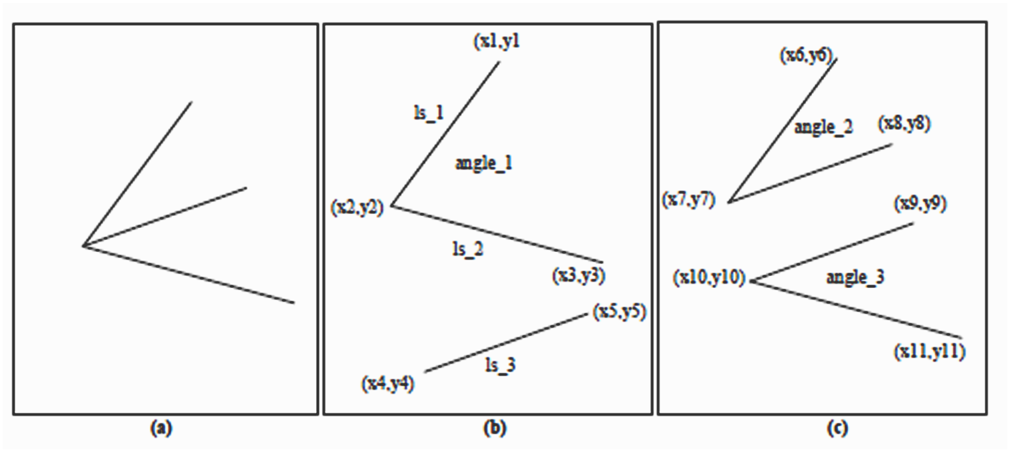


Figure 3.13: (a) Bisection of an angle, (b) the angle is angle_1 and the bisector is ls_3, (c) equal angles, angle_2 and angle_3 formed after bisection

The verbs *produce* (*produced*) and *extend* are quite often used in geometric problems. As both convey the same concept (refer Figure 3.14), only *produce* is defined; *extend* simply refers to *produce* to inherit its characteristics.

```
produced
ls_1
point_3
```

```
#y3-m1x3=y1-m1x1
#m2=m1
end

extend
produced
end
```

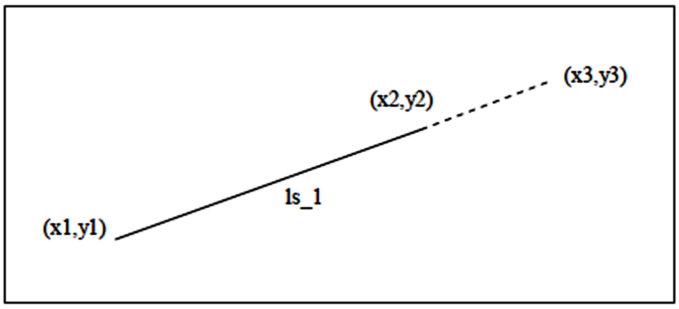


Figure 3.14: ls_1 produced or extended to point_3=(x3,y3)

3.2.5 Implementation Issues

Following the model of *WordNet* [85] initially we proposed specific file formats for *GeometryNet* database. For each syntactic category we used two files namely *index file* and *data file* to represent the *Net*. Both the files were in ASCII format. Each *index file* was an alphabetized list of all of the word forms for the corresponding syntactic category along with its starting address or equivalent byte offset in the corresponding *data file*. Each *data file* contained word forms or *synsets* and under each *synset* was listed the target words or *synset* pointed to (marked by the *relational pointers*) and other information. *Relational pointers* are symbols used to imply relations like *hypernymy, meronymy, troponymy, antonymy,* etc. as shown below.

bisector @ $\rightarrow$ line

side #3 $\rightarrow$ triangle

bisect ~$\rightarrow$ intersect

circumscribe ! $\rightarrow$ inscribe

The *relational pointers* @ $\rightarrow$ implied *hypernymy*, i.e. `bisector` is a *subordinate* of `line`, #3 $\rightarrow$ implied *meronymy* i.e. `triangle` has 3 `sides`, ˜ $\rightarrow$ implied *troponymy* i.e. to `bisect` is to `intersect` and ! $\rightarrow$ implied *antonymy* i.e. `inscribe` is conceptually opposite to `circumscribe`. Two typical entry in the *data file* for nouns was:

```
bisector @→ ls, ls_1,ls_2, x2=(x3+x4)/2, y2=(y3+y4)/2
center @→ point, #s circle
```

The similar meaning verbs were organized into *synsets* denoted by { }. For example, {meet, cut, intersect}, {join, connect}, {expand, extend, produce,} etc.

After some trial runs, it was felt to further simplify the structure of the *Net* towards optimization of processing or search time. Hence, in the later version of the *GeometryNet* some changes were made. Instead of having *data file* and *index file* for each POS category, a single text file is now used containing all the geometric terms in alphabetic order in separate blocks pertaining to noun, adjective, and verb. Description of each term is delimited by an *end* statement as shown in the examples in Sections 3.2.2, 3.2.3 and 3.2.4. No *relational pointers* or *synsets* are anymore used. The concept of inheritance is retained by simply including reference of related noun or adjective or verb within the definition of a term and suitably modifying the search program that mines knowledge about a geometric term from the *Net*. The *parser* programs, while scanning the entries under a search item (geometric term), would automatically point to the referred noun or adjective or verb and extract the relevant information from their original location. In Section 3.2.2, we have shown the entries made under different nouns. The reference to *ls* is automatically made when information regarding *side* is searched for in the Net, no *hypernymy* pointer (@ $\rightarrow$) is additionally required to make the program point to the original entry of *ls*. Similarly, for *triangle* or *parallelogram*, it is not an absolute necessity to use *meronymy* relational pointer (# $\rightarrow$) to indicate that there are 3 and 4 *sides* as part of a *triangle* and *parallelogram* respectively. It is also evident from the definition cited in Section 3.2.4 for *bisectl* and *intersectll* that *troponymy* pointers (˜ $\rightarrow$) are not required to store the concept that *biscetl* is a special case of *intersectll*. It is observed that such organization of the geometric terms have improved the performance of the system compared to the initial design of *GeometryNet*.

Positions or sizes of geometric entities like *line, circle, triangle, parallelogram,* etc. are seldom mentioned in a school-level geometric problem - most are generalized statements and the students are free to draw figures at different size or orientation provided the specific relations as described in the problem and basic geometrical concepts are not violated. Intuitively, given a geometric problem, we tend to draw some standard or easily draw-able figures. For example, we draw a horizontal *straight line* to represent a statement like - `AB is a straight line`; we draw an *equilateral triangle* with a horizontal base to represent a `triangle ABC`, a *quadrilateral* with horizontal and vertical *sides* to represent a `rectangle ABCD` and likewise. To follow this human process of drawing standard figures of geometric objects, we have kept provision for storing default numerical values of defining parameters of different geometric entities, e.g. coordinates of *vertices* of a *parallelogram*, *center* and *radius* of a *circle*, *endpoints* of a *line segment (ls)*, etc. Such values are used to instantiate generic variables representing entity parameters (normally listed under each geometric entity or noun) to draw default figures when the concerned entity is not constrained by the relations mentioned in the problem description. Such default numerical values are stored separately in *GeometryNet* as shown in Table 3.1.

Table 3.1: Default parameter values stored in *GeometryNet*

parallelogram	vertex_1, vertex_2, vertex_3, vertex_4	(50,200), (100,100), (300,100), (250,200)
triangle	vertex_1, vertex_2, vertex_3	(100,300), (50,100), (150,100)
ls	point_1, point_2	(50,100), (150,100)
circle	center_1, radius_1	(100,100), 50
---	---	---
---	---	---

Let us consider the following example problem.

`In` $\Delta PQR, \angle PQR = \angle PRQ$`. QR is produced to M and N. Prove that angle MQP=angle NRP.`

In this problem `PQR` is any *triangle* with base *angles* same. To draw the figure we can use the default numerical values of the *vertices* of a *triangle* stored in Table 3.1 because it represents an *equilateral triangle*.
Again consider the following problem.

```
Triangle ABC is circumscribed about a circle with centre I. The
sides AB, BC and CA touch the circle at points D, E and F
respectively.  If AD=16 cm, CF=9 cm, BE=7 cm then find the sides
of the triangle.
```

Here we cannot draw *triangle* `ABC` with stored default values of the *vertex* coordinates because the length of the *sides* are constrained by the values and relations described in the problem. We have to solve the variables representing the *vertex* coordinates from the equations describing the relations.

As we developed the working modules of the t2d system, requirement for storing additional information (other than the geometric terms and default values) in the *Net* was felt and the Net was expanded accordingly. To help the text processing task, an array like Table 3.2 was introduced that stored all possible geometric entities and the number of points necessary to represent them. This is required to identify the entity type (common noun) from its name (proper noun) when it is not explicitly mentioned in the problem. For example, the problem statement, `X is the midpoint of BC` does not explicitly mention what `X` and `BC` are. From Table 3.2 it is interpreted that a single alphabet proper noun like `X` stands for a *point* while `BC` is a *ls* as it is an entity name with two alphabets.

Meaning of some symbols that are frequently used in school-level geometry problems are also stored in *GeometryNet* (Table 3.3). These symbols are used instead of some typical common nouns or adjectives and thus play important role in defining the entities

Table 3.2: Entity-wise number of definition points

point	1
ls	2
triangle	3
angle	3
arc	3
parallelogram	4
square	4
---	--

or relations. For example, $\angle ABC = \angle XYZ$ implies `angle ABC is equal to angle XYZ`.

Table 3.3: Storing geometry symbols

$\triangle$	triangle
$\angle$	angle
$\parallel$	parallel
$\perp$	perpendicular
=	equal
--	---
--	---

3.3 Summary

We have tested our system with around 40 school-level geometry problems covering most of the geometric entities and their different attributes/relation and interaction situations. Except few error cases, the system generated diagrams successfully. The errors as reported in Chapter 4 are mostly in the language processing part and are hardly due to any information deficiency of *GeometryNet*. However, while testing new problems new situation may be encountered necessitating some additional information to be stored or some change of format of information already stored in *GeometryNet*. The simplicity of the design of *GeometryNet* gives the flexibility to modify the *Net* towards developing it as a more complete knowledge base for geometry domain.

Till date, the scope of our system and also *GeometryNet* has been limited only to the domain of school-level geometry problems that deal with common geometric entities.

It does not deal with complicated (higher level) geometry such as menstruation, 3D geometry, solid geometry, and geometric calculus. We expect to extend or recreate the knowledge base to incorporate processing of problems or texts pertaining to the above mentioned fields in future.

Going beyond geometry our research can be extended to address similar diagram-based problems in other subject domains, e.g. physics, chemistry or civil, mechanical, electrical, electronics engineering, etc. if suitable domain-specific knowledgebase similar to *GeometryNet* can be developed. First, we have to identify typical entities/elements and concepts pertaining to a given domain.

The drawing elements like pulley, incline, mass, weight, etc. for physics/mechanics diagrams (e.g. Figure 3.15), resistance, capacitor, EMF source, wire, etc. for electrical/electronic diagrams (e.g. Figure 3.16(a)), and door, window, wall, etc. for civil engg. diagrams (e.g. Figure 3.16(b)) mostly comprise basic geometric entities like line, circle, arc or a combination of such entities. The properties and features of such elements need to be identified in the context of the subject and level we like to address. Then following the same approach as described in this chapter, generic information and default value can be stored to yield simple knowledge base of different technical terms of a given subject domain. Eventually, electrical or electronic circuits, building diagrams, free body diagrams, etc. can be generated from problem texts (that describe such circuits or diagrams) using respective knowledge base. The feedback from our current experiments can be used in effectively developing such knowledge base.

In the next two chapters the use of *GeometryNet* is discussed. Chapter 4 shows the utility of the *Net* in linguistic analysis of a problem text while Chapter 5 illustrates why the *Net* is indispensable in machine understanding of a problem and subsequent generation of geometric diagrams.

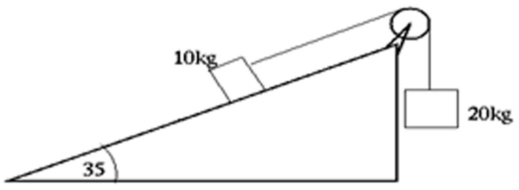
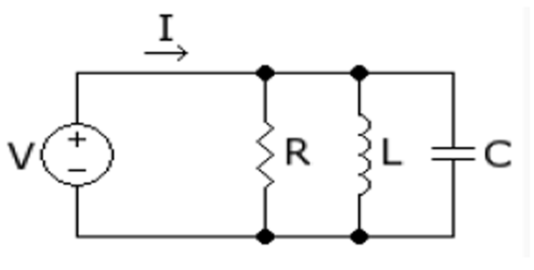
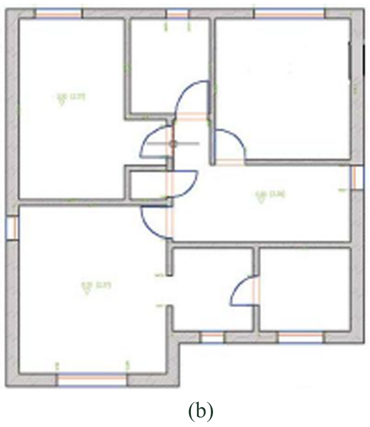


Figure 3.15: Diagram representing the physics problem: *Given an incline with angle 35 degrees which has a mass of 10 kg placed upon it. It is attached by a rope over a pulley to a mass of 20 kg which hangs vertically. Taking downward as the positive direction for the hanging mass, calculate the acceleration of the masses.*



(a)



(b)

Figure 3.16: (a) An electrical/electronic circuit diagram, (b) A floor plan (civil) diagram

CHAPTER 4

Language Processing in Automatic Diagram Drawing

4.1 Introduction

This chapter describes the *Natural Language Processing (NLP)* module [8] of our system which processes the text of the input geometry word problem and translates it into a language-free formal representation. Though *GeometryNet* [11] acts as the backbone of our system, the *parser* belonging to the NLP module is the most difficult to generalize and hence it is the most important part of the system. The success of the whole system depends on the correct interpretation of the problem text which is often ambiguous and incomplete as far as meaning of the text is concerned. The major challenges faced while developing this module are: (i) there being no fixed structure of data input the user is free to state a given problem in different ways (sentences or phrases), (ii) a word or a phrase can bear different meaning in different context, and (iii) complete geometric information is often not provided in a school-level geometry word problem. It is the human experience and expertise in the domain of a given subject (here geometry) and natural language (English) that helps to figure out the essence of a word problem out of all lexical complexity. The *parser* is developed to follow this logical and knowledge-based human approach so as to analyze a problem-text, extract the useful parts, and correlate the parts to generate a language-free technical summary for further processing, i.e. machine understanding and diagram drawing.

Figure 4.1 shows a schematic diagram of the overall system. The problem statements are fed into the first level *parser*, i.e. *Parser 1* [8] 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 (POS) tagger*) and the *GeometryNet* knowledge base. Once the *parser* provides a formal description of the input statements, 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 as another language independent, unique, unambiguous representation of the input problem. A second *parser*, i.e. *Parser 2* [8, 7] acts on this *intermediate representation* of the problem text, aided by the *GeometryNet* [11] knowledge base, to generate a draw-able representation. A graphical module [7] can then generate the required diagram by calling a set of graphical functions. The dotted portion in Figure 4.1 shows the language processing part (of the overall system) that this chapter deals with.

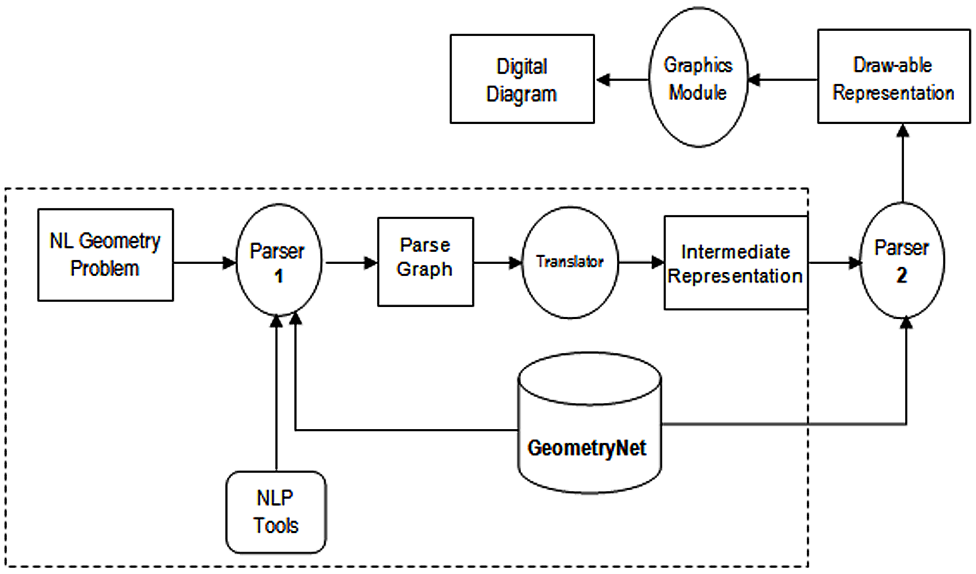


Figure 4.1: The schematic diagram of the overall text-to-diagram conversion system

As the formal representation of the input problem largely dictates the success of the overall system, *Parser 1* [8] is considered as the most important module of the system. This chapter describes the design methodology of *Parser 1* [8] and the associated *transla-*

tor [8]. The parsing approach can be stated in terms of following six steps:

- **Step 1:** Give a geometry word problem as input to a *POS tagger*; the output is tagged problem statements.
- **Step 2:** Take a tagged problem statement and break it into lines. Extract all geometrical entities in the problem statements along with their types and special attributes (if any).
- **Step 3:** Decompose the entities into atomic ones with the help of which the parent entity may be structurally defined. Generate a *parse graph* with the entities (parent and decomposed) as nodes. Use the parent-child relationships to connect the nodes in the graph.
- **Step 4:** Using the problem statements that had been broken into lines, extract *connectors* which connect the various entities in the problem so that the information within the problem statements remain preserved.
- **Step 5:** Incorporate the *connectors* into the graph generated in Step 3 as nodes. The graph is language-free i.e. free from language constructs.
- **Step 6:** Scan the graph obtained and generate an unambiguous, language-free representation of the problem using the *translator*.

4.2 Method

The details of each of the above mentioned steps are illustrated here with a typical example.

Step 1: The input to our system is a school level geometry problem like the following one.

```
ABCD is a parallelogram and X is the midpoint of BC. The line AX
produced meets DC produced at Q. CQ is extended to P and the
parallelogram ABPQ is completed.  Prove that DC = CQ = QP.
```

The *Brill POS tagger* [36] is used to tag the problem statements as follows.

```
ABCD (NNP) is (VBZ) a (DT) parallelogram (NN) and (CC) X (NNP)
is (VBZ) the (DT) midpoint (NN) of (IN) BC (NNP). (.)  The (DT)
line (NN) AX (NNP) produced (VBN) meets (VBZ) DC (NNP) produced
(VBN) at (IN) Q(NNP).(.)  CQ (NNP) is (VBZ) extended (VBN) to
(IN) P and (CC) the (DT) parallelogram (NN) ABPQ (NNP) is (VBZ)
completed (VBN).(.)  Prove (VB) that (IN) DC (NNP) = (SYM) CQ
(NNP) = (SYM) QP (NNP).(.)
```

Step 2: This step consists of following sub-steps.

2a. The tagged problem is broken into lines. 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. At this stage the above problem will be broken as shown in Table 4.1.

Table 4.1: Tagged problem text is broken into lines

ABCD(NNP)is a parallelogram(NN) .(.)and (CC)
X(NNP) is(VBZ) the(DT) midpoint(NN)of(IN) BC(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)
parallelogram(NN) ABPQ(NNP) is(VBZ) completed(VBN) .(.)
Prove(VB) that(IN) CD(NNP) =(SYM)CQ(NNP)=(SYM)PQ(NNP) .(.)

As in Table 4.1, the first and third lines in the problem are broken into two as *and* form a connection between two complete sentences. But several cases arise when *connectors* like *and* or *comma* (,) appear in a problem statement.

- If an *and* or a *comma* (,) separates two complete sentences:

In such a situation the line will be broken into two as shown for the example in Table 4.1.

- If an *and* or a *comma* (,) 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 *comma*s (,) encountered do not join two different sentences as in the example in Table 4.1; rather, they separate more than one NNPs of the same type. If such lines appear in a problem statement, the division discussed previously will not take place.

- If an *and* or a *comma* (,) separates two different types of proper nouns (e.g. 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 *comma* (,) lies between two different types of 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 *comma* (,) separates two equations:

They will be divided into two lines. For example, `AB=BC` and `CD=DF` will call for a division at *and*.

2b. At this stage the task 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 `ABCD, X, BC, AX, Q, CD, CQ, P, ABPQ` are the NNPs and `parallelogram, midpoint` and `line` are the only NNs mentioned in the problem. From this information, we can construct a table partially as (Table 4.2).

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

Table 4.2: Partial entity table

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

particular entity, if mentioned in the problem statement. For example, isosceles, equilateral are special types of triangles which build on the special features of a triangle.

2c. The task at this step is to recognize the types of the remaining NNPs (other than `ABCD, X, AX` and `ABPQ` in the above problem) whose corresponding common nouns are not mentioned explicitly in the problem statement. Here the table (Table 3.2 of Chapter 3) is referred that stores all possible geometric entities and the number of points that are used to represent them.

The number of points is counted in the proper noun (`ABCD` for example has 4 points) whose NN field is blank in Table 4.2. 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 entry in the Table 3.2 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, from Table 4.2, the completed entity table (Table 4.3) will be generated.

Sometimes we may encounter NNs without an NNP mentioned in a problem statement, e.g. `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 4.4.

It may be noted that the NNP for *bisector* is blank in the above case. For such cases we

Table 4.3: Complete entity table

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

Table 4.4: Example of NN without NNP

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

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* 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 table thus generated from Table 4.4 is Table 4.5.

Table 4.5: Assigning NNPs to NNs without name

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

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

Step 3: This is the most important step whereby a graph is generated from the entity table. The graph formation process is actually divided into two steps - Step 3 and Step 4.

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, e.g. 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 about the line `AB` and line `BA`, or about the parallelogram `ABCD` and the parallelogram `CDAB` 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 represented in Table 4.3 the entities after canonicalization and removal of repetitions are: `ABCD, AB, BC, CD, AD, A, B, C, D, X, AX, Q, CQ, P, ABPQ, BP, PQ, AQ` and the graph formed until now is as shown in Figure 4.2.

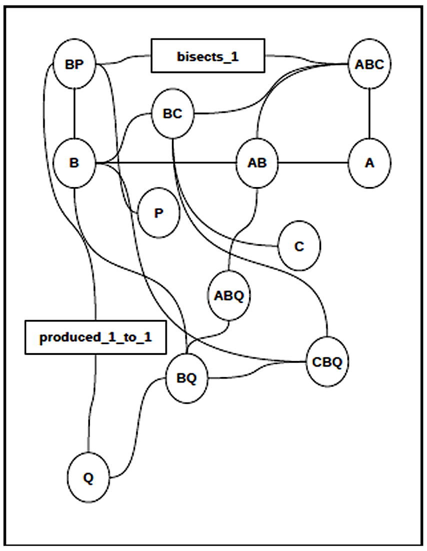


Figure 4.2: Graph produced in step 3

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, the disconnected graph may not lead to the generation of correct diagrams. This leads to the formulation of the next step.

Step 4: This is the final step of the parsing method. The goal of the 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.

Usually, in geometry word problems, the terms or phrases that establish a relation between two entities are: *of, in, produced, has, extended, meets*, etc. Our design is flexible enough to add more such terms as and when they appear. In general, such terms are called as *connectors*. The detection of *connectors* in our approach is a rule based process. The POS tags against the words in the problem statements are used. We need to scan the problem once again. 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.
- Also there are certain storage rules that are followed strictly for refinement of idea about the problem. Many words, which are stored under the purview of these

rules, may not at all appear in the final output. However, 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 array for correct interpretation.

Following processing of Step 4 the Table 4.6 is produced from Table 4.3.

Table 4.6: The entity-connector table for the problem in Table 4.1

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, *connectors* with the same name 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 a provision in our method to assign default names as explained before. These default values are then linked to the NNs in a problem statement. This is helped by the fact that all the entities are stored in Table 4.3 in the order that they occur in the problem. Therefore, in order to construct the *entity-connector table* (Table 4.6), the complete *entity table* (Table 4.3) and the tagged statements as in Table 4.1 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 Table 4.7.

Table 4.7: Entity-connector table for an NNP-less statement

`DE (bisector)`	`FG (bisector)`	`of_1`	`AB line`	`AC line`	`intersect_1`	`at_1`	`O point`

Step 5: In this step *connectors* are included in the graph using the table produced in the last step. The cardinality of a *connector* is determined as follows:

1. One to one, e.g. `BC` (bisector) `of_1 PQR` (angle)
2. One to many, e.g. `O` (point) `lies_1 on_1 AB` (line) `CD` (line)
3. Many to one, 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 from the *connectors* and appropriate edges created in the graph. However, while creating the edges, the four cases above must be taken care of. Case 1 is trivial. The NNP on the LHS is connected to the NNP on the RHS through the *connector*. For case 2, each NNP on the LHS is connected to the corresponding NNP on the RHS through the *connector*. For case 3, the NNP on the LHS is connected to each NNP on the RHS through the *connector*. For case 4, the NNP on the RHS is connected to each of the NNPs on the LHS through the *connector*. Whether multiple copies of the *connector* is made or not for case 2 will depend on the implementation. The resulting graph from Figure 4.2 and Table 4.6 is shown in Figure 4.3 (the *connectors* are shown inboxes).

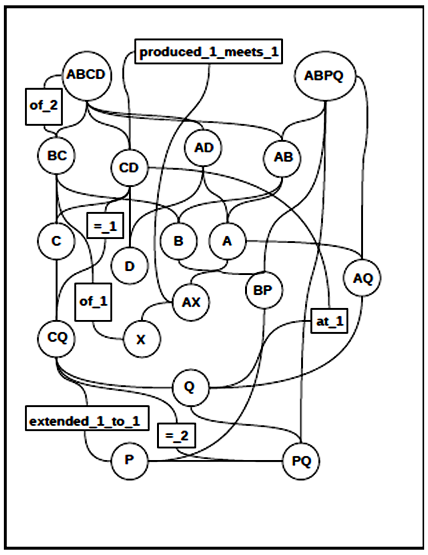


Figure 4.3: The final graph corresponding to Figure 4.2

Finally, it may be noted that while parsing a statement like `...`α `of` β `of` γ `meets` η `at` δ`...`, the connection is made as: α `---` `meets_1_at_1` `---`δ instead of γ and δ being connected through meets at δ. For example, a statement like `The bisector BE of ∠B of △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 `BE` being connected through *meets at*.

Step 6: Step 1-5 describes the parsing approach. *Parser 1* generates parse output in the form of a graph. The graph can be considered as a formal representation of an input problem. 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, a *translator* is designed 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. Step 6 is about translation of the *parse graph* into an *intermediate representation*.

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 will be represented as *attr(NN_1=NNP)*. The entities are then merged according to the relations between them. If two entities are connected with *conn* then, in the summary, it appears as:
conn_1(attr1(NN1_1=NNP1), attr2(NN2_1=NNP2))
For example, 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 represents a sense of belonging of one entity to another then it is represented as:
attr1(NN1_1=NNP1).attr2(NN2_1=NNP2)
For example, `X is the midpoint of BC` is represented as:
`(midpoint_1=X).(side_1=BC)`
For the example problem in Table 4.1, the summary generated from the graph in Figure 4.3 is:

```
parallelogram_1=ABCD
line_1=AB
line_2=BC
line_3=CD
line_4=AD
point_1=A
point_2=B
point_3=C
point_4=D
midpoint_1=X
line_5=AX
point_5=Q
line_6=CQ
point_6=P
parallelogram_2=ABPQ
line_7=BP
line_8=PQ
line_9=AQ
(midpoint_1=X).(line_2=BC)
(line_2=BC).(parallelogram_1= ABCD)
produced_1_meets_1(line_3=CD,line_5=AX)
at_1(line_3=CD,point_5=Q)
```

```
extended_1_to_1(line_6=CQ,point_6=P)
=_1(line_3=CD,line_6=CQ)
=_2(line_6=CQ,line_8=PQ)
```

4.3 Utility of the Intermediate Representation

In this section, we discuss about the utility of the *translator* [8] in the context of the t2d 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 t2d conversion. Let us look at the properties of the summary generated above.

- 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 like 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 4.5. 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 proposed 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* [7] in Figure 4.1) 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.

4.4 Analysis of the Method

Here we attempt to derive a complexity analysis of our parsing approach. We take up each step as described in Section 4.2 and discuss the complexity. The generation of Table 4.3 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 4.1). When a comma (,), a full stop (.) or the word *and* (which doesnt 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 (n) in the problem statement.
- The second step is to identify the words which can be classied 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 in n.

- 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 lookup 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* 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. It is a fair assumption to make that only a small fraction of the words in a problem statement will be entities without a name. A lesser number will be plural entities.

Generation of the intermediate graph as in Figure 4.2 involves:

- Decomposing each entity into atomic entities requires processing of each vertex of each entity in Table 4.3. This step is therefore linear in terms of the number of vertices in each entity in Table 4.3, which will be less than n 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 4.3, which is bounded by n.

- Now each canonical entity must be checked against existing nodes to prevent duplication. This is done in constant time using hash technique.

- Next, the nodes are put into parent-child relationships to form the intermediate graph as in Figure 4.2. This step is linear in terms of the number of nodes in the graph in Figure 4.2. 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 *n*.

The generation of the *entity-connector table* (Table 4.6) and including the *connectors* in the *parse graph* broadly involves the following steps:

- *Storing the geometrical entities (with tags NNP, NN, CD)*: In this phase, the problem statement (broken into lines) is scanned line-by-line. Whenever a geometrical entity or a constant term is encountered, the program stores it in a table (Table 4.6 in this case). So, the parsing complexity varies linearly with *n*.
- *Storing the connectors* (with tags VBZ, VBN, JJ, SYM): In this phase too, the problem statement is scanned line-by-line. 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 *n*.
- *Finding common nouns for the entities stored in the table*: After all 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 Table 4.3 generated in Step 4 to find the corresponding common nouns. It requires scanning every row of Table 4.3 to find such proper nouns and then for each one of them, scanning every row of Table 4.3 to obtain its matching common noun. Considering the number of rows in Table 4.3 to be much less than *n*, we find that the algorithms 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 4.6. Each row of Table 4.6 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 (Table 4.3), which has an upper bound of n.

- The step to include the *connectors* in the graph requires parsing of entity-connector table (Table 4.6) as many times as the number of *connectors* present in the table. This step is therefore linear in terms of the number of entries in Table 4.6 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 in the number of words n in the input problem statements.

4.5 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 a form compatible to the graphic functions. Therefore, our evaluation strategy consists of two steps: (i) evaluate whether the 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.

4.5.1 Test Set and Evaluation Strategy

We develop a test data set for evaluating the *parser* and the *translator*. The test set consists of 40 geometry problems taken from school level mathematics books of grades VIII, IX and X 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. one which generates the same final diagram) may be narrated differently in different books. Such problems are also considered to check uniqueness of the method. It may be noted 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 groundtruth 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 output of the *translator* 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 (i.e. ascertained 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 32 out of 40 problems. Some of these results and related discussions are presented in the next section. Errors for processing the remaining 8 problems are also analyzed and presented in Section 4.5.3.

4.5.2 Experimental Results

In this subsection we discuss some of the experimental results.

Test Case 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 given in Figure 4.4.

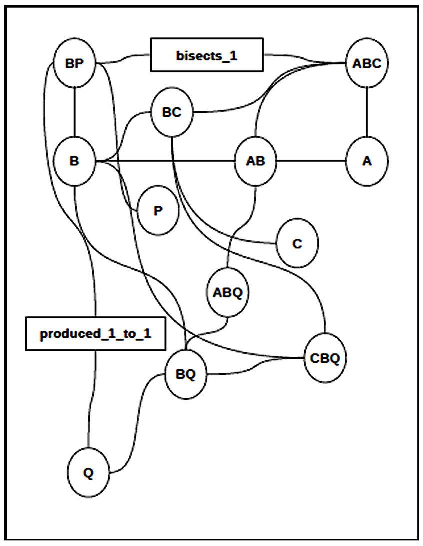


Figure 4.4: Output parse graph for Test Case 1

The *intermediate representation* generated by the *translator* corresponding to Figure 4.4 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)
```

Test Case 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 shown in Figure 4.5.

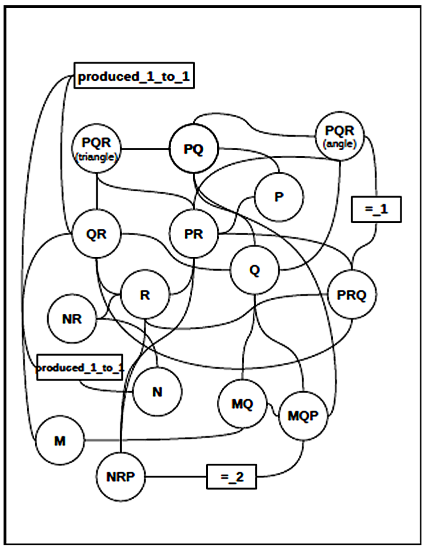


Figure 4.5: Output parse graph for Test Case 2

The *intermediate representation* out of the *parse graph* in Figure 4.5 is:

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

Test Case 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 as shown in Figure 4.6.

The *intermediate representation* for the *parse graph* in Figure 4.6 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
```

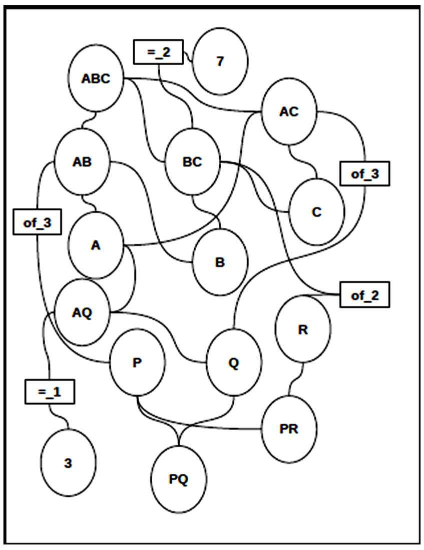


Figure 4.6: Output parse graph for Test Case 3

```
midpoint_3=R
line_4=AQ
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)
```

Test Case 4

```
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 final graph produced is Figure 4.7.

The *intermediate representation* for the *parse graph* in Figure 4.7 is:

```
bisector_1=GH
point_1=G
point_2=H
```

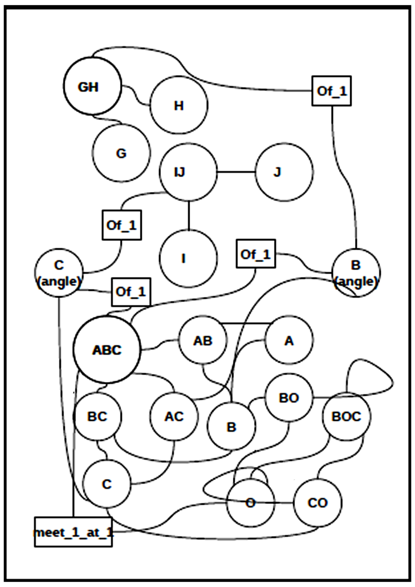


Figure 4.7: Output parse graph for Test Case 4

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

Test Case 5

```
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 for this problem is shown in Figure 4.8.

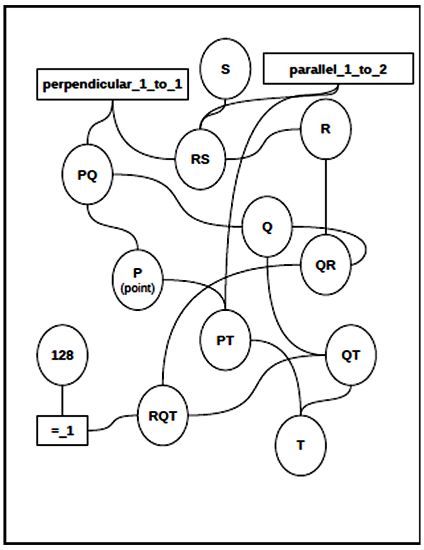


Figure 4.8: Output parse graph for Test Case 5

The *intermediate representation* corressponding to the *parse graph* in Figure 4.8 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)
=_1(angle_1=RQT,const_1=128)
```

Our implementation of the graphs are adjacency matrix. However, we modied the adjacency matrix so that the *connectors* would not have to be replicated for the cases having multiple NNPs connected through a single *connector*. Instead of the cells at the intersection of *connector* and NNP (rows and columns) having a simple 0 or a 1, they contain the index (belonging to the square matrix) of the column of the entity to which the entity in the current column connect through the *connector* in the current row.

4.5.3 Error Analysis

It is noted that out of 40 test problems 32 problems are perfectly parsed and translated into *intermediate representation*. However, for 8 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 *translator* 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:

Test Case 6

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 occurrence of that plural entity.

For example, in the problem statement, `The bisectors of the angles of triangle ABC meets at O` the number of `bisectors` depends on the properties of the entity `triangle`.

The expected output of Step 2 should be Table 4.8.

Table 4.8: Expected entity table for Test Case 6

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 method is Table 4.9.

Table 4.9: Entity table actually generated for Test Case 6

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 in the present form our *parser* cannot resolve this problem.

Test Case 7

When an entity is denoted by more number of letters (of the English alphabet) than such entities usually are denoted by, then our model cannot recognize it.

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.
```

Test Case 8

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 angle ABC = angle BCA. Prove that it is
an equilateral triangle.
```

Test Case 9

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 as Table 4.10, but it actually produced Table 4.11.

```
AB is parallel to DE and angle ABC and angle DEF are on the same
side of AB such that angle ABC = angle DEF. Prove that BC is
parallel to EF.
```

Table 4.10: Expected entity table for Test Case 9

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

Table 4.11: Entity table actually generated for Test Case 9

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

Test Case 10

Our method also produces erroneous results when there are no named entities (NNPs) like the following example.

```
Circles are centered on vertex of a square of side 4 cm such
that they touch each other.  What is the area of these circles?
```

4.6 Summary

In the context of automatic text-to-diagram conversion, translation of a natural language problem statement into a formal description is attempted. An input natural language statement is parsed and then converted into a formal *intermediate representation*. The *parser* is a rule based one. It makes use of a *POS tagger* and the lexical knowledge base *GeometryNet* that contains information 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 beauty of the *parser* that generates the *parse graph* lies in the fact that it intelligently gathers and assimilates information presented in the problem itself to figure out the mathematical relationships among the elements thereby avoiding the rudimentary approach of linguistic pattern matching. The entire approach is tested with 40 high school problems out of which 32 are correctly parsed and errors for the other 8 problems are analyzed in details. The actual utility of this NLP module is realized when the *intermediate representation* is converted into drawing and the entire t2d conversion approach is tested in an end-to-end manner.

The summary or *intermediate representation* produced by this NLP module can also be used as input for the purpose of machine translation of the problem from English to other language or to aid in the solution of a geometry problem. To the best of our knowledge, this is the first attempt of an algorithmic approach of producing formal representation of natural language geometry problems, supported with an evaluation of the approach.

CHAPTER 5

Diagram Drawing and Evaluation

5.1 Introduction

In Chapter 4 it is explained how language processing (NLP) module [8] is effectively employed to analyze the English word statements of an input geometry problem and subsequently generate the language-independent formal representation [8] of the problem in the form of a graph as well as summary. In this process the *GeometryNet* [11] is used in a limited way just to decide and extract the useful information explicitly conveyed or implied in the problem. It is not possible to draw the geometric figure of the input problem simply from this problem summary - the machine must understand the meaning of the different technical terms in the summary first. In this chapter it is shown how the summarized representation in the previous stage is conceptualized with complete geometrical meaning (including all parameters and conditions) of the problem and then valid numerical data is produced which is both necessary and sufficient to draw a diagram. A second *parser* module *(Parser 2)* [7] makes maximum use of the *Net* and produces the output in following five steps.

- **Step 1:** *Parser 2* takes as input the structured intermediate representation of the input word problem outputted by the NLP module and invokes the function for searching the *GeometryNet*.

- **Step 2:** Relevant knowledge (child entities and related expressions) about each term featuring in the NLP output is mined from *GeometryNet*.

- **Step 3:** Dynamic correlation is made between the generic information extracted from *GeometryNet* and corresponding information stored in the connected parse graph outputted by the NLP module. The output is thus an instantiated expanded form of the intermediate representation having all the named atomic entities (co-ordinate variables) and relations (equations).

- **Step 4:** All the variables outputted in Step 3 are assigned suitable values by either referring to default numerical values stored in *GeometryNet* or solving contsraint equations. The output is thus a draw-able representation (of the input problem) listing numerical values of all the parameters (variables) of all the entities to be drawn.

- **Step 5:** A graphic module is invoked that passes the numerical values of the entity parameters as arguments of corresponding graphic functions (line draw or circle draw). In effect, all the entities are drawn step-by-step on display thereby giving shape to the diagram representing the input problem.

The working of the diagram drawing module and the output produced at each step is demonstrated in Section 5.2 using the same parallelogram problem taken as example in Chapter 4.

The drawing module is tested with the NLP output of around 40 school-level geometry problems. The experimental results, i.e. the drawing outputs are evaluated by algorithmically matching with corresponding ground-truth diagrams created using CAD software. Two methods of evaluation are described in Section 5.3 - one based on matching physical attributes while the other one based on matching geometric properties. Though, we do not get 100% correct output (diagram) in all the test cases, for well-defined problems the output is satisfactory. Upon analysis, we found that if the interpretation of natural language (that describes a problem) and subsequently the formal representation of a problem are correct, the drawing output is also correct as the working principle of the diagram drawing module is quite straight forward barring few typical cases.

5.2 Method

The input to this module is the formal representation [8] of the input problem as produced by the NLP module described in Step 6 under Section 4.2 of Chapter 4. For the parallelogram problem taken as example, the output of Step 6 is:

```
parallelogram_1=ABCD
line_1=AB
line_2=BC
line_3=CD
line_4=AD
point_1=A
point_2=B
point_3=C
point_4=D
midpoint_1=X
line_5=AX
point_5=Q
line_6=CQ
point_6=P
parallelogram_2=ABPQ
line_7=BP
line_8=PQ
line_9=AQ
(midpoint_1=X).(line_2=BC)
(line_2=BC).(parallelogram_1=ABCD)
produced_1_meets_1(line_3=CD,line_5=AX)
at_1(line_3=CD,point_5=Q)
extended_1_to_1(line_6=CQ,point_6=P)
=_1(line_3=CD,line_6=CQ)
=_2(line_6=CQ,line_8=PQ)
```

Now corresponding to each different noun, adjective, and verb outputted by the NLP module in the above form, the *GeometryNet* is searched by a second *parser* program to retrieve relevant knowledge or generic data set, i.e. all defining parameters and conditions in terms of variables as shown below.

```
parallelogram_1=ABCD
```

```
n=4
vertex_1=point_1=A(x1,y1)
vertex_2=point_2=B(x2,y2)
vertex_3=point_3=C(x3,y3)
vertex_4=point_4=D(x4,y4)
side_1=line_1=AB(point_1,point_2,m1=(y2-y1)/(x2-x1))
side_2=line_2=BC(point_2,point_3,m2=(y3-y2)/(x3-x2))
side_3=line_3=CD(point_3,point_4,m3=(y4-y3)/(x4-x3))
side_4=line_4=AD(point_1,point_4,m4=(y4-y1)/(x4-x1))
#((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
#(y2-y1)/(x2-x1)=(y4-y3)/(x4-x3)
#(y3-y2)/(x3-x2)=(y1-y4)/(x1-x4)
end
midpoint_1=X=(x5,y5)
end
line_5=AX(point_1,point_5,m5=(y5-y1)/(x5-x1))
end
point_6=Q=(x6,y6)
end
line_6=CQ(point_3,point_6,m6=(y6-y3)/(x6-x3))
end
point_7=P=(x7,y7)
end
parallelogram_2=ABPQ
n=4
vertex_1=point_1=A(x1,y1)
vertex_2=point_2=B(x2,y2)
vertex_3=point_7=P(x7,y7)
vertex_4=point_6=Q(x6,y6)
side_1=line_1=AB(point_1,point_2,m1=(y2-y1)/(x2-x1))
side_2=line_7=BP(point_2,point_7,m7=(y7-y2)/(x7-x2))
side_3=line_8=PQ(point_7,point_6,m8=(y6-y7)/(x6-x7))
side_4=line_9=AQ(point_1,point_6,m9=(y6-y1)/(x6-x1))
#((x2-x1)^2+(y2-y1)^2)^0.5=((x6-x7)^2+(y6-y7)^2)^0.5
#((x7-x2)^2+(y7-y2)^2)^0.5=((x6-x1)^2+(y6-y1)^2)^0.5
#(y2-y1)/(x2-x1)=(y6-y7)/(x6-x7)
#(y7-y2)/(x7-x2)=(y6-y1)/(x6-x1)
end
point_5=X(x5=(x2+x3)/2,y5=(y2+y3)/2)
end
#y6-m5*x6=y1-m5*x1, y6-m3*x6=y4-m3*x4
#(y6-y1)/(x6-x1)=m5,(y6-y3)/(x6-x3)=m3
#m5!m3
```

```
end
#y7-m6*x7=y3-m6*x3
#(y7-y6)/(x7-x6)=m6
end
 #((x4-x3)^2+(y4-y3)^2)^0.5=((x3-x6)^2+(y3-y6)^2)^0.5
end
 #((x3-x6)^2 +(y3-y6)^2)^0.5=((x7-x6)^2+(y7-y6)^2)^0.5
end
```

The mining of the knowledge base is implemented with a recursive function *geoshape()*. It scans the semantic output line-by- line, whenever an entity (e.g. `parallelogram_1 = ABCD`) is found it passes the basic entity (e.g. `parallelogram`) as argument of the function and returns all the entities and expressions listed against that entity in the *Net*. If the returned value is anything other than a *point* or an expression (e.g. *vertex, side, etc.*) those are sequentially sent as arguments of *geoshape()*; the recursive function call is terminated when the function returns point(s) or expression(s). Thus search for `parallelogram` triggers sequential searching of knowledge about *vertex, point, side* and *line (ls)* in the *Net*. Dynamic correlation is made between the generic information (e.g. `point_1=(x1,y1)`) extracted from *Net* for each constituent point corresponding to the NN (e.g. `parallelogram_1`) searched for and the relationship already stored in the *connected graph* for the respective NNP (i.e. `ABCD`). This results in assignments like `point_1=A:(x1,y1), line_1=AB(point_1,point_2, m1=(y2-y1)/(x2-x1)).` the `midpoint` term is first encountered as `midpoint_1=X`, the assignment that is made is `midpoint_1=X=(x5,y5).` Corresponding to the next occurrence of the `midpoint` in the NLP output, the program correlates the term `(midpoint_1=X).(line_2=BC)` with the *GeometryNet* definition of `midpoint` which is as follows.

```
midpoint
point
#x=(x1+x2)/2
#y=(y1+y2)/2
end
```

The resultant information obtained is:

```
point_5=X(x5=(x2+x3)/2,y5=(y2+y3)/2)
```

All the points after being assigned generic values against point names are stored in a 2D array as shown in Table 5.2.

Table 5.1: Array of points

0	A	x1,y1
1	B	x2,y2
2	C	x3,y3
3	D	x4,y4
4	X	x5,y5
5	Q	x6,y6
6	P	x7,y7

Thus the output of this step contains all parameters as variables and constraints as equations pertaining to the geometric elements of the target diagram. The problem is now fully understood. But the variables need to be instantiated with numerical values so as to actually plot the elements on computer screen. This is done in the next step; a more concise and executable representation is formed that would facilitate transformation of geometrical knowledge into diagram for the problem concerned.

While instantiating variables the equations expressing parameter conditions or entity relations are checked or solved to derive the numerical value of the variables. Often defining parameters (like *vertices, end points, center, radius*) of base entities like *line, circle, triangle, rectangle, square, parallelogram,* etc. are neither specifically mentioned in a problem nor are derivable from geometric relations. To resolve such cases, default numerical values are stored separately in the knowledge base against each defining parameter of basic entities (refer Table 3.1 of Chapter 3). In our example problem, variables `(x1,y1),(x2,y2),(x3,y3),(x4,y4)` of the vertices of *parallelogram* `ABCD` are assigned default numerical values `(100,200),(50,100),(250,100),(300,200)` respectively. All other variables (point coordinates) in this problem are derived by solving linear equations involving the variables. For coordinates `(x6,y6)` of `Q` we get following two equations in two unknowns `x6` and `y6` that are solved using the *Gauss-Jordan* method.

```
y6-m5*x6=y1-m5*x1
y6-m3*x6=y4-m3*x4
```

When only one equation `y7-m6*x7=y3-m6*x3` in two unknowns `x7` and `y7` is encountered for coordinates `(x7,y7)` of `P`, a routine searches for another linear equation in the same unknowns (finds that in the expression `(y2-y1)/(x2-x1)=(y6-y7)/(x6-x7)` under `ABPQ` *parallelogram*) and solves them using *Gauss-Jordan* method. However, as point `P(x7,y7)` cannot be evaluated directly in the first pass, the subsequent definition of `line_8` and `parallelogram_2` remains incomplete or only partially evaluated; a flag is set to zero to record each such incomplete evaluation. In the second pass when applying the *Gauss-Jordan* method `P(x7,y7)` is evaluated; the flag is reset to 1 and all subsequent variables are successfully evaluated. This is a common situation encountered while evaluating intermediate representations and thus an algorithm is made to handle it with acceptable accuracy[1]. Finally the evaluated output generated at this stage is:

```
A:(100,200)
B:(50,100)
C:(250,100)
D:(300,200)
AB:(100,200),(50,100)
BC:(50,100),(250,100)
CD:(250,100),(300,200)
DA:(300,200),(100,200)
X:(150,100)
AX:(100,200),(150,100)
Q:(200,0)
CQ:(250,100),(200,0)
P:(150,-100)
BP:(50,100),(150,-100)
PQ:(150,-100),(200,0)
AQ:(100,200),(200,0)
```

Now, a graphic module works on the above output. For the lines `AB, BC, CD, AD, AX, CQ, BP, PQ,` and `AQ`, it invokes the standard line draw function from the graphics library and passes respective set of coordinate values as arguments. Corresponding to the points `A, B, C, D, X, Q, P`, point labels are given at the respective

[1]Inaccuracies result when, in absence of linear equations, quadratic equations are solved for evaluating unknowns.

coordinate locations.

The diagram finally drawn out of sequential execution of all the line drawing instructions is shown in Figure 5.1.

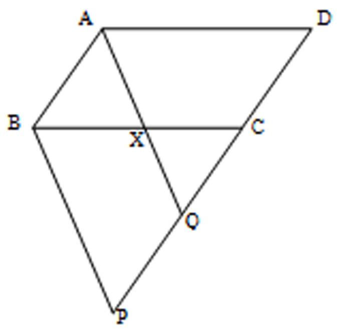


Figure 5.1: The computer generated diagram for the parallelogram problem

5.3 Evaluation

In this section we have described two approaches to evaluate the diagrams produced from textual description of geometry problems. In Method I, the strategy adopted is based on matching of physical attributes of geometric entities following some recent work [48] on evaluating graphics recognition systems. In Method II [10], we have proposed a novel approach that evaluates diagram based on the geometric properties rather than the physical properties and thus overcomes the limitations of Method I.

5.3.1 Method I

Graphic Recognition Systems [48] are designed to convert scanned paper based drawing (raster image) into vector format. The performance evaluator [48, 67, 80] of such systems matches the output vectors to the corresponding vectors in the groundtruth vector file synthesized separately for each drawing. An object-process schematic for such an evaluator [48] is depicted in Figure 5.2. The matching algorithm is based on a set of matching criteria and acceptance threshold for pairs of graphical entities (*line-line, circle-circle, arc-circle,* etc.). Using a linear combination of count of *one-to-one, many-to-one, one-to-many* *matches,* and *mismatches* (missed and false entity) a set of performance evaluation metrics are defined, e.g. *detection rate, recognition accuracy, missed detection rate,* etc. A benchmark data set is used as ground truth for comparison of different vectorization packages.

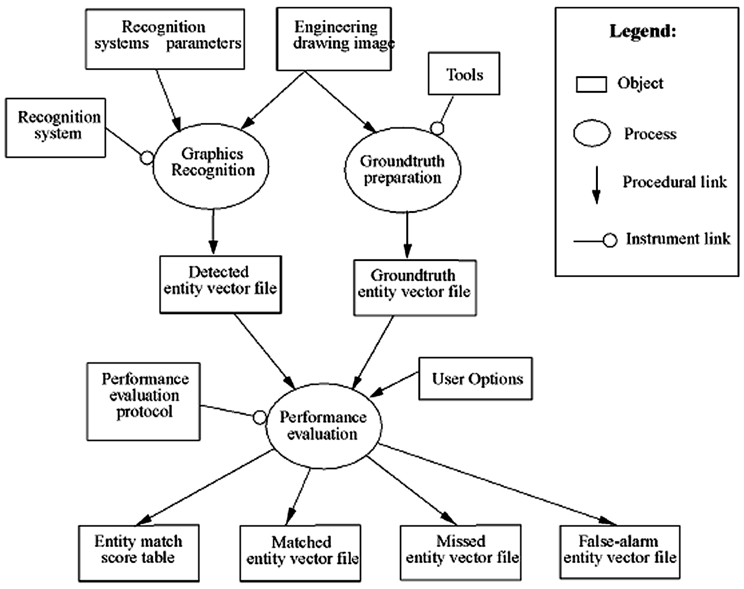

Recognition systems parameters
Engineering drawing image
Tools
Recognition system
Graphics Recognition
Groundtruth preparation
Detected entity vector file
Groundtruth entity vector file
Performance evaluation protocol
User Options
Performance evaluation
Entity match score table
Matched entity vector file
Missed entity vector file
False-alarm entity vector file
Legend:
Object
Process
Procedural link
Instrument link

Figure 5.2: The object process diagram of the performance evaluator [48]

Method I proposed by us, takes the above idea but requires reference diagrams to be separately drawn using any CAD software. This is required for physical matching of a pair of entities - one which is drawn automatically by the machine and another drawn using the CAD software, the input (problem description) being same for both.

Evaluation Metrices

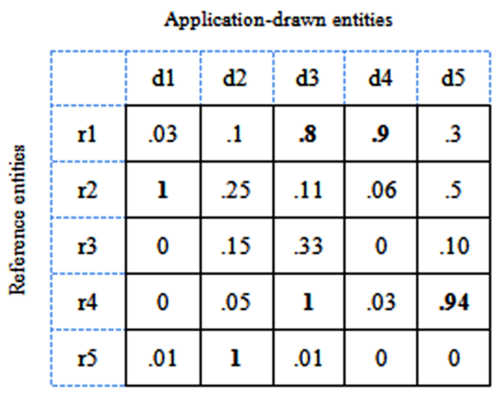
Method I can be divided into following four steps.

Step 1: First we draw the reference diagram of a problem using line or circle draw commands of a standard CAD tool (like AutoCAD) following the problem description line by line. In order to minimize coordinate calculation and correctly represent relative positions and relations, we use the snap functions of the CAD tool like *end of, mid of, center of, perpendicular to, intersection of, extend to, tangent to,* etc. Conceptually there may be many different correct diagrammatic representation of a given problem provided we allow the drawing at any position, any orientation and any dimension keeping the basic relative positions or relations intact. But to narrow down our problem and make comparison easier, we put the constraint of using the same physical attributes of basic drawing entities as that used by our t2d conversion system. Thus we use the base value of the definition points (e.g. *vertices*) or other parameters (e.g. *radius*) of the starting entities for a problem. The base values are the default values of parameters stored against each entity type in *GeometryNet*. The starting entities are those that need to be drawn first and which is not bound by any relation or parametric constraint as far as the problem description is concerned. For example, *parallelogram* `ABCD` is the starting entity for the parallelogram problem dealt in Section 5.2 and the base values of a *parallelogram* are the default coordinates of the four vertices stored in *GeometryNet* against *parallelogram* entity - these are A(100,200), B(50,100), C(250,100), D(300,200). The CAD-drawn or reference entities and the system-drawn entities are listed (sorted type-wise) in two separate text files along-with their arguments (e.g. *end point* coordinates for a *line*, *center point* coordinates, and *radius* for a *circle*, etc.).

Step 2: The performance of our automatic diagram drawing system is measured by comparing (from two text files) the entities automatically drawn by it with the reference (groundtruth) entities and counting -

- the number of matches, i.e. correctly drawn entities = a
- the number of misses, i.e. described in the problem but not drawn entities = b
- the number of false entities, i.e. not described in the problem yet drawn entities = c

An overall performance factor is formulated involving the count of a, b and c. Again, taking into account the possibility that an entity may not be drawn perfectly due to incorrectly derived or collected parameter values, the quality of drawing or degree of correctness is measured for each entity drawn by comparing it with the reference entities. When a pair of reference entity and system-drawn entity has *match quality* higher than a prefixed *acceptance threshold*, the match is accepted. All perfect matches are given full credit. All other accepted matches are considered for partial credit. A sample *match quality matrix* is shown in Figure 5.3.



Application-drawn entities

Reference entities

	d1	d2	d3	d4	d5
r1	.03	.1	.8	.9	.3
r2	1	.25	.11	.06	.5
r3	0	.15	.33	0	.10
r4	0	.05	1	.03	.94
r5	.01	1	.01	0	0

Figure 5.3: A sample *match quality matrix*

First, each entity corresponding to the system-drawn diagram are matched or compared with all reference entities same type, e.g. a *line* with *lines*, a *circle* with *circles*. This implies if there are m system-drawn entities of a given type and n reference entities of the same type then there will be total mn matches for that type. Pair-wise *match quality* is computed having values between 0 and 1; 1 is considered a perfect match. The algorithm for comparing *line* and *circle* entities belonging to two different sets are illustrated in Section 5.3.1

.

Step 3: To find the count of *a, b* and *c*, the *match quality matrix* is converted into a one-zero integer matrix that we term as *match count matrix*. Any *match quality* value greater than *acceptance threshold* (let us say, 0.80) is considered as a good match and set to 1 in the *match count matrix*. All the other matches are rejected and set to 0. The *match count matrix* corresponding to the *match quality matrix* (Figure 5.3) is shown in Figure 5.4.

Two arrays R(i) and D(j) are generated from the *match count matrix*. The values of R(i) array are computed as the sum of the matches in the i^{th} row of the *match count matrix*. Similarly, D(j) array is computed as the sum of matches in the j^{th} column. These two arrays are interpreted as per following logic to find the counts of *a, b* and *c*.

$$\begin{array}{c|ccccc|c}
 & d1 & d2 & d3 & d4 & d5 & \\
\hline
r1 & 0 & 0 & 1 & 1 & 0 & 2 \\
r2 & 1 & 0 & 0 & 0 & 0 & 1 \\
r3 & 0 & 0 & 0 & 0 & 0 & 0 \\
r4 & 0 & 0 & 1 & 0 & 1 & 2 \\
r5 & 0 & 1 & 0 & 0 & 0 & 1 \\
\hline
 & 1 & 1 & 2 & 1 & 1 &
\end{array}$$

match_with_reference D(j) match_with_drawing R(i)

Figure 5.4: The *match count matrix* corresponding to the matrix in Figure 5.3

```
If
R(i)=1
match_count(i,j)=1
D(j)=1
Then
  dj matches perfectly with ri
  count of a is increased by 1
Else If
R(i) = 0
Then
   ri matches no system-drawn entity
   count of b is increased by 1
Else If
D(j) = 0
Then
   di matches no reference entity
   count of c is increased by 1
Else match is partial
   If R(i) >1
    count of one_r_to_many_d match is increased by 1
   If D(j) >1
    count of many_r_to_one_d match is increased by 1
```

For the *match count matrix* shown in Figure 5.3, r2-d1 is a perfect match because R(2) = 1, match_count (2,1) = 1 and D(1) = 1. Similarly, r5-d2 is another perfect match. That implies entities d1, d2 are perfectly drawn.

Any R(i) = 0 implies i^{th} r-entity does not match any system-drawn entity implying a *miss* in the drawing. In our example r3 is missed, i.e. not drawn.

Any D(j) = 0 implies j^{th} d-entity does not match with any reference entity. So that is a *false entity*. The example does not show any *false entity*.

If the value of a R(i) is more than 1 it implies i^{th} reference entity matches with more than one drawn entity. In our example R(1) = 2 as r1 partially matches with each of d3 and d4; also r4 partially matches with each of d3 and d5. Similarly, we can find the d-entities that matches more than one reference entities by checking the D(j) entries which have values greater than 1. In our example, D(3) = 2 as d3 matches partially with each of r1 and r4.

According to the above algorithm the evaluation of the *match count matrix* and D(i), R(j) arrays yield the following values.
perfect match (a) = 2
miss (b) = 1
false (c) = 0
one_r_to_many_d (x) = 2
many_r_to_one_d (y) = 1

Step 4: To get a true reflection of performance of our system, the following empirical formulae or performance metrics involving the variables evaluated earlier are proposed.

1. *Credit Index (CI)* = (a + ($\sum$ maximum of each of x) + ($\sum$ maximum of each of y))$/n$

   Here n is the no. of reference entity and for a zero error drawing, the drawing credit will be 1.

2. Missed Entity Rate = b/n

3. False Drawing Rate = c/m

4. Performance factor (PF) = $(n\ CI + w_1 b + w_2 c)/n$,

   where w_1 and w_2 are negative weights to be decided.

For a zero error drawing, CI = PF = 1 or 100%.

For the example case cited before,

1. CI = (2.0+(0.9+1.0)+(1.0))/5 = 4.9/5 = 0.98, i.e. 98%
2. Miss rate = 2/5 = 0.4
3. False rate = 0/5 = 0
4. PF = (5(0.98)-1(1)-0.5(0))/5 = 3.9/5 = 0.78 = i.e. 78%, where w_1=-1, w_2=-0.5

Comparison of Drawing Elements

The algorithms for comparing system-drawn and CAD-drawn entities are illustrated below.

Line-Line Matching

Let, i^{th} r, i.e. ri and j^{th} d, i.e. dj be two straight lines belonging to reference entity set and system-drawn entity set respectively.

```
If
ri and dj have same end points,
Then
set match_quality(i,j)=1
Exit
Else If
angle(ri,dj)>$T_a$
Then
set match_quality(i,j)=0
Exit
Else If
distance(ri,dj)>$T_d$
Then
set match_quality(i,j)=0
Exit
Else If
proj(dj,ri)<$T_l$*dj And
proj(dj,ri)<$T_l$*ri
Then
set match_quality(i,j)=0
Exit
Else
set match_quality(i,j) =
proj(dj,ri) / max(length(dj),length(ri))
```

In the above algorithm, T_a and T_d stands for *acceptance threshold* for angle and distance respectively. Here angle( ) implies smaller angle between ri and dj and distance( ) implies average of the orthogonal distances from midpoint of dj to ri and vice versa. T_l( ) implies the threshold percentage of length of a line while proj( ) means length of projection of dj on ri.

It is evident from the above algorithm that if end point coordinates for the compared entities are same, then it is a perfect match (value = 1) and we need not check any other aspects. If the coordinates do not match then it is either a poor match (0 value) or partial match (measured by the ratio of projected length and actual length) depending on whether the internal angle, orthogonal distance and projected length of the lines are within acceptable limits or not.

Circle-Circle Matching

Let, ri and dj be the reference circle and the system-drawn circle respectively to be compared. Also let, the center and radius of the two circles are (C_{ri}, R_{ri}) and (C_{dj}, R_{dj}) respectively.

```
If
$C_{ri} = C_{dj}$ And $R_{ri} = R_{dj}$
Then
set match_quality(i,j)=1
Exit
Else If
Distance($C_{ri}$,$C_{dj}$)>$T_c$
Then
set match_quality(i,j)=0
Exit
```

```
Else If
$|R_{ri} - R_{dj}| > T_r$
Then
set match_quality(i,j)=0
Exit
Else If
min($R_{ri}, R_{dj}$)/max($R_{ri}, R_{dj}$) $< T_r$
Then
set match_quality(i,j)=0
Exit
Else
set match_quality(i,j) =
min($R_{ri}, R_{dj}$)/max($R_{ri}, R_{dj}$)
```

The above algorithm awards partial credit to a system-drawn circle if its center and radius are not identical with that of the reference circle but the centers and radii differ within acceptable limits T_c and T_r respectively. In that case *match_quality* value is computed by finding the ratio of the minimum and maximum of the two radii.

Experimental Results

Here we present five geometry problems (out of the test set of 40 selected exercise problems) and corresponding diagrams (Figure 5.5 to Figure 5.9) that are automatically drawn by our system from the problem descriptions. The performance metrices are also evaluated by comparing the system-drawn diagrams with corresponding reference diagrams drawn in AutoCAD.

Problem 1:

BP bisects angle ABC and PB is produced to Q. Prove angle ABQ=angle CBQ. (Refer Figure 5.5)

Evaluation result:

Credit Index: 1, Missed Rate: 0, False Rate: 0, Performance Factor: 1

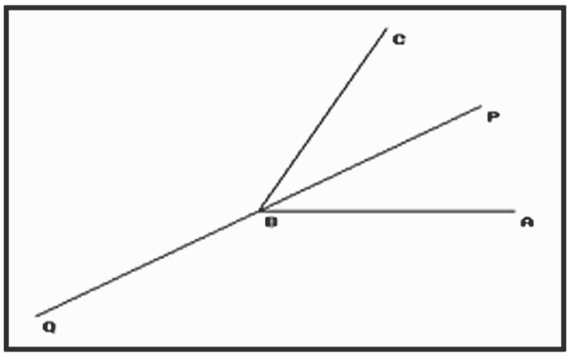


Figure 5.5: t2d system-drawn diagram for Problem 1

Problem 2:

CO stands on AB at O. OD and OE bisect angle AOC and angle BOC. Prove that angle DOE=90 degree. (Refer Figure 5.6)

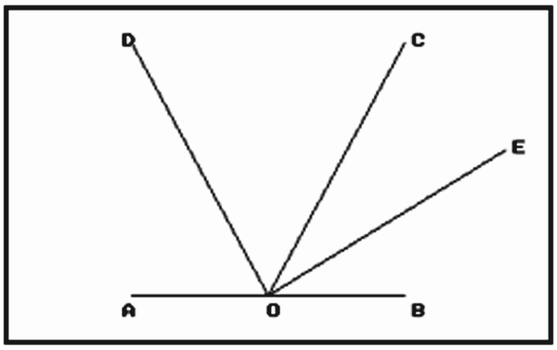


Figure 5.6: t2d system-drawn diagram for Problem 2

Evaluation result:

Credit Index: 1, Missed Rate: 0, False Rate: 0, Performance Factor: 1

Problem 3:

PQ is perpendicular to RS. PT is parallel to RS. Prove that angle RQT=128 degree. (Refer Figure 5.7)

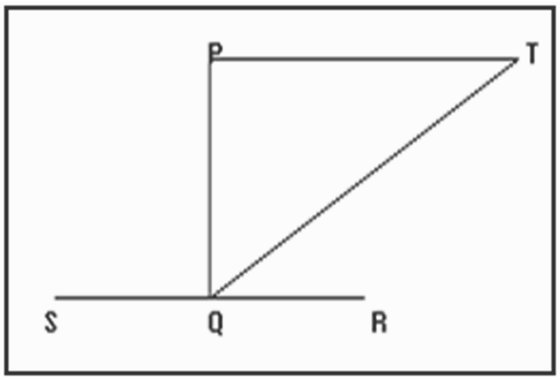


Figure 5.7: t2d system-drawn diagram for Problem 3

Evaluation result:

Credit Index: 1, Missed Rate: 0, False Rate: 0, Performance Factor: 1

Problem 4:

In triangle ABC, angle A=60 degree, angle ABC = 90 degree, AB is produced to E and AC is produced to F. The bisectors of the exterior angles CBE and BCF meet at H. Calculate angle BHC. (Refer Figure 5.8)

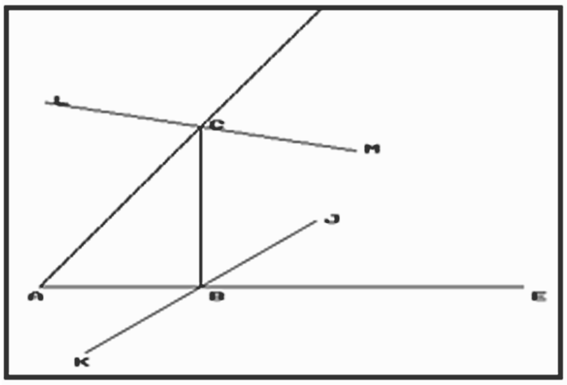


Figure 5.8: t2d system-drawn diagram for Problem 4

Evaluation result:

Credit Index: 0.8, Missed Rate: 0.2, False Rate: 0, Performance Factor: 0.6

Problem 5:

The bisectors of angle B and angle C of an equilateral triangle ABC meet at O. Find angle BOC. (Refer Figure 5.9)

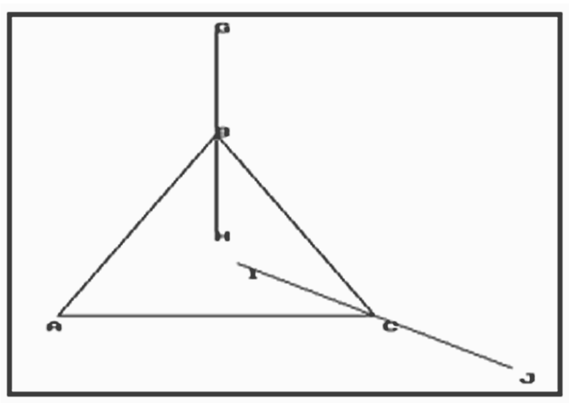


Figure 5.9: t2d system-drawn diagram for Problem 5

Evaluation result:

Credit Index: 0.75, Missed Rate: 0.25, False Rate: 0, Performance Factor: 0.5

Thus correctness of all the diagrams automatically drawn from a test set of 40 school-level geometry problems are evaluated algorithmically. Result of evaluation of 40 problems show a Credit Index of 1.0 (i.e. 100% correctness) for 32 problems, varies between 0.75 (75%) to 0.94 (94%) for 7 problems and 60% for remaining 1 problem. It is observed that the credit is less than 1 for 20% diagrams mostly due to missed entities - false entities are hardly drawn.

5.3.2 Method II

Employing Method I, system-drawn diagram can be successfully evaluated with respect to the groundtruth diagram (drawn in CAD) based on physical attributes of diagram entities, provided we use the same set of values of the definition points (e.g. vertices) and other parameters (e.g. *radius*) of the base/starting entity (as used by the t2d system) while drawing the groundtruth diagram in CAD. But this approach has a serious limitation for evaluating output graphics that can be drawn practically at any position and in any size or orientation just as a student does on a piece of paper. For example, if a system-drawn parallelogram and a groundtruth parallelogram are positioned near two different corners of the display screen in different size and orientation, then they will no way match with each other based on physical attributes and the system will reject it as a mismatch. But the system-drawn parallelogram might actually be correct geometrically and would therefore deserve full credit. Only if the base figures (*parallelogram* in this case) are identically positioned (i.e. vertex coordinates are same), the deviations of the internal elements (e.g. *diagonals*) can be physically compared following the matching criterions of Method I. That is why the Method I is not successful where physical attributes may vary significantly compared to graphic recognition problems.

Considering any geometry diagram as a labeled graph, it naturally comes to mind whether graph matching algorithms can be employed to compare the system-drawn diagram with the groundtruth diagram. There are many graph matching algorithms [47] but one inherent limitation of these algorithms is that they only check the pattern of connectivity of the edges without considering the geometrical relationship of the edges and nodes. Geometrical relations may not be so significant for comparing chemical structure [58] or floor plan [13] as it is for comparing geometry diagrams. For example, consider the two geometry structures shown in Figure 5.10. From the perspective of graph (edge) connectivity, these two are identical (isomorphic). But quite clearly they are very different from geometric perspective - one is a parallelogram while the other is a triangle. The existing graph matching algorithms fail to differentiate these two structures.

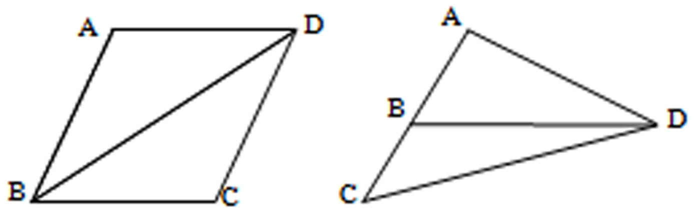


Figure 5.10: Different geometry diagrams seen as similar graphs

The above limitation motivates our work in finding an alternative method [10] that gives an automatic (and reasonable) quantitative evaluation of geometry diagrams drawn by text-to-graphics conversion systems. Just as geometry figures drawn on paper by a student are checked by a teacher for conceptual correctness, the evaluation system should be able to check the geometric properties of the figures rather than relying only on the physical attributes or connectivity of the individual entities.

Input

Given a geometric description, the t2d system automatically produces a diagram after generating a set of coordinate pairs and coordinate-radius duo respectively for constituent lines and circles of the diagram. Whereas, for the same description, another set of coordinates and their connections will be generated for the groundtruth diagram drawn in CAD. Method II [10] will take as input the connectivity of the points in the form of adjacency matrix and also the points coordinate values given in the same order for both the system-drawn and groundtruth diagram. For example, consider the geometric diagram in Figure 5.11 with vertices A(30,30), B(10,60), C(60,60) and D(80,30). Input for this diagram will be: 30,30 / 10,60 / 60,60 / 80,30 and 0 1 0 1 / 1 0 1 1 / 0 1 0 1 / 1 1 1 0. As the coordinate values are entered in the order of A-B-C-D, the adjacency matrix is also entered in the order of A-B-C-D.

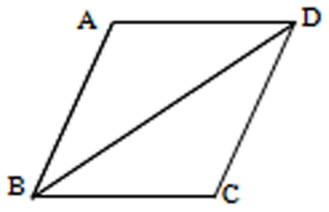


Figure 5.11: Edge connectivity and vertices of a diagram

Processing

From the input coordinates and adjacency matrices, the evluation module calculates the length matrix and slope matrix for the edges of a diagram. From the slope matrix it computes the smaller (acute) angle between every pair of edges connected at any point. A program then searches for the angles of same values and forms different sets of same angles (if more than one). For example, if there are four 90° angles, three 30° angles and one 180° angle between different pairs of edges in a diagram then same-angle sets will be 4 and 3. Similarly, sets are found for same-length edges and these sets are computed for both the diagrams being compared. Next, the system evaluates the geometrical similarity of the two diagrams by comparing the following parameters of the diagrams.

1. Adjacency matrix
2. Total number of points
3. Total number of edges
4. Total number of angles between edges
5. Sets of same-length edges
6. Sets of same angle between edges

If at first comparison the adjacency matrices (of same order) of two diagrams do not match element by element then it is checked whether they match if one of the matrices is read in reverse row and/or reverse column direction. For example, consider the two parallelograms with one diagonal in Figure 5.12. The respective adjacency matrices in the order of A-B-C-D are shown beside the diagrams:

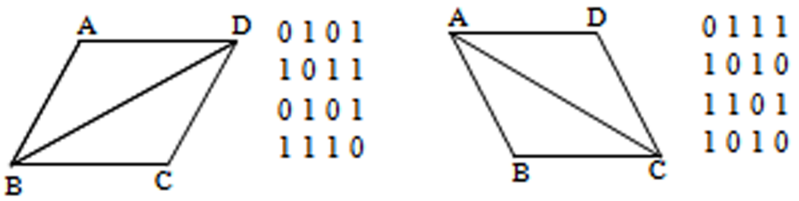


Figure 5.12: Geometrically similar diagrams

The matrices are of same order but apparently they do not match. If we read the second matrix in bottom to top and right to left direction and compare with the first matrix read normally in top to bottom and left to right direction, then the matrices match. This implies the diagrams are geometrically equivalent. Even after this effort if the adjacency matrices (of same order) do not match, then a match score is awarded based on the number of rows completely matched. If only 1 out of 4 rows match, the score is 1/4 out of a maximum possible score of 4/4 or 1. If the orders of adjacency matrices are not same implying unequal number of points, then the score is 0 as no row will match.

With regard to each of parameters (2) to (6), the score is simply calculated as the ratio of the parameter value for the system-drawn diagram to that of the groundtruth diagram.

If x be a parameter value for a system-drawn diagram and y be the corresponding value for the groundtruth diagram then:
score = x/y, for $x \leq y$
= 1-$(x\text{-}y)/y$, for $x > y$
For $x > y$, penalty is assigned for x-y additional elements generated wrongly against expected y elements. It is obvious from the equations above that if a system-drawn diagram is 100% correct implying $x = y$, then score = 1.

Frequently, there will be multiple values of x and y against parameter (5) and (6). Now for each element of set y, the system will try to find a match in set x. For every y:x perfect match, the score is set to 1. Then keeping aside the perfectly matched element(s) of y and x, the nearest match of each of remaining elements of y is searched in the subset of x. The score for each pair of y value and nearest matching x value is calculated following above equations. For cases where there is no y value left in set y (implying y = 0) to match an unused value in set x, the score cannot be evaluated as per the second equation. In such case the score is set to -0.5 to impose a reasonable penalty for undesired value of x that occurs due to incorrect machine drawing. However, if such situation arises for parameter (5) then penalty is not imposed and score is set to 0 because a system-drawn diagram may be correct geometrically with some edges becoming equal which may not be the case in ground-truth diagram. The maximum possible score for each y:x pair of parameters (2) to (6) is 1 except 0 in case of $< 0, x >$ match for parameter (5).

For example let the x values against (6) are $< 3, 2, 1 >$ and the corresponding y values are $< 4, 3 >$. A match is obtained in x for y = 3, but y = 4 has no match. Now a match for y = 4 is searched in x and x = 2 is found closest to 4 among rest of the x elements (2 and

1). The unused element $x = 1$ has got no element left in y to match with. The match score calculated for the $< y, x >$ pairs $< 3, 3 >, < 4, 2 > and < 0, 1 >$ are 3/3 = 1 and 2/4 = 0.5 and -0.5 respectively.

Output

The cumulative score of the system-drawn diagram considering all the parameters and their multiple values is calculated. Also the sum of the maximum possible scores against each of these parameter values is calculated. The maximum possible score corresponds to a 100% correctly drawn diagram. The final output of evaluation is the percentage of cumulative score of the system-drawn diagram and this is calculated as:
[($\sum$ scores of parameters (1) to (6))/($\sum$ maximum scores of parameters (1) to (6))]$\times$100

Experimental Result

The proposed evaluation method has been tested with 40 geometry diagrams - all of which has been produced by the t2d system. We present here five representative cases for illustration of the method. Under each Case, a geometric word problem is cited corresponding to which screenshots of the CAD-drawn groundtruth diagram on top or left of 2td system-drawn diagram are shown. Subsequently, the calculation of scores against individual parameter values and overall score or accuracy percentage is presented for each Case.

Case 1:
Description: *PQ is perpendicular to RS such that Q is the midpoint of RS. PT is parallel and equal to RS. Find angle RQT.* (Refer Figure 5.13 and Table 5.2)

$\sum$ score = 7, $\sum$ max score = 7, overall score = 100%

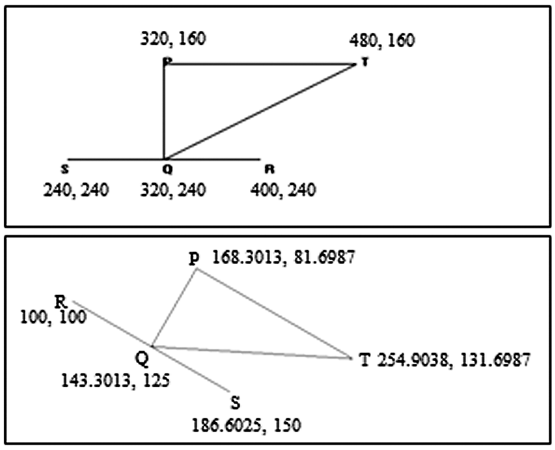


Figure 5.13: Case 1 - Correct system-drawn diagram

Table 5.2: Scoresheet for Case 1

-	**Adjacency matrix**	**Points**	**Edges**	**Angle**	**Same length edge**	**Same angle**
y-values	5x5	5	5	8	3	3,3
x-values	5x5	5	5	8	3	3,3
Score	1	1	1	1	1	1,1
Max score	1	1	1	1	1	1,1

Case 2:

Description: *In triangle ABC, angle A=80 degree, angle CBA = 55 degree, AB and AC are produced to H and K. BE bisects angle CBH, CE bisects angle BCK. Calculate angle BEC.* (Refer Figure 5.14 and Table 5.3)

Table 5.3: Scoresheet for Case 2

-	**Adjacency matrix**	**Points**	**Edges**	**Angle**	**Same length edge**	**Same angle**
y-values	6x6	6	7	14	2	3,3,2,2,2
x-values	6x6	6	7	14	0	3,3,2,2,2
Score	1	1	1	1	0	1,1,1,1,1
Max score	1	1	1	1	1	1,1,1,1,1

$\sum$ score = 9, $\sum$ max score = 10, overall score = 90%

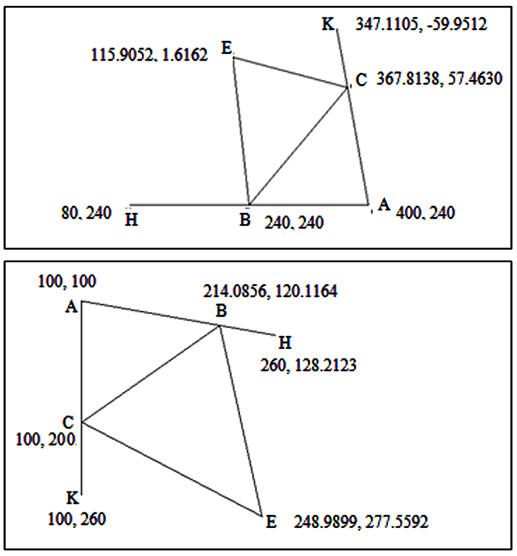


Figure 5.14: Case 2 - Almost correct diagram; only instead of $\angle CBA$, $\angle ACB$ is 55°

Case 3:

Description: *BP bisects angle ABC and PB is produced to Q. Prove angle ABQ = angle CBQ.* (Refer Figure 5.15 and Table 5.4)

Table 5.4: Scoresheet for Case 3

-	**Adjacency matrix**	**Points**	**Edges**	**Angle**	**Same length edge**	**Same angle**
y-values	5x5	5	4	6	0	4
x-values	5x5	5	4	4	2	2
Score	2/5=0.4	1	1	0.67	-	0.5
Max score	1	1	1	1	-	1

$\sum$ score = 3.57, $\sum$ max score = 5, overall score = 71.4%

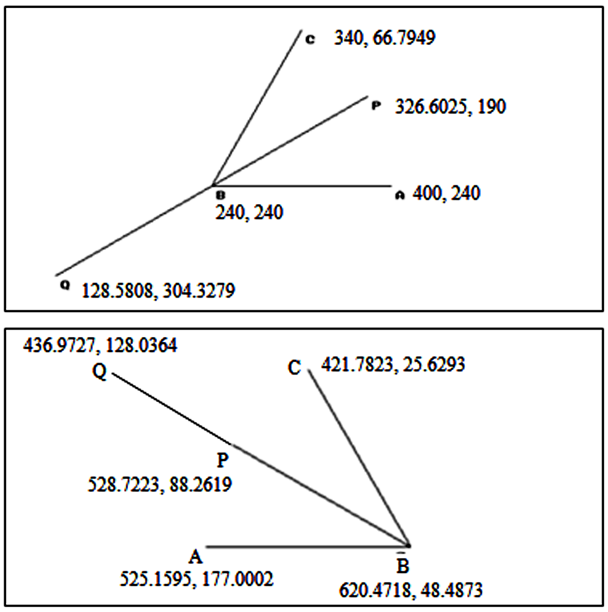


Figure 5.15: Case 3 - Incorrect diagram; PB produced in wrong direction

Case 4:

Description: *The bisectors of angle B and angle C of an equilateral triangle ABC meet at O. Find angle BOC.* (Refer Figure 5.16 and Table 5.5)

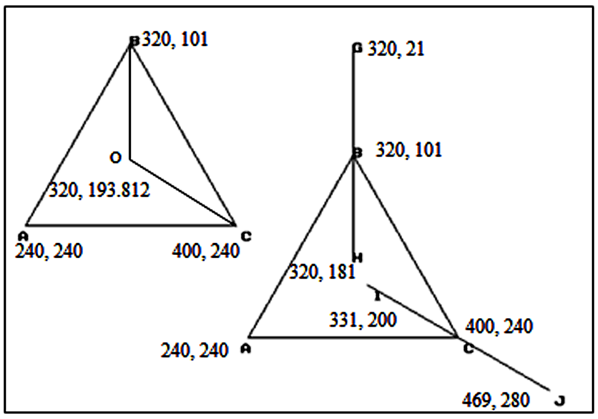


Figure 5.16: Case 4 - Incorrect diagram; angle bisectors produced in wrong direction

Table 5.5: Scoresheet for Case 4

-	Adjacency matrix	Points	Edges	Angle	Same length edge	Same angle
y-values	4x4	4	5	8	3,2	4,4,0,0,0
x-values	7x7	7	7	13	3,2	4,3,2,2,2
Score	0	1-3/4 =0.25	1-2/5 =0.6	1-5/8 =.38	1,1	1,.75,-0.5,-0.5,-0.5
Max score	1	1	1	1	1,1	1,1,0,0,0

$\sum$ score = 3.48, $\sum$ max score = 8, overall score = 43%

Case 5:

Description: *PQ is a diameter of a circle centered at O. The tangents drawn at P and Q of the circle are APB and CQD. Prove that AB*$||$*CD.* (Refer Figure 5.17 and Table 5.6)

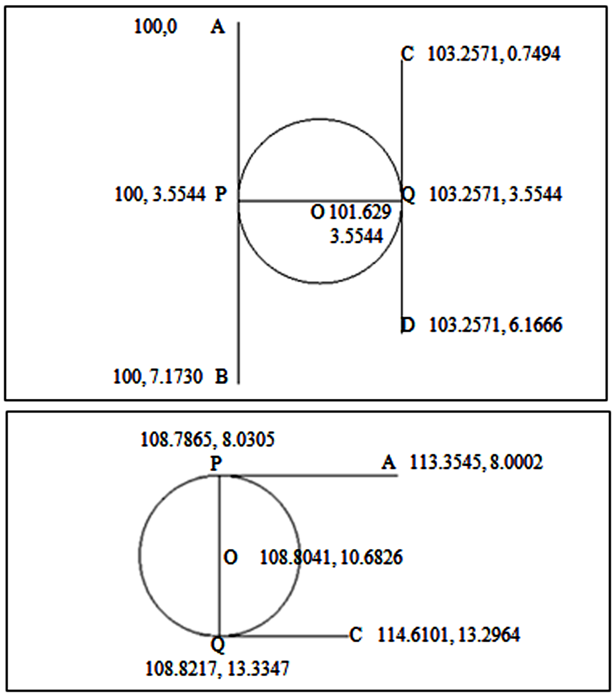


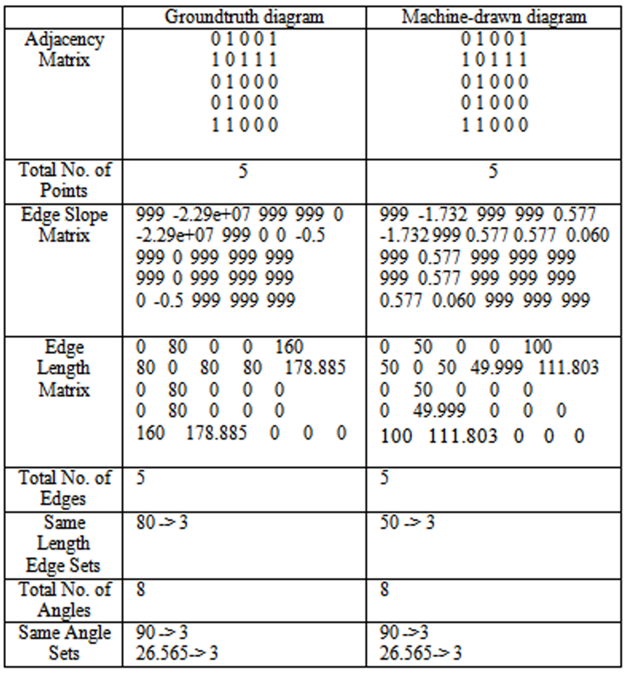
Figure 5.17: Case 5 - Incorrect diagram; tangents not produced in both directions

$\sum$ score = 5.06, $\sum$ max score = 8, overall score = 63.3%

Table 5.6: Scoresheet for Case 5

-	Adjacency matrix	Points	Edges	Angle	Same length edge	Same angle	Circle flag
y-values	7x7	7	7	13	2	8,5	1
x-values	5x5	5	5	7	2	4,3	1
Score	0	0.71	0.71	0.54	1	0.5,0.6	1
Max score	1	1	1	1	1	1,1	1

The intermediate output given by the evaluation system, i.e. the slope matrix, edge length matrix, same length edge sets and same angle sets are depicted only for Case 1 in Figure 5.18. The input adjacency matrix is given in the order of P-Q-R-S-T for both the diagrams in Figure 5.13.



	Groundtruth diagram	Machine-drawn diagram

Adjacency Matrix	0 1 0 0 1
	1 0 1 1 1
	0 1 0 0 0
	0 1 0 0 0
	1 1 0 0 0
		0 1 0 0 1
		1 0 1 1 1
		0 1 0 0 0
		0 1 0 0 0
		1 1 0 0 0

Total No. of Points	5	5

Edge Slope Matrix	999 -2.29e+07 999 999 0
	-2.29e+07 999 0 0 -0.5
	999 0 999 999 999
	999 0 999 999 999
	0 -0.5 999 999 999
		999 -1.732 999 999 0.577
		-1.732 999 0.577 0.577 0.060
		999 0.577 999 999 999
		999 0.577 999 999 999
		0.577 0.060 999 999 999

Edge Length Matrix	0 80 0 0 160
	80 0 80 80 178.885
	0 80 0 0 0
	0 80 0 0 0
	160 178.885 0 0 0
		0 50 0 0 100
		50 0 50 49.999 111.803
		0 50 0 0 0
		0 49.999 0 0 0
		100 111.803 0 0 0

Total No. of Edges	5	5
Same Length Edge Sets	80 -> 3	50 -> 3
Total No. of Angles	8	8
Same Angle Sets	90 -> 3 26.565-> 3	90 ->3 26.565-> 3

Figure 5.18: Different properties for Case 1

Discussion

Quantitative evaluation of five system-drawn diagrams show reasonably good results from geometric perspective. The intermediate system output is shown for only Case 1 to illustrate the computation of sets of same length edge and same angle. The edge matrix and slope matrix corresponds to the adjacency matrix. A high positive value (999) is assumed in the slope matrix where no edge connection is implied and this value will not be considered as a valid slope while calculating the angle between two edges. It is evident from Case 1 and Case 2 that the system is able to evaluate correctness of diagrams in-spite of being drawn with different coordinate sets, different order of labeling and at different orientation compared to the groundtruth diagram. Referring to Figure 5.10, if we consider the diagram with triangle as the system-drawn diagram and the one containing parallelogram as the groundtruth diagram then our system gives an evaluation score of 55.6%. This clearly shows that the diagrams are very different though the number of points, edges, angles and adjacency are same.

Edges or angles that ought to be equal in a correct diagram are actually equal in a system-drawn diagram may not be always represented by the fact that the scoring against parameter (5) and (6) is maximum. Different sets of angles or edges may incidentally become same in an incorrect diagram and still the score may be maximum, if the sets match. In an experimental set of 40 diagrams drawn by the machine from textual description, one such case is found where the evaluation score is 100% but there is mismatch of same angle sets. Overall, we find that out of 40 diagrams, 31 correctly drawn diagrams (as manually observed) are evaluated correctly yielding a score of 98%-100%, 1 incorrect diagram is given 100% score, and remaining 8 incorrect diagrams score between 43% and 90%. 3 incorrect diagrams having circle seems to deserve better score. Though there may be question regarding appropriateness of the scores awarded against incorrect drawings, all correctly drawn diagrams are given high score by the system. Thus we can say that the system near perfectly (if not perfectly) identifies diagrams that are correctly drawn by a t2d conversion system.

Our system has no provision of counting curved edges, particularly circle. Hence for diagrams with circle, the entire circle or the semicircles on both side of a diameter are not counted as edges. In both the diagrams in Figure 5.17, PQ connection is considered only once (as a linear edge) in addition to edges PO and OQ. To compensate this approximation, we introduce a new parameter circle flag. With this flag the total number of parameters against which scores are to be evaluated by the system becomes seven. Whenever the system encounters a coordinate-length pair (center point-radius) as input for a given diagram, the value of the flag will be incremented by 1. If, say, there are 3 circles in the system-drawn diagram then value of x against circle flag will be 3. For simply point coordinates as input, implying only linear edges, flag value will be 0. The score against this parameter will be calculated as usual with the following exceptions - if flag values x and y are both 0 then the score will be 0; if y is 0, but x has a positive value, then score (penalty) will be -0.5. The score of the system-drawn diagram in Case 5 is 58% without this flag vis--vis 63.3% with the flag. However, the inability to detect connectivity of circular edges with other edges of the diagram cannot be fully compensated by this adhoc measure.

5.4 Summary

This chapter discusses the final part of our text-to-graphic conversion system and shows the actual output (geometry diagrams) as produced from geometry word problems. Machine understanding of structured geometric information extracted from a natural language problem and its corresponding graphical representation is described through an example. The core processing module is the *Parser 2* that processes generic information mined from the *Net* to represent it in a numerical form which is compatible with the standard graphics functions. The rest of the drawing process is relatively straight forward. This drawing module is tested with same 40 geometry word problems that were used to test the NLP module. Evaluation of these 40 drawing output gives a measure of accuracy of the proposed t2d conversion system.

It is experienced that the existing evaluation techniques are not sufficient to formulate a suitable evaluation method for the present problem. We have designed two evaluation methods for judging the performance of a t2d conversion system for geometry problems. Evaluation results show that the proposed methods are moderately successful in measuring the accuracy of the diagrams. In depth analysis of our Method II [10] shows that some issues still remain unaddressed. For example, handling of circle requires further effort as it is represented in the adjacency matrix. Moreover, computation of the sixth parameter considers only the number of same angles between edges. This requires modification as it may happen that in one diagram there are two 30° angles which are not present in the second diagram where there are two 60° angles but in different context. In this case, the evaluation method fails to capture the difference and gives full credit instead of inducing penalty.

In the future extension of this study, we are planning to address this problem from two different viewpoints: (a) involvement of *relational graphs* [74] where we would try to represent two diagrams as *relational graphs* and then formulate the evaluation task as matching of two relational graphs, (b) in the second attempt, we want to follow a reverse engineering where we want to convert a graphical figure into its corresponding textual description and then evaluation would be done by comparing the two textual descriptions (converted vs. original). In this process, we do not require to use any groundtruth graphical diagram for evaluation.

CHAPTER 6 Automatic Diagram Drawing using Braille Text

6.1 Introduction

We have seen that in the course of explaining or comprehending or solving a scientific description or problem, usually, a statement (text) is first translated into a sketch (diagram) which visually articulates the essential problem parts; mechanical models, free-body diagrams, electrical/electronic circuits, geometry diagrams are instances of such transformation. From the point of view of understanding a problem, the representative diagrams are not a mere convenience but also an inherent component in a person's cognitive representation of the description or problem. A blind person might rely on and need such diagrammatic representations just as much as a sighted user does and appropriate tactile representation may play that role.

For the sighted users, standard textbooks of science and engineering are widely available that contain numerous examples illustrated with diagrams. On the contrary, similar textbooks with embossed illustrations are not easily available to the blind students in India. In a classroom of sighted students, teachers can explain any topic or problem with free-hand sketch of suitable diagrams on blackboard. They can also use computer-based teaching (CBT) tools for better presentation of text and graphics. For the blind students, there are Braille image embossers and compatible graphics programs for desktop production of tactile graphics. Few sophisticated audio-tactile, audio-haptic, and multimodal interfaces have been developed to produce diagrams in a different non-visual form accessible to the blind students. A detailed description of some of these systems have been given in Section 2.2 of Chapter 2.

A practical problem lies in the fact that most of the systems referred above are not commercially available. Braille (image) embossers are available and used in developed countries but they are prohibitively expensive[1] for wide usage in developing countries. The schools in developing regions cannot afford to have image embossers or even image enhancer (for translating any pre-existing image to tactile form). At best the students of these schools can have access to traditional Braille text printers that cannot print tactile image.

The poor state of use of tactile graphics prevailing presently in the blind schools in India and in other developing countries is reported in the literature [68, 65]. Though nail board or wooden pieces are sometimes used to perceive simple diagrams in lower grades, the lack of frequent access to tactile diagrams of wide varieties in higher grades seriously limits their science education. For that matter, it is a pedagogical problem for teaching any science subject to the blind students in India. In many parts of India, even today, blind students are forced to leave studying mathematics after 7th or 8th grade because of inconvenience of learning diagrams [68]. On the other hand, Braille text printer is available in the schools so it would be nice if geometry drawings can be printed on Braille paper. But owing to the irregular dot-grid of the Braille text printing system, standard computer graphics programs or tactile graphics programs cannot be directly used to draw diagrams through such medium. As discussed in Chapter 2, *BrlGraphEditor* [69] and *Sparsha Chitra* [4] tools as also the prototype developed by Rahman et al. [68] can convert digital images into suitable form for printing on a Braille text printer. But these tools have distinct limitation in generating tactile geometry diagrams directly from textual description.

In the backdrop of the above scenario, one of the distinct contributions of our research is that we introduce a method [9] by which digital diagrams can be converted to tactile diagrams using traditional Braille text printer which is the cheapest of its kind

[1]Cost of embossers are in the range US$ 5000 to 26,000

and commonly available in most blind schools. The proposed method does not make use of any sophisticated interface rather relies on Braille dot grid with uneven spacing of dots to map elements of a diagram already defined for digital display. Improvisation of such Braille printer (meant to print text only) to represent geometric diagrams is far more challenging than using image embossers with evenly spaced dots. Here, the size or height of the embossed dots cannot be varied and can only be placed within a cell (Braille character). In our study, we address a complete set of algorithms for representing point, straight line, circle, and their combinations using Braille code. Therefore, any simple geometric shape (like triangle, rectangle, angle, etc.) and diagrams comprising many shapes and configurations can be generated through Braille text printer. In this chapter we have also proposed a simple yet useful method [9] to evaluate line and circle drawn in Braille by quantifying the approximation errors. The total error of a Braille diagram is also computed, which gives a measure of the accuracy of Digital to Braille mapping.

As our proposed method is capable of producing geometric diagrams on low-cost Braille text printer, it will certainly provide significant benefit to the blind students in accessing diagrams. But to enjoy the benefit, assistance of a sighted person (may be the teacher) will be required who will create the basic digital diagram upon reading a geometric description. Else, the blind students themselves will not be able to generate Braille diagrams according to a description to comprehend it or solve a geometry word problem by their own. To address this issue we have further enhanced our Braille drawing module by integrating it with our t2d system ([7]). As a result of this integration (refer Figure 6.1), given a geometry word problem as input, the system automatically generates diagrams (optionally step-wise) on Braille text printer. This provides the proposed system with the much needed utility of a self-learning tool. The blind students simply need to type the text (of a problem) using a Braille type writer to get the system-generated tactile diagram illustrating the text (problem). This option is also helpful for the teacher who can just key in a text and easily generate multiple copies of the representative tactile diagram. This saves the teachers effort wasted otherwise in creating and presenting accessible graphics for the blind students in a class.

A salient point of our research on Braille-based diagram representation is the series of field tests [12] conducted over a span of seven months with seven groups of blind students and teachers belonging to a blind school and also working professionals who are blind. The acceptance of our system was reflected in the interest generated in our subjects; most of them could recognize the braille diagrams with reasonable accuracy and could relate the diagrams to the problem statements within a short learning time. The results of the tests clearly underline the need and impact of such an affordable learning aid on the quality of science education for the blind community in India.

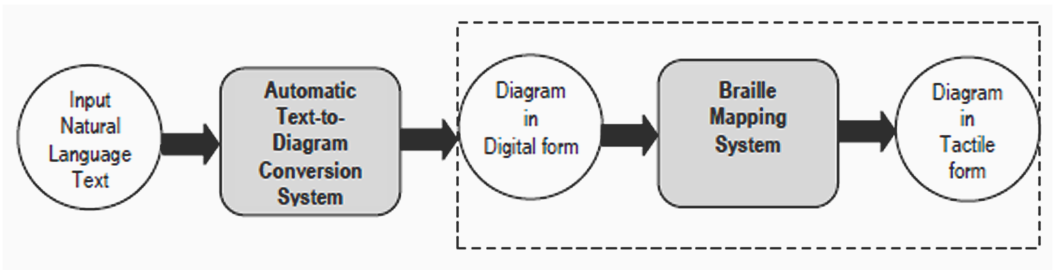

Input Natural Language Text
Automatic Text-to-Diagram Conversion System
Diagram in Digital form
Braille Mapping System
Diagram in Tactile form

Figure 6.1: The connection between the t2d system and the proposed Braille mapping system (within dotted boundary)

The evaluation and user study [12] of our integrated automatic Braille drawing system [9] is done at a blind school in Eastern India tests how easily a teacher can present geometry diagrams (on embossed paper) to the blind students while explaining a topic or problem in the class. More importantly it tests how, without any human assistance or pre-existing printed diagrams, a blind student can create and access tactile geometry diagrams on their familiar medium of accessing text.

6.2 Drawing in Braille

6.2.1 Computer Graphics vs Braille Graphics

Braille is a system commonly used by blind people to read and write. As far as computer graphics is concerned, the smallest addressable picture elements are pixels. In Braille, the smallest physical unit is an embossed dot, while the smallest logical unit is a Braille cell or character - a combination of six such dots arranged in a 3 x 2 array. While each pixel can display a color according to the bitmap value, each dot of a Braille cell may be in embossed state depending on the NUMBRL code [15] of the character to be represented in that cell.

NUMBRL is basically a numeric code that represents the dot patterns in Braille cells. Each dot in a cell has a fixed position value (as shown in Figure 6.2). The NUMBRL code of a cell is just the sum of the position values of the embossed dots of the cell. For example, consider a cell with embossed dot positions 4-2-6. The NUMBRL code for this cell is 1+20+4 = 25.

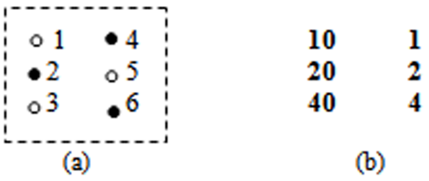


Figure 6.2: (a) Dot positions, (b) Position values

As there are 6 dots in a Braille cell the number of different patterns or NUMBRL codes that can be generated in a cell by embossing a subset of 6 dots at a time is 2^6 or 64. This implies 64 different Braille characters can be printed by a Braille text printer. A given NUMBRL code corresponds to a character of English alphabet or a punctuation symbol or a digit.

In digital graphis [26], we can algorithmically select and illuminate a set of pixels to generate a line or circle. Similarly, in Braille, the first problem is to identify which dots in which cells are to be embossed for the best possible representation of a line or circle. Next cell-wise patterns of identified dots or the NUMBRL codes, each representing a Braille character, can be passed on to a Braille text printer for producing the Braille-text version of a line or circle.

6.2.2 Graphics Algorithms for Braille

Let us consider a grid of pixels that resembles the actual spacing of dots of a 2D array of Braille characters or cells. We now discuss how dot positions can be identified algorithmically in the grid to best represent lines or circles.

Line Drawing

For embossing a straight line in Braille we have used the *mid-point subdivision* method. The midpoint subdivision method is based on a recursive algorithm (*Midpoint_line*) that keeps on finding the midpoints of bisected line segments and subsequently finds the nearest Braille dot from the newly found midpoint *pm (x,y)* in each iteration. When the length of a divided line segment becomes less than the minimum distance between two adjacent Braille dots (say, *n*), the recursion unwinds. The arguments of the function are the line endpoints *p1 (x1,y1)* and *p2 (x2,y2)* respectively. Figure 6.3 illustrates the sequential selection of Braille dots in the grid while line *p1p2* is processed.

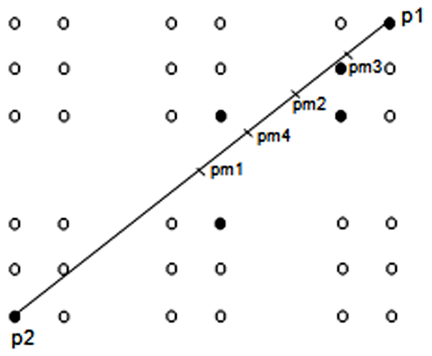


Figure 6.3: Line drawing in Braille - selected Braille dots (shown in thick black dots) corresponds to midpoints *pm1, pm2, pm3, pm4* found in the upper half of the line *p1p2*. Similarly for the lower half of the line there will be few more Braille dots selected.

Pseudocode of the function Midpoint_line( )

```
Midpoint_line(p1, p2)
Input:  Coordinates of two pixels (p1 and p2) which are the end
points of a digital line
Output:  A list of Braille dots (positions) representing the
corresponding Braille line

n:  minimum distance between two adjacent Braille dots

{
   pm = Midpoint(p1,p2)
   Find_nearest_Braille_dot(pm)
   if Distance(pm,p1)>n
      Midpoint_line(p1,pm)
   end if
   if Distance(pm,p2)>n
     Midpoint_line(pm,p2)
   end if
}
```

The functions *Midpoint( ), Distance( )* are standard functions to calculate the midpoint between two points and distance between two points respectively.

Find_nearest_Braille_dot( ) is the function used to search a Braille dot that lies closest to the newly found midpoint *pm* on the line path. The function measures and compares the distance of *pm* from each of the Braille dots that fall within a small square region (2n x 2n) around *pm*. The dot that gives the least distance is the nearest dot found in the search and its position is saved in a file. If there is more than one such dot, only the one found first will be selected as the nearest Braille dot.

Pseudocode of the function Find_nearest_Braille_dot( )

```
Find_nearest_Braille_dot(p)
Input:  A pixel p whose x and y coordinates are p.x and p.y
Output:  Position of the nearest Braille dot corresponding to p
written to a file

n:  minimum distance between two adjacent Braille dots

{
Set Dmin = max_value
 for (i = p.x-n to p.x+n)
   for(j = p.y-n to p.y+n)
     if Braille_dot(i,j) = TRUE
        D = Distance((p.x,p.y),(i,j))
        if D < Dmin
          Dmin = D
          Nearest_dot = (i,j)
        end if
     end if
     j = j+1
   end for
   i = i+1
 end for
 Save Nearest_dot to File
}
```

It is to be noted that like the midpoint algorithm, other standard algorithms for drawing digital lines (e.g. *DDA* or *Bresenham's* algorithm ([26]) can also be modified by applying the function *Find_nearest_Braille_dot( )* to select nearest Braille dots corresponding to digital pixels generated by the algorithms.

Circle Drawing

As far as drawing curved line in Braille is concerned, the popular midpoint algorithm ([26]) for drawing digital circle is suitably improvised to create the Braille function *Midpoint_circle( )*. The change that we have made in the midpoint algorithm is in the function for generating 8 symmetric pixels (one in each octant) on the digital circle in every iteration. For every pixel position *p (x,y)* determined in the first octant, the function *Generate_symmetric_Braille_dot( )* is called within *Midpoint_circle( )*. This function then finds a Braille dot nearest to the circle path by calling the function *Find_nearest_Braille_dot( )* corresponding to each of the eight symmetric pixel positions including *p*. The process is illustrated in Figure 6.4.

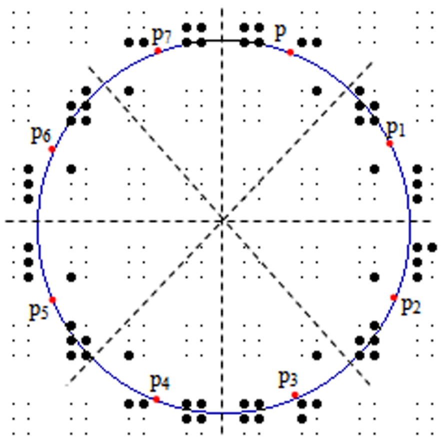


Figure 6.4: Circle drawing in Braille - The digital circle is represented by continuous line. The circle is divided into octants with dotted lines. Corresponding to pixel *p* (shown in faded thick dot) on the circle in the first octant, *p1-p7* are 7 symmetric pixels in 7 other octants. One Braille dot is selected nearest to each of these pixels. The thick black dots are all selected dots that represent the Braille circle.

Pseudocode of the function Midpoint_circle( )

```
Midpoint_circle(c,r)
Input:  center:  c, radius:  r

Output:  A list of Braille dots (positions) representing the

circle p:  a pixel on the digital circle whose x and y

coordinates are p.x and p.y

{
  p.x = 0, p.y = r
  Q = 1-r
  while (p.x < p.y)
  {
    if Q < 0
      p.x = p.x+1
      Q = Q+2*p.x+1
    else
      p.x = p.x+1
      p.y = p.y-1
      Q = Q+2*(p.x-p.y)+1
    end if
    Generate_symmetric_Braille_dot(c,p)
  }
}
```

Pseudocode of the function Generate_symmetric_Braille_dot( )

```
Generate_symmetric_Braille_dot(c,p)
Input:  center:  c, a pixel on a circle:  p
Output:  Eight Braille dots corresponding to the eight symmetric
pixels on the digital circle

p1:  a symmetric pixel on the circle corresponding to p

{
  p1.x = c.x+p.x; p1.y = c.y+p.y;
  Find_nearest_Braille_dot(p1)
  p1.x = c.x+p.x; p1.y = c.y-p.y
  Find_nearest_Braille_dot(p1)
  p1.x = c.x-p.x; p1.y = c.y+p.y
  Find_nearest_Braille_dot(p1)
  p1.x = c.x-p.x; p1.y = c.y-p.y
```

```
    Find_nearest_Braille_dot(p1)
    p1.x = c.x+p.y; p1.y = c.y+p.x
    Find_nearest_Braille_dot(p1)
    p1.x = c.x+p.y; p1.y = c.y-p.x
    Find_nearest_Braille_dot(p1)
    p1.x = c.x-p.y; p1.y = c.y+p.x
    Find_nearest_Braille_dot(p1)
    p1.x = c.x-p.y; p1.y = c.y-p.x
    Find_nearest_Braille_dot(p1)
}
```

6.2.3 Digital to Braille Mapping

In the previous sections we have shown how the standard algorithms for drawing digital lines or circles (using pixels) are modified to represent lines or circles using Braille text characters. Now to demonstrate the actual implementation of the Braille mapping system we have made a simulation of Braille character mapping of geometric entities. The computer screen emulates the grid of Braille cells (each having 6 dots) printed by a traditional Braille printer; the array of 6-pixel groups displayed on the screen resembles the actual spacing of dots in the Braille grid. A standard size of Braille sheet is 11x11 inches and some Braille text printers can print 28 rows and 42 columns of Braille cells (or a periodic, but uneven grid of 84x84 dots) in a Braille sheet. One Braille dot having diameter of 1.5mm is represented by a pixel. The distance between two adjacent dots in a cell is kept uniform (2.4 mm) while the horizontal and vertical distances between two corresponding dots in adjacent cells are set unequal (6.8 mm and 10.1 mm respectively) as usually found in the traditional Braille system. In our experiment with the integrated system, we have used an Automatic Braille Embosser (BPRT) developed by Webel Mediatronics Limited (an India Govt. undertaking organization). This is basically a Perkins Brailler and it costs about US $3000.

Braille Mapping of Basic Entities

The functions *Midpoint_line( )* or *Midpoint_circle( )* generate a set of Braille dots that represent in Braille a digital line or circle respectively. Screen pixels at all such selected dot positions are marked with thick dots to emulate actual embossing of dots by a Braille text printer. By searching in an array that previously stores coordinates of all dots of all cells, it is then found which selected dot belongs to which cell. Position values of the selected dots in a cell are then added up to find the NUMBRL code of that cell. If there are no dots selected in a cell, the NUMBRL value of that cell is 0 which implies that the cell doesnt participate in Braille mapping. This process is illustrated in Figure 6.5.

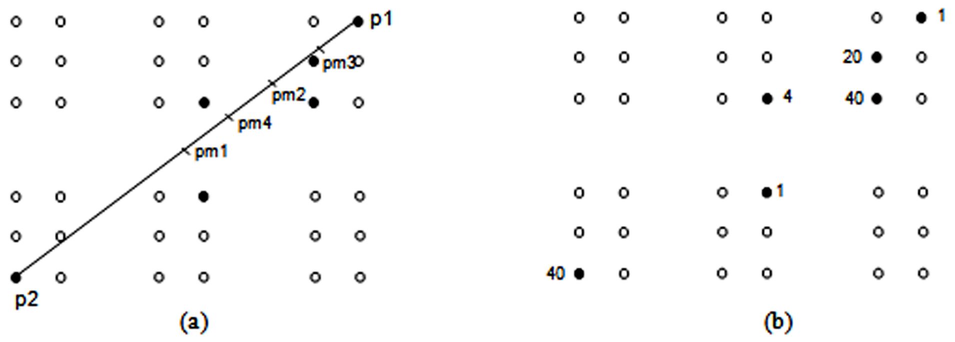


Figure 6.5: (a) Corresponding to each digital point (*pm1*, *pm2*, etc.) on a line, a Braille dot (thick dot) is selected, (b) The NUMBRL position values shown for selected dots in a Braille cell; the corresponding NUMBRL codes are (left to right, top to bottom): 0, 4, (20+40+1 =) 61, 40, 1, 0. The output list given by the Braille function contains the following triplets: < 110 > < 124 > < 1361 > < 2140 > < 221 > < 230 >

Pseudocode of the function for finding NUMBRL code of a participating Braille cell

```
Find_NUMBRL()
Input:  List of dots (positions) selected by the Braille drawing
algorithm
Output:  List of NUMBRL values of all Braille cells written to a
file

i:  Braille cell row position, j:  Braille cell column position

pos:  position value of a dot

{
for all (i,j)
   NUMBRL(i,j) = 0
end for
for every dot d in the input list
  Find the Braille cell (i,j) corresponding to d
  pos = position value of d in cell(i,j)
  NUMBRL(i,j) += pos
end for
for all (i,j)
```

```
  write(i,j,NUMBRL(i,j)) to File
end for
}
```

Braille Mapping of Problem Diagrams

The set of numerical values (parameters) of component lines and circles of a digital diagram (corresponding to a geometric statement or problem) is taken as input by the Braille mapping module. For example, the numerical values of the endpoint coordinates of the four component lines of a digitally drawn parallelogram are, (say):

```
AB: line (100,200),(50,100)
BC: line (50,100),(250,100)
CD: line (250,100),(300,200)
DA: line (300,200),(100,200)
```

The t2d system being integrated with the Braille mapping system, each of the above output is passed on as input (argument) to the *Midpoint_line( )* function separately. This function along with the *Find_NUMBRL( )* function then creates four Braille lines to produce the Braille version of the *parallelogram* `ABCD`.

A portion of the emulated Braille dot grid and embossed dots approximating different geometric shapes are shown in Figure 6.6. Initially we generated entities like straight line and circle (Figure 6.6(a) and (b) respectively) from corresponding digital representation. This is followed by generating simple geometric objects like parallelogram (as in Figure 6.6(c)), triangle (as in Figure 6.6(d)), rectangle, square, pentagon, hexagon, etc.

After generating simple objects, we have tested our system for diagrams comprising multiple objects. The system, upon feeding the numerical values of the defining parameters of the digital diagrams, yielded Braille diagrams for a test set of 32 geometry problems. The simulated Braille diagrams for 2 out of 32 problems are cited in Figure 6.7.

In the Braille objects (or diagrams) finally produced on Braille sheet, points are not labeled automatically and that is done intentionally to keep our study focused on the use of Braille for conveying a diagram rather than text. Labeling diagrams at this stage

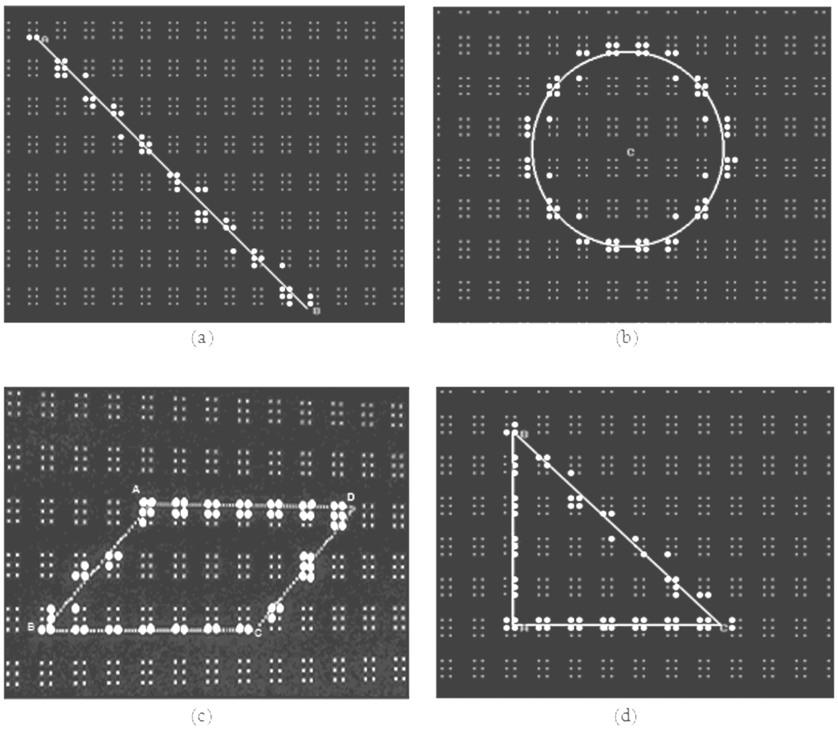
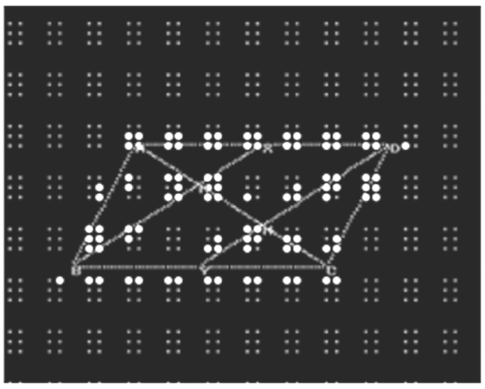


Figure 6.6: Simulation of Braille character grid and Braille-converted entities; the digital entities are shown in continuous lines while selected Braille dots are shown in solid dots. (a) line, (b) circle, (c) parallelogram, (d) triangle

might confuse users and reduce their recognition accuracy. Once the users are familiar with Braille character chains as diagrams, labels can be generated along with diagrams to better illustrate the problem description. We have put the labels of the points in the figures here for the sake of illustration only.

6.2.4 Design Optimization

It is seen that the numerical values passed as arguments of Braille drawing functions often cause entities produced in Braille too small in size to be separately recognized by touch only. To improve perception of a Braille-mapped figure by a blind user and also to improve drawing accuracy the better option is to generate figures in a larger scale. Hence, depending on the size, a digital diagram is first resized so that it can fit to the full-screen. Then digital to Braille mapping is done for the enlarged diagram causing the simulated Braille diagram also to fit to the screen. This in turn yields longer lines and



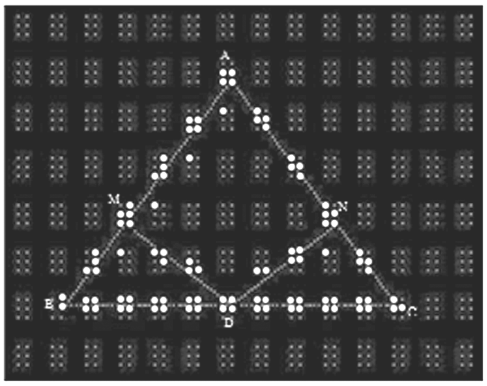


Figure 6.7: Simulated diagrams for the problems: (a) *ABCD is a parallelogram. X and Y are the midpoints of AD and BC. BX and DY cuts AC at M and N respectively. Prove that AM = MN = NC,* (b)*In triangle ABC, AB = AC. D is the midpoint of BC. From D, perpendiculars DM, DN are drawn to AB and AC respectively. Prove that DM = DN*

larger circles in the printed Braille diagram. It may be noted that Figures 6.6(a)-(d) are each screenshot of a portion of the screen whereas Figures 6.7(a),(b) are screenshots of the entire screen area.

Despite the inherent poor resolution of the Braille system, we have tried to improve the quality of the Braille diagrams by applying some heuristic methods. Earlier the *Find_nearest_Braille_dot( )* function selected too many dots. To retain optimum number of dots, we start with a reduced range (in x and y direction) for searching of dots near to a digital pixel by the Braille functions *Midpoint_line( )* or *Midpoint_circle( )*. If no dot is selected, the range is increased in the next iteration and the process is repeated until a prefixed maximum range is attained.

Pseudocode of the optimized function Find_nearest_Braille_dot( )

```
Find_nearest_Braille_dot(p)
Input:  A pixel p whose x and y coordinates are p.x and p.y
Output:  Position of the nearest Braille dot corresponding to p
written to a file
n:  minimum distance between two adjacent Braille dots
w:  length used to define the window for searching a Braille dot
Found:  flag that indicates whether any dot is found within
```

```
search window

{
set Dmin = max_value
set Found to 0
set w to n/2
while (NOT Found AND w <= 2*n)
{
  for (i = p.x-w to p.x+w)
    for(j = p.y-w to p.y+w)
        if Braille_dot(i,j) = TRUE
          Found = 1
          D = Distance((p.x,p.y),(i,j))
          if D < Dmin
            Dmin = D
            Nearest_dot = (i,j)
          end if
        end if
      j = j+1
    end for
    i = i+1
  end for
  w = w+n/2
}
If (Found) Save Nearest_dot to File
}
```

Again, another problem noted was selection of more than one dot in the same row for an inclined line as is evident from Figure 6.6(a). This might confuse a blind user who while tracing a line path would expect a single raised dot at any position on the line path. This redundancy is eliminated by comparing the distances (from the line path) of all the selected dots in a row and retaining only the dot at minimum distance. However, the elimination of extra dots is avoided if it causes a wide gap between the retained dot and the next selected dot in the adjacent row along the same line path. The maximum allowable gap is the distance between two diagonally adjacent cells. Overall, the Braille figures produced after implementing the design optimizations in the system are qualitatively improved output in the sense the blind students could recognize those figures with greater accuracy. The resultant simulated diagrams (e.g. those shown in Figure 6.8) reflect this optimization effect.

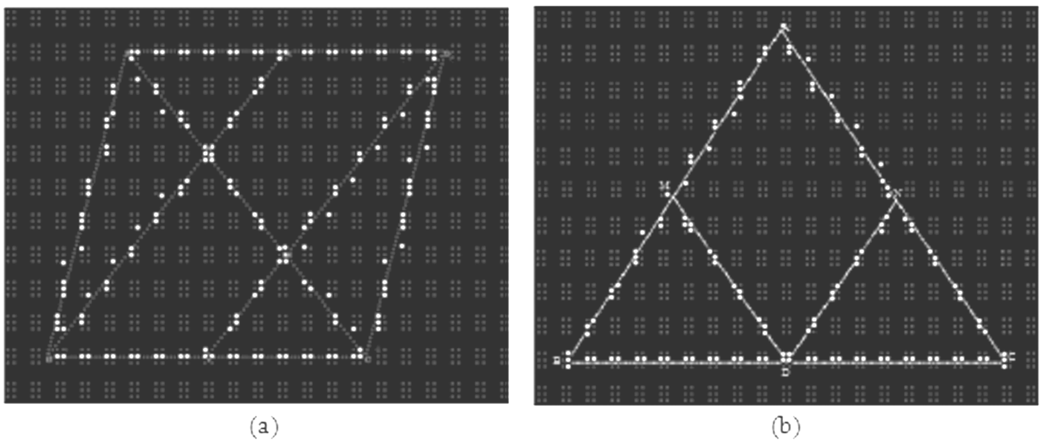


Figure 6.8: (a) Modified simulated diagram for the problem in Figure 6.7(a), (b) Modified simulated diagram for the problem in Figure 6.7(b)

6.2.5 Calculation of Error

The pixel resolution of computer display being much higher than the dot resolution of Braille system, the approximation of entities in the latter system is naturally much coarser than that in the former one. Another reason behind this approximation error is the uneven spacing of dots in the traditional Braille system. This error if quantified will have a bearing on future research on improving the output and thereby the efficacy of the system in supporting mathematics education of the blind students. Here we have applied a standard method to measure the approximation errors in Braille mapping of line and circle which are the basic components of geometry diagrams.

Error in Line

Figure 6.9 shows the simulated Braille version of a digital straight line. The digital line is shown in continuous line.

The error between the actual line and the Braille version of that line can be found by first calculating the deviation or distance of each selected Braille dot from the actual line path and then finding the average of the squares of all such distances. This average is called the *Mean Square Error (MSE)* of selected Braille dots with respect to the actual line. The general steps for finding the MSE of a Braille line is explained next.

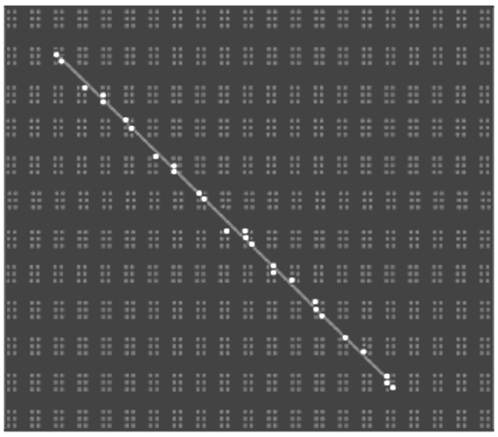


Figure 6.9: A simulated Braille line represented by thick and bright dots; the corresponding digital line is superimposed to show the deviation of the Braille dots from the line path

Step 1: From endpoint coordinates (and slope) of the given line calculate y-value on the actual line corresponding to each sampling position or x-value.

Step 2: At every sampling position find the y-value(s) of the selected Braille dot(s).

Step 3: For every sampling position calculate the difference between the y-value on actual line and the y-value of a selected dot on the Braille line. This difference is the deviation of the Braille line from the actual line or the approximation error at that sampling position.

Step 4: Find the mean square of all the differences calculated in Step 3. This gives the MSE of the Braille approximation of the line.

The MSE of the line shown in Figure 6.9 thus turns out to be 0.035. It may be noted that there may be more than one selected Braille dot at a sampling position. In such case, error or difference should be calculated separately for all selected Braille dots having same x-value.

Error in Circle

Figure 6.10 shows the simulated Braille version of a digital circle shown in continuous line.

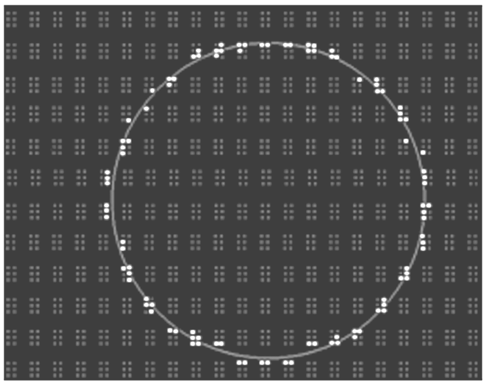


Figure 6.10: A simulated Braille circle represented by thick and bright Braille dots; corresponding digital circle (in continuous line) is superimposed to show the deviation of the Braille dots from the circle path

Similar to the Braille line, error of a Braille circle is found in terms of MSE of the selected Braille dots with respect to the actual circle.

Step 1: For each selected Braille dot calculate the difference between its (Euclidean) distance from the center of the circle and the actual radius of the circle. This difference is the deviation of that Braille dot from the actual circle path.

Step 2: Find the mean square of all the differences calculated in Step 1. This gives the MSE of the Braille approximation of the circle.

For the example shown in Figure 6.10 the MSE of the circle is calculated to be 0.049.

Error Comparison with Other System

It is evident from the methodology of linear mapping from an evenly spaced pixel-grid to an unevenly spaced Braille dot-grid by the system [68], distortions will occur in shapes represented in Braille. Moreover, in this process redundant dots get selected at positions other than the best possible positions with respect to the actual shape line. Our system, on the contrary, logically finds the dot nearest to the actual entity path at any sampling position and also optimizes the number and position of dots getting selected. This causes better representation of diagrams in Braille.

For a quantitative comparison of Braille lines produced in our system to that produced by the system [68], we have calculated the MSE of a Braille line produced by the latter system (see Figure 6.11). For this we have considered the same digital line as shown in Figure 6.9.

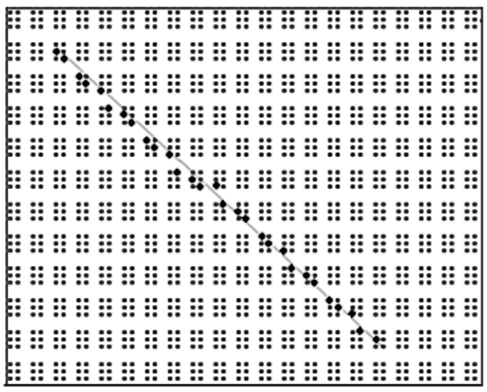


Figure 6.11: A simulated Braille line created using the method in [68]

The MSE calculated for the participating dots with respect to the actual line is 0.165. The reason behind getting larger MSE for the system [68] is attributed to the linear mapping from an evenly spaced pixel-grid to an unevenly spaced Braille grid. Because of this mapping, many redundant dots are often selected and necessary dots are missed out. On contrary, the optimization technique used by our method contributes to produce lower MSE.

The simulated Braille approximation of a digital circle following the approach described in [68] is shown in Figure 6.12. The basic parameters (i.e. the radius and center) of the circle are taken same as that used in Figure 6.10. The MSE obtained is 0.147 against 0.049 in our case.

As is evident from the simulated figures (Figures 6.9- 6.12) of Braille lines and circles produced in both the systems, qualitatively (visual appearance) the shapes are much better in our system.

To find a measure of the errors in representing a geometric object (like triangle, parallelogram, etc.) or a diagram (e.g. diagrams shown in Figure 6.6, 6.7, 6.8, etc.) in Braille,

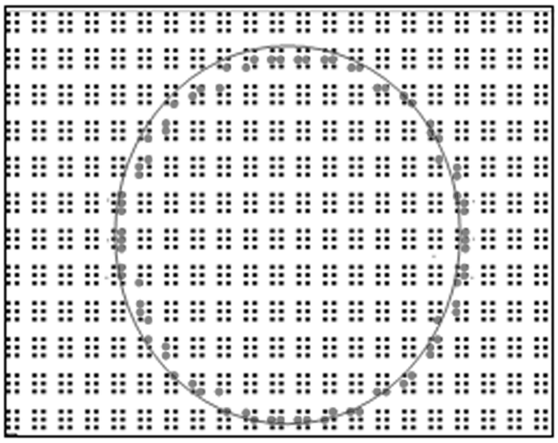


Figure 6.12: A simulated Braille circle created using the method in [68]

we first evaluate the MSE of each constituent line and circle of a diagram. Then we find the cumulative MSE (of all the entities) which when divided by the number of entities gives the average MSE of a diagram. Table 6.1 lists the average MSEs (of same figures) calculated in two systems (ours vs. [68]).

Table 6.1: Comparison of MSEs of Braille Diagrams

Shape/Diagram	**Component Entity**	**MSE in our System**	**MSE in System [68]**
Equilateral Triangle	one horizontal line, two inclined lines	0.020	0.047
Right angled Triangle	one horizontal line, one vertical line and one inclined line	0.012	0.028
Rectangle	two horizontal lines, two vertical lines	0	0
Parallelogram	two horizontal lines, two inclined lines	0.026	0.104
Hexagon	two vertical lines, four inclined lines	0.015	0.049
Diagram in Figure 6.8(a)	two horizontal lines, five inclined lines	0.023	0.080
Diagram in Figure 6.8(b)	one horizontal line, four inclined lines	0.022	0.077

Thus we find from the above table that the average errors in Braille diagrams are much more in [68] compared to those in our system.

For our system we have evaluated Braille diagrams automatically produced from 32 geometry word problems - the same set which produced 100% correct digital diagrams upon conversion of problem text [7]. The MSE for these 32 problems varied from 0 to 0.055. It is observed that the MSE increases if number of inclined lines and circle increases whereas the increased number of vertical and horizontal lines effectively reduces the MSE. For the geometric objects, the error is mostly restricted within 0.020 (for parallelograms we find it worst, i.e. 0.026). The remaining 8 problems from the test set of 40 problems were not evaluated because the *intermediate representations* produced by the NLP module for those 8 problems were not correct.

6.3 User Evaluation and Impact Analysis

Over a period of nearly seven months pre and post development study [12] was conducted at *Blind Boys School, Narendrapur* and the *Blind Person's Association, Baruipur* - the former being a school for the blind students while the latter is a professional organization for the blind people, both situated in Kolkata, India.

6.3.1 Study of Existing Practice

Given the standard of average blind schools in India, the *Narendrapur* school is quite rich in resource. They have couple of desktop Braille printers or TED (text embossing device) and uses *DBT* (*Duxbury Braille Translator*) software to create Braille documents and print those in raised-dot form using TED. However, not many teachers and students have the skill of producing even simple line diagrams with DBT and TED. Tested with a set of 10 geometric problems only one teacher and one blind alumnus could successfully represent two simple problems diagrammatically using those tools. Still slate/stylus and nail-board system continues to be popular medium for teaching geometry in the class. The *Baruipur Association* has a Braille embosser or PED (plate embossing device) with compatible *TGD* (*Tactile Graphics Design*) software (Figure 6.13). Few members often use the TGD and PED to produce tactile documents with images but most of the members do not know how to use the PED.

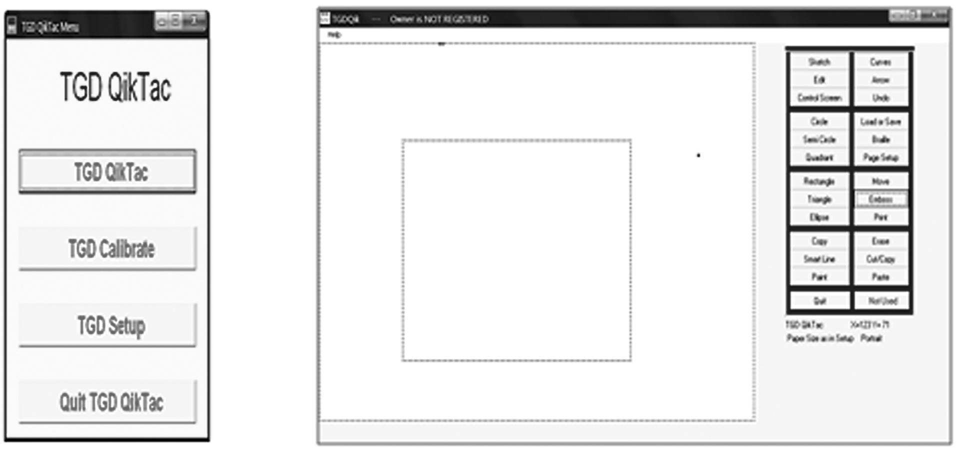


Figure 6.13: TGD QikTac drawing interface; simple geometric shapes can be drawn on screen and then embossed using costly image embossers

6.3.2 Need Analysis

To find out the limitations and problem areas for teaching and learning geometry in a classroom of the blind students, a questionnaire survey was conducted amongst the students and teachers separately. Some sample questions from the questionnaire are given below.

For students:

1. *Have you learnt geometry?*
2. *Have you ever drawn/made any geometry shape? If yes then how? - by using (a) nail-board, (b) wooden pieces, (c) DBT and Braille printer.*
3. *How do you perceive geometry shapes? - by reading tactile shapes printed/created (a) in books, (b) on Braille sheet, (c) using slate-stylus/nail-board/wooden pieces,(d) never perceived shapes.*
4. *Do you face difficulty in learning geometry in class? If yes then why? - (a) cannot visualize geometric description, (b) teachers often do not explain with diagrams, (c) can follow when teacher explains but cannot understand problems by my own.*
5. *Do you want more help for learning geometry? If yes then how? - by (a) providing good text books with more illustrated examples, (b) providing more trained teachers, (c) providing learning kits that can help us draw quick sketches of geometry.*

For teachers:

1. *Do you teach (or have you ever taught) geometry in class?*

2. *Have you learnt geometry in your school days? If yes then upto which standard? - (a) primary school, (b) middle school, (c) high school.*
3. *Do you use tactile geometry shape/diagrams while teaching geometry in the class? If yes then how? - by using (a) slate-stylus/nail-board/wooden pieces, (b) Braille books (c) printouts using DBT and Braille printer.*
4. *Do you face difficulty in teaching geometry in class? If yes then why? - (a) cannot explain geometry only verbally, (b) producing tactile diagrams and conveying it to every student in the class is not easy with nail-board/wooden pieces, (c) creating Braille printouts of diagrams using DBT software is not easy.*
5. *How, do you think, teaching geometry can be made more effective? - by (a) using better drawing tools (e.g. image embosser) to draw and make multiple copies of tactile diagrams, (b) providing specialized training to the teachers, (c) providing easy-to-use self-learning mathematics tools/kits to the students.*

Respective set of questions were asked to each student and teacher in the survey and responses in terms of *Yes* or *No* or options (a), (b), (c) or (d) were recorded. Group discussion was also held involving the principal and teachers of the school and a group of working professionals. The topic of discussion was *Problems of teaching geometry to the blind students*. The following is the summary of the facts that came up after qualitative analysis of data collected through questionnaire survey and discussion.

The school does not have enough funding to afford an image embosser which would have been an ideal solution for generating tactile diagrams to help teaching geometry. Drawing diagrams using DBT/TED is a tedious process. One has to type 6-dot characters at suitable positions such that the character chain represents a geometric shape. See Figure 6.14. In the lower grade, slate/stylus or nail-board is preferred by the teachers to give idea about basic geometric shapes and simple diagrams. Overall, the inconvenience of sharing and accessing diagrams has practically limited teaching geometry to dictating facts and procedures only, deliberately omitting any reference to diagrams. The students have option to avoid geometry questions in mathematics examinations – so they are usually reluctant to learn geometry. Some blind teachers revealed that they themselves were deprived of geometry education owing to lack of learning aids. The majority's opinion was that the nail-board or slate-stylus system is more effective than TGD or DBT though only simple diagrams can be tried with those. The exercise revealed the need for a low-cost utility requiring minimal manual operation to serve as a self-learning tool for the blind students as well as a useful teaching aid for diagram-based subjects.

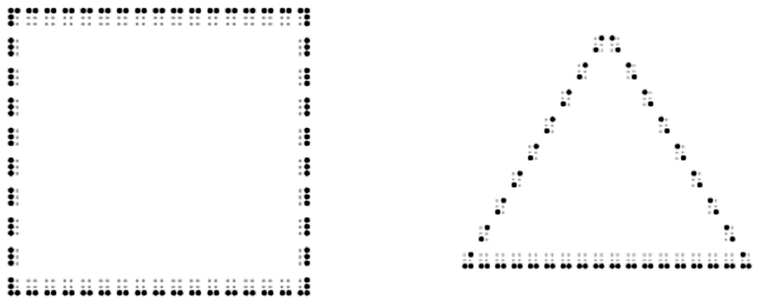


Figure 6.14: Simple geometric shapes printed using 6-dot cells in Braille with the help of DBT software

6.3.3 Selecting Test Subjects

To get a subjective assessment of how different categories of users react to our prototype, seven groups of test subjects as detailed in Table 6.2 were selected. 80% of the student population in the school is completely blind and remaining 20% partially visually impaired. We have selected those students as test subjects who are completely blind.

Table 6.2: Profile of test subjects

Groups	**Composition**	**Average age**	**Visual Impairment**
Group I	4 students of grade VIII (2 male, 2 female)	13 years	Completely blind
Group II	4 students of grade IX (2 male, 2 female)	14 years	Completely blind
Group III	4 students of grade X (2 male, 2 female)	15 years	Completely blind
Group IV	2 senior teachers (1 male, 1 female)	55 years	Sighted
Group V	2 young teachers (1 male, 1 female)	30 years	Sighted
Group VI	2 technical professional (1 male, 1 female)	40 years	Completely blind
Group VII	2 non-technical persons (1 male, 1 female)	30 years	One person completely blind, the other person almost completely blind with deteriorating vision

As the students were children with special needs, communication with them is very different from that with the sighted students. So the sample size in each group was deliberately kept small to impart effective training with limited number of trainers and also to ensure that each reaction or performance of each subject is observed closely. In the student groups at least one student each from high, low and medium rank category were chosen to test whether the system is equally useful irrespective of the academic merit of the subject. In each group 50% female member were included because female literacy/education is a national issue in India; how the blind female subjects perform with a new learning tool may be significant for further study.

6.3.4 Training and Test of Recognition

Firstly, the working mechanism of our integrated system, its input and output were verbally explained to the test groups. Then practical demonstration of the system was given by first showing automatic generation of simple shapes like line, circle, rectangle, triangle (using TED) corresponding to input statements like `AB is a line`, `A circle is centered at O`, `ABCD is a rectangle`, etc. Next figures representing relations between two geometric entities, e.g. `AB is perpendicular to CD`, `Lines AB and CD intersect at E`, etc. were generated and explained. Initially we found most students having difficulty in understanding the output as diagram – they rather read the diagram as Braille text, attempting to pronounce the individual characters that formed the diagram. The teachers were asked to perceive objects through touch only (without seeing) and they behaved at par with the blind students. At this stage their responses with some tactile shapes printed on Braille sheet using our system was recorded and shown in Figure 6.15.

Each group was given 2-weeks initial training @ 1.5 hours each day until they were able to perceive Braille character chains as geometric shapes rather than text. In the beginning, concept of tracing the outline of a shape was given. We guided the student's finger over a figure-outline with verbal commentary and pointed out the key elements of the figure (e.g. sides and vertices of rectangle). Then we trained them to differentiate the characteristics of the shapes by emphasizing the finer aspects (orientation, length, etc.) of an element of a particular shape. Some students struggled to identify shapes as

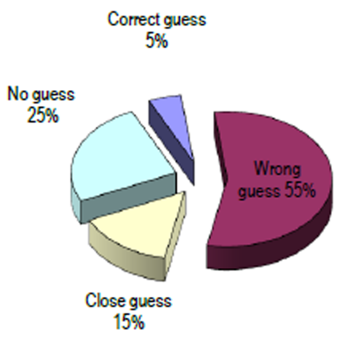


Figure 6.15: Only 5% guesses were correct, the rest were incorrect; some guesses were very close like identifying a parallelogram as a rectangle

they did not carefully trace the entire outline and were still confusing the dots as part of Braille text rather than image. But most subjects could successfully retrace a shape without our help if they were prompted about the object and its starting/ending position. Gradually this support was withdrawn and after a week of practice @ 1 hour each day, the performance of each group in recognizing 10 different geometric objects was recorded (Figure 6.16(a)). The 10 objects are – point, line, square, rectangle, parallelogram, pentagon, trapezium, arc, circle and angle. Next, 10 geometric relations each involving two entities (namely, perpendicular lines, parallel lines, intersecting lines, diagonal of a rectangle, bisecting lines, bisector of an angle, tangent to a circle, diameter of a circle, two concentric circles, two intersecting circles) were given to recognize after another 10 days of training-cum-practice @ 1 hour each day. Figure 6.16(b) shows the performance of each group.

From Figure 6.16(a) we find the mean is 6.29 and standard deviation (SD) is 1.50 for the male subjects, while for the female subjects the mean and SD are 6.71 and 1.38 respectively. The t-value is 1 and p=0.3559 (>0.1). In Figure 6.16(b), mean and SD for male and female performances are 5.57, 1.72 and 5.86, 2.19 respectively. Here also the t-value is 1 and p=0.3559 (>0.1). Therefore, the difference in performance of male and female in recognizing both object and object relation is not statistically significant. However, upon calculating the overall means (6.5 and 5.71)from both the charts we find the recognition accuracy diminishes by 12% when more than one entity features in the diagram.

Students (particularly those belonging to Group II) were quite receptive and they learnt very fast. Out of 10 single objects and 10 configurations (with two objects) given for testing, 7.6 objects and 7.1 relations respectively were recognized correctly by the three student groups on an average. The average student's performance over the two tests was therefore 7.41 or 74.1%. The average performance of the senior teachers was 45% while that of the young teachers was 62.5%. The performance of Group VI and VII were 37.5% and 60% respectively. It was noted that maximum failure occurred for circle, pentagon and parallelogram amongst objects while for tangents, bisector and concentric circles amongst the relations. Similar tests with different object-relation set were conducted at every one week interval and the performance recorded. After one month the average performance of the student groups, teacher groups and the remaining groups were 92.5%, 74% and 67% respectively which shows improvement with practice.

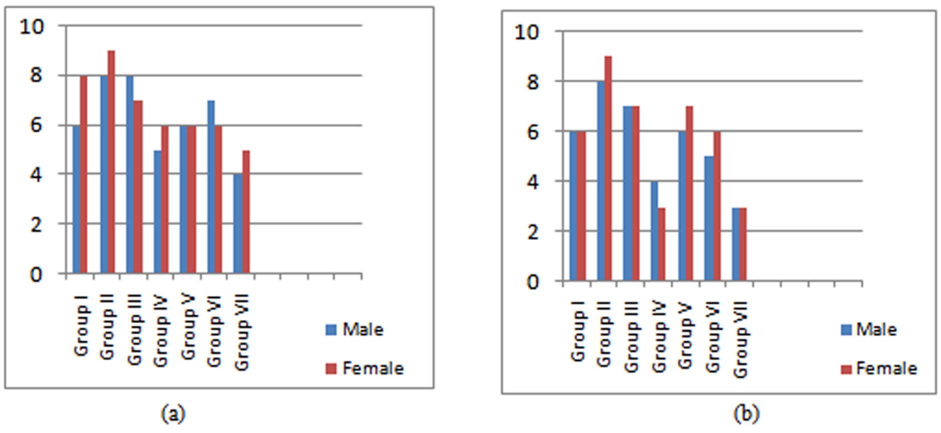


Figure 6.16: Recognition accuracy for: (a) objects generated in Braille, (b) object relations generated in Braille

6.3.5 Test of Classroom Teaching-Learning

A total of 6 experiments, spanning a duration of 3 months, were carried out with each student group having a young teacher with them; the students had a desktop Braille text printer and the teacher was using a desktop PC with our system installed in it. Each time the teacher typed a geometry problem in his machine, the problem-diagram was drawn by the system simultaneously in the simulated form in the teachers' machine and in the tactile form through the Braille printer accessed by the students. The test set contained 10 geometry word problems of varying hardness selected from the set of 32 problems for which the t2d system achieved cent percent accuracy. It was observed that when a diagram was presented as a whole, the students at first could not follow most of the objects and their relations. It seemed difficult for the students to determine whether the elements within a bigger entity (e.g. two smaller lines within a triangle as in Figure 6.8(b)) were part of the outer outline or a detail inside the diagram. Then the teacher was asked to verbally dictate a problem one statement at a time and we checked whether the student could recognize the Braille objects mentioned in the statement. This was effective as the students could now correctly recognize more objects/relations than before, though some students still struggled to find and trace the correct object. More help was then provided by the teacher by orientating those students to a starting position (like the vertex X of a triangle XYZ) and along a tracing direction (like along a side XY from point X). The result was much better this time.

After this initial support, the recognition accuracy was measured in terms of number of diagrams perfectly identified and traced. If they misinterpreted any one object or relation of a diagram we considered it a failure to recognize the diagram. With repeat tests conducted every 15 days (same problems given in random order), the performance of the subjects gradually improved (see Figure 6.17). To minimize the chance that the students could memorize the patterns and therefore recognize correctly, they were not told about what they did was correct or not in each test. But in-between two tests usual practice (on an average 1 hour each day) was given on other simple shapes and diagrams drawn on Braille. In the last test (Test 6) conducted at the end of 3 months, Groups II and III emerged most successful (7 diagrams recognized correctly out of 10, i.e. 70% success). The score of other two Groups were 6 (60%) and 4 (40%) and the average final score of all the groups was 6 (60%). If we look at the respective minimum scores of 3 (30%), 2 (20%) and 1(10%) in the beginning of the test series then the improvement achieved in 3 months is statistically significant (t-value 9.79, $p=0.0023 < 0.01$) and quite encouraging as far as trainability of the technology is concerned.

It was commonly found that as the number of geometric entities/relations increased the problem-diagrams became more complicated and therefore the subjects misinterpreted one or more objects/relations. However, the recognition accuracy and speed of tracing increased by around 20% on an average in case of step by step drawing supported by verbal guidance. This result clearly indicates the potential of our system as a useful and effective classroom tool for the blind pupil.

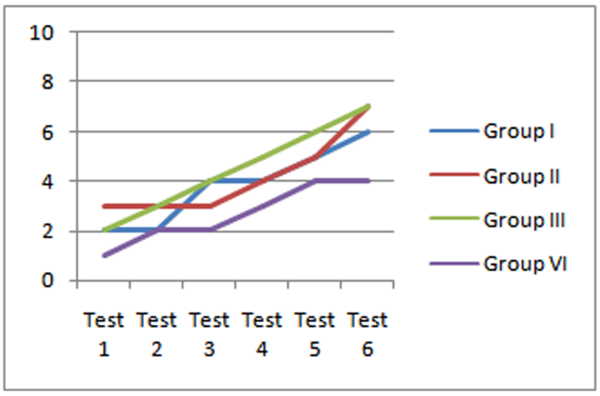


Figure 6.17: Gradual improvement of performance of each group; performance measured in terms of number of problems (along vertical axis) recognized correctly

6.3.6 Test of Self-Learning

Finally the students self learning ability upon using the system was tested. with four groups for more than 1 month. A total of 10 problems were tested @ 2 tests every week. In between tests, the participants practiced themselves and training/help was seldom provided during this period. They were asked to type the text of a word problem as input to the system upon reading the same already printed in Braille. Then they were asked to auto-generate the diagram in Braille (which requires a simple key press) and explain the problem with reference to the diagram. All the subjects operated the system comfortably without any significant help and could correlate the problem texts with parts of the diagrams. The average time taken by each group from typing to tracing the auto-generated diagram for each problem is shown in Figure 6.18. Though initially the time consumed was quite high for all the test groups, later on it came down to 45 minutes for a 3 or 4 line problem. If we can pre-feed the problems in the system (as maximum time was taken for typing a problem) then time required by the subjects to generate and trace the diagrams would come down to around 15 minutes per problem.

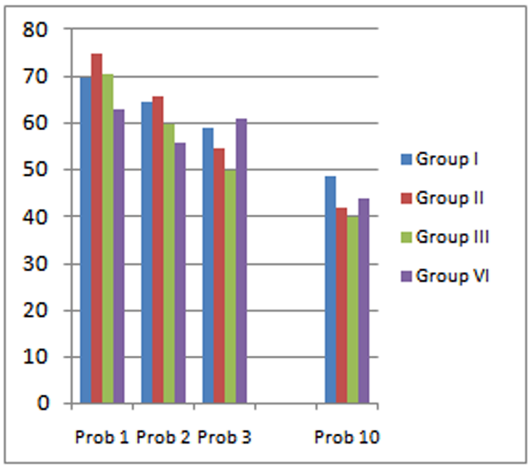


Figure 6.18: Time taken (in minutes) to draw and trace a problem using the new system

6.3.7 Result and Analysis

After a series of trainings followed by tests conducted for around 4.5 months, a questionnaire survey was conducted again to get quantitative data on user perception about the new system. Beside the test groups (including 12 students and 8 seniors), 14 more students (i.e. a total of 26 students) participated in this survey as those students (who did not take part in the training) by this time got used to the system knowing from their peers of Group I, II and III. Questions were framed to bring out the average value of the evaluation indexes like satisfaction, ease of use, impact and effectiveness. The different indicators under each index and user response against those are illustrated in Tables 6.3-6.6. Each indicator was assessed on a five-point *Likert scale* that ranged from 1 (strongly disagree - SDA) to 5 (strongly agree - SA). The values in the columns SA to SDA are the percentage of number of users responded in respective category. Considering the response percentage against each category (SA to SDA) as frequency (f) of occurrence of respective scale values (x), we calculate the mean as $\overline{x} = (\sum \mathrm{xf})/100$ and standard deviation as $\sigma = \sqrt{\sum (\mathrm{x}^2\mathrm{f})/100 - \overline{x}^2}$against each indicator.

The average mean of the satisfaction index (Table 6.3) is 3.98, which implies user's comfort in and positive attitude towards using the system. Only 5% (average) users disagreed that the tool is useful, helpful and can be effectively used in the classroom.

Table 6.3: Satisfaction

Indicators	**SA (5)**	**A (4)**	**N (3)**	**DA (2)**	**SDA (1)**	**Mean ($\overline{x}$)**	**SD (σ)**
Satisfied with the tool and willing to use it in class now; could not use the existing tools so effectively	22	64	8	3	3	3.99	0.83
Perceived as a useful tool as any and many geometry problem can be discussed during a class	14	56	22	8	0	3.76	0.79
Perceived as an effective tool as there will be more time for teacher-student interaction	17	66	11	6	0	3.94	0.72
Perceived as a helpful tool as a student can practice geometry problems by his own thus enhancing self-efficacy and confidence level	33	56	11	0	0	4.22	0.63
Average						3.98	0.74

Table 6.4: Ease of Use

Indicators	**SA (5)**	**A (4)**	**N (3)**	**DA (2)**	**SDA (1)**	**Mean ($\overline{x}$)**	**SD (σ)**
The interface is user-friendly	11	45	28	8	8	3.43	1.85
The operations are simple	22	61	11	3	3	3.96	0.85
It requires no external help	22	56	11	8	3	3.86	0.95
Tracing geometric objects is easy	11	44	6	22	17	3.10	1.33
Flexibility of storing and reusing a diagram once produced	28	61	11	0	0	4.17	0.60
Average						3.70	1.12

The Technology Acceptance Model theory (Davis, 1989) states that acceptance and use of a technology is determined by two factors: perceived usefulness and perceived ease of use. From the above statistics, the perceived usefulness is given by $\overline{x}$ = 3.76, σ = 0.79 (Table 6.3) and the average mean for ease of use is 3.70 (Table 6.4). Simple operation ($\overline{x}$ = 3.96, σ = 0.85) (Table 6.4) and flexibility of storing the diagrams for future reuse ($\overline{x}$ = 4.17, σ = 0.60) (Table 6.4) are the two aspects accepted by majority. However, there are several reasons behind low scoring against *Tracing geometric objects is easy* ($\overline{x}$ = 3.10, σ = 1.33) (Table 6.4). Firstly, the subjects were habituated in recognizing the 6-dot Braille characters as text only. With lot of practice and training they were gradually successful in recognizing the Braille character chains as part of diagrams - it was definitely not so easy

Table 6.5: Impact

Indicators	**SA (5)**	**A (4)**	**N (3)**	**DA (2)**	**SDA (1)**	**Mean ($\overline{x}$)**	**SD (σ)**
Interesting	39	61	0	0	0	4.39	0.49
Enjoyable	19	58	17	6	0	3.90	0.77
Like	28	56	13	3	0	4.09	0.72
Average						4.13	0.66

Table 6.6: Effectiveness

Indicators	**SA (5)**	**A (4)**	**N (3)**	**DA (2)**	**SDA (1)**	**Mean ($\overline{x}$)**	**SD (σ)**
Accurately represents a problem	6	41	28	17	8	3.20	1.05
Minimizes user operation	28	72	0	0	0	4.28	0.45
Minimizes cognitive load of students	19	69	6	6	0	4.01	0.70
Minimizes effort of teacher	30	56	6	0	8	4.00	1.04
Increases learning outcome	33	64	3	0	0	4.30	0.52
Average						3.96	0.75

that they could trace the diagrams correctly at first attempt. If the students are exposed to Braille text based diagrams from lower classes, then tracing complex diagrams will be much easier in higher classes. Secondly, there were some senior teachers and non-technical participants who had a mindset of not welcoming the new technology - they rather felt comfortable with the traditional methods even after accepting limitations of those methods. The young teachers are more receptive and can always be trained easily on this new technology. This is supported by the test result shown in Figure 6.17.

Impact gives a general evaluation of the user's reaction to the new system, especially when they used it for the first time. The average mean of 4.13 (Table 6.5) implies that the users liked it in-spite of possible impeding factors like typing a problem or tracing a diagram which may be time-consuming and training-dependent. Effectiveness measure as shown in Table 6.6 estimates whether the new system fulfills its specific educational and pedagogical objectives for the target users for whom it was designed. The low score ($\overline{x} = 3.20$, $\sigma = 1.05$) against *Accurately represents a problem* (Table 6.6), may be attributed to the inherent poor resolution and non-uniform spacing of the Braille dots which causes unevenness in shapes. It obviously does not mean that the Braille diagrams are geometrically incorrect, rather implies that the smoothness of the Braille lines or circles is not as good as those created using nail-board-elastic band/slate-stylus/PED which produce better perception of tactile geometric objects. Few participants had some previous exposure to figures drawn using those tools/devices. But as a teaching-learning tool our system is quite effective due to certain user-perceived advantages – firstly, diagram drawing is totally automated and it minimizes manual operations which in turn helps to reduce the cognitive load of the student and effort of the teacher in a class. Secondly and more importantly, the students can self-operate the system and learn without assistance. That's why the average mean (3.96) of effectiveness (Table 6.6) and user's perceived learning outcome ($\overline{x}$ = 4.30, σ = 0.52) (Table 6.6) are quite high.

6.4 Summary

This chapter clearly explains the real world need for generating low-cost tactile graphics for providing better science education to the blind students in the developing countries. A method for converting digital diagram into graphic pattern of Braille text has been implemented. Later on, this approach is integrated with automatic text-to-diagram conversion module. Using the combined system one can give (by typing) a geometry word problem as input to the system and get the corresponding diagram as Braille printout. The techniques used are simple yet pragmatic as only low-cost traditional Braille text printer is used instead of sophisticated and costly devices (like the image embosser). The system is installed at a Blind school in India and rigorously evaluated by the target users. The impact analysis brings out the potential of the system as an effective teaching/learning tool for the blind students. The enthusiasm and motivation of the target group behind using this system promises that this low-cost learner-centric tool could be a sustainable solution in the typically resource-constrained Blind schools in India.

It is evident from the simulated diagrams that the lines or circles that may be generated using our system are not as smooth as those which are manually set or embossed using image embossers. The error in a Braille diagram has been measured. However, it is experienced that small errors do not hamper much the effective cognition of a diagram by the target users.

The experiments with the system hint that there is enough scope for improvement in the Braille conversion algorithms. In future, we hope to make even better approximation of graphic entities so that average recognition accuracy can be made higher. The research by Bhowmick and Bhattacharya [82] can be of some significance in this connection. Some other issues are to be addressed in future. One of them is addition of editing and scaling capability. Labeling of points on a diagram has been deliberately ignored in the present study and hence, will be taken up in future. We plan to integrate our text-to-diagram conversion module with other existing Braille graphics software tools (like QikTac, IVEO Viewer, etc.) and generate tactile graphics using high-end image embossers. The quality of the output from this system can be compared with the result obtained using Braille text printer. Moreover, we can make further survey with a control group of users to compare the learning outcome in two systems.

Though geometry problems are taken as a case study, this work can be extended to address similar problems in other subject areas (e.g. physics, engineering drawings, etc.). Following the same approach as described here, simple electrical or electronic circuits, building diagrams, free body diagrams, etc. can be generated (using Braille text printer) from the problem texts that describe such circuits or diagrams. However, this would require development of suitable knowledge bases containing descriptions of diagram-elements specific to a subject.

CHAPTER 7 Conclusion

7.1 Review

In this section an attempt is made to review the research work done so far and also provide a qualitative measure of achievement against the initial goals of this research.

In the year 2006-2007, it all started with the very idea - can machine be made intelligent enough to read a text and draw the underlying figure automatically? It would be great if the answer is yes. Because then the classical approach for explaining a scientific text or presenting the solution of a word problem through illustrative figures, which forms the basis of teaching-learning science subjects, can be fully automated. It was known that machine can draw figures if step-wise graphics commands are given. It was also known that machine can read and analyze English text with the help of *parsers*. But how machine can understand the entities and their physical relations described in a scientific text and self-generate graphics commands to illustrate such relations with suitable figure seemed to be an unaddressed problem. A comprehensive review of the relevant literature established this fact. Though some work was found that attempted to provide mathematical solution of a limited set of scientific word problems (stated in English), no framework was found that could clearly depict the configurations of the problem-entities (typically those found in problems of mechanics, geometry, electrical engg., etc.). Therefore the initial goal of our research was set as: to establish an implementable framework of text-to-diagram (t2d) conversion of scientific descriptions stated in English. As a focused case study the word problems of school level geometry were chosen.

While addressing the research problem, the natural human approach of diagram drawing from a scientific text was closely analyzed and the necessary basic skills were identified as: knowledge about the subject, ability to understand language (English) and ability to draw. So we decided to propose working models of a (geometry) knowledge base and a language *parser* which could be used to form a language-free formal representation of the input text for machine-drawing of diagram with standard graphics functions.

Our first attempt was to develop a geometry knowledge base (which we later named as *GeometryNet*) containing meaning of different geometric terms. For this, 300 geometry problems of school level (grade VIII, IX, and X) geometry books were analyzed to find the optimum number of entities and relations to describe a geometric term or concept. The style of knowledge representation in *GeometryNet* is grossly adopted from WordNet maintaining the concept of inheritance to minimize data redundancy. With regards to importance of *GeometryNet* in the proposed t2d conversion system, firstly, it is used by a *parser* for semantic analysis of the input geometric statements; secondly, it is used for machine-understanding and formation of a draw-able representation of the input statements. To the best of our knowledge no such knowledge base exists in geometry domain. However, the *GeometryNet* in its present form is only a prototype and may not be comprehensive enough to be used as part of a fully functional t2d software. But it can easily be extended to accommodate more geometric definitions and relations because of its simple design.

The next challenge of our research was to develop a language *parser* that can produce a language-free formal representation of the input geometric description (stated in English). This was necessary to resolve the problem of ambiguity often carried by any natural language text as a given fact can be stated in many different ways. After lot of experiments with the existing schemes of representing information (extracted from input statements), the *parse graph* approach was found effective. We tried to avoid rule-based processing as much as possible, but owing to the diversity of cases generalization was difficult. Though we did not apply any standard parsing approach, the functionality of the *parser* and *translator* seems to be quite fitting for the present case. There are failure cases too for which the NLP module needs to be fine tuned. Our NLP approach may not be the best possible approach, but it is definitely an effective one as the success of the entire t2d conversion system is attributed mostly to the correct interpretation and representation of the input text.

Even though the NLP module effectively represented a problem free from linguistic elements, the output was not sufficient to be converted into diagram. Another *parser* was required to find the geometric meaning (defining variables and equations) of different terms upon consulting the *GeometryNet*. This task was relatively straight forward. The next task of instantiating the generic variables was rather critical; several pass of the solver routine was sometimes necessary to find numerical values of the variables from the constraint equations. Finally, the diagram drawing part was accomplished using standard graphics functions upon passing numerical data as arguments to these functions. The fact that out of a test set of 40 school-level geometry problems, correct diagrams are generated for 32 problems (for which the NLP module produced correct representation), itself speaks of the achievement of our diagram drawing module.

Reporting of a newly developed system is never complete without an evaluation, i.e. a measure of performance/accuracy of the system. Two methods of evaluating the t2d conversions are presented. The first method was adopted from an existing mechanism of performance evaluation of Graphic Recognition Systems. A better method was felt necessary to overcome the limitation of this method in not being able to measure errors of diagrams drawn without any positional constraint. Finally, a second evaluation method is proposed which is unique in its approach of measuring conceptual correctness of any geometry diagram. In fact, the problem of evaluating a machine-drawn diagram (from text) was never addressed before.

While carrying out research on t2d conversions, we thought about a similar system for the blind persons. A visit to couple of Blind schools in and around Kolkata revealed the poor scenario of science education for the blind students. To make the t2d conversions accessible and affordable by these students, we decided to find ways of producing diagrams using only Braille text printer - it called for research in a new direction (computer graphics and Braille system). Finally, we were successful in mapping digital lines and circles to the Braille character grid which in turn yielded tactile representation of geometry diagrams with Braille text. We did not find any such (automatic) tactile drawing utility in the literature. That the utility is useful and meaningful for the blind students is reflected from the field tests and user's feedback. Had there been no infrastructural constraints in the field tests, experiments with larger sets of test subjects and geometry problems would have revealed more general observations about our system. However, there is enough scope for improvement in the Braille conversion algorithms as is evident from the errors measured for the Braille diagrams. Though this particular work was not part of our initial research goal, it has given us maximum satisfaction considering its potential societal impact.

7.2 Scope of Future Research

Though the framework proposed for t2d conversion is likely to work for various domains, the scope of the present research has been limited only to the domain of school level geometry problems. It does not deal with complicated (higher level) geometry such as menstruation, 3D geometry, solid geometry and geometric calculus. The *GeometryNet* can be extended to incorporate related terms. Again going beyond geometry the research can be extended to address similar diagram-based problems in other subject domains e.g. physics, mechanics or civil, mechanical, electrical, electronics engineering, etc. if suitable domain-specific knowledgebase similar to *GeometryNet* can be developed. Then following the same approach free body diagrams, electrical or electronic circuits, building diagrams, etc. can be drawn by the machine upon feeding the problem texts that describe such circuits or diagrams.

The rule based and graph based approach employed by the NLP module produces errors for some typical cases as described in Section 4.5.3 of Chapter 4. A standard parsing approach for *intermediate representation* may be tried that can further reduce the errors of NLP stage. Similarly, object-oriented approaches for developing domain-specific ontology may also be adopted to test if it is more effective as a knowledge base for t2d conversions.

The *intermediate representation* can be used as input of a solver for actual solution of a word problem. It can also be used for machine translation of diagram describing word problems or text from English into other languages.

In the evaluation of diagrams converted from text, Method II is better than Method I but the former cannot properly evaluate figures containing circles or diagrams whose adjacency matrix do not match with that of the groundtruth diagrams. It can be explored whether *relational graphs* (or other techniques) can be of any use in this context. It would be nice if the machine-drawn diagram can be evaluated with reference to the input word problem itself without necessitating drawing of groundtruth diagrams separately.

In future even better Braille approximation of graphic entities must be tried for so that users do not confuse inner details with outlines of a diagram and overall recognition accuracy is increased. Some other issues of Braille based drawing that may be addressed in future are: labeling of diagrams, diagram editing and scaling capability, low-cost touch-based voice guidance utility that would provide voice labels to different parts of a diagram, and better option for step-by-step drawing. It may be explored how our Braille mapping module can be made to work with high-end image embossers so as to generate high resolution tactile graphics automatically from textual description.

BIBLIOGRAPHY

[1] A Bundy, L Byrd, G Luger, C Mellish, and M Palmer, *Solving Mechanics Problems Using Meta-Level Inference*, In Proc. of IJCAI-79, Tokyo, Japan, 1017-1027, 1979.

[2] A Crossan and S Brewster, *Multimodal Trajectory Playback for Teaching Shape Information and Trajectories to Visually Impaired Computer Users*, ACM Transactions on Accessible Computing (TACCESS), 1, 2008.

[3] A Dasgupta, A Mukherjee, and S S Barat, *Representation of Triangular Shaped Geometric Figures in Braille*, In Proc. of International Conference on Emerging Applications of Information Technology, Kolkata, India: Elsevier, 2006.

[4] A Lahiri, S J Chattopadhyay, and A Basu, *Sparsha: A Comprehensive Indian Language Toolset for the Blind*, In Proc. of 7th International ACM SIGACCESS Conference on Computers and Accessibility, USA, 2005.

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

[6] A Mukherjee and U Garain, *Intelligent Tutoring of School Level Geometry Using Automatic Text to Diagram Conversion Utility*, In Proc. of 2nd International Conference on Emerging Applications of Information Technology (EAIT), Kolkata, India: IEEE, 45-48, (the proceedings available in IEEE Xplorer), 2011.

[7] A Mukherjee and U Garain, *Understanding of Natural Language Text for Diagram Drawing*, In Proc. of IASTED International Conference on Artificial Intelligence and Soft Computing, Palma De Mallorca, Spain, 138-145, 2009.

[8] A Mukherjee, S Sengupta, D Chakraborty, A Sen, and U Garain, *Text to Diagram Conversion: a Method for Formal Representation of Natural Language Geometry Problems*, In Proc. of IASTED International Conference on Artificial Intelligence and Applications, Austria, 137-144, 2013.

[9] A Mukherjee, U Garain, and A Biswas, *Diagram Drawing Using Braille Text: A Low Cost Learning Aid for Blind People*, In Global Trends in Intelligent Computing Research and Development, Chapter 14, 384-406, Eds. B.K. Tripathy and D.P. Acharjya, Publisher: IGI Global, USA, 2013.

[10] A Mukherjee, U Garain, and A Biswas, *Evaluation of the Graphical Representation for Text-to-Graphic Conversion Systems*, In 10th IAPR International Workshop on Graphics Recognition (GREC 2013), Bethlehem PA, USA, 2013.

[11] A Mukherjee, U Garain, and M Nasipuri, *On Construction of a GeometryNet*, In Proc. of IASTED International Conference on Artificial Intelligence and Applications, Innsbruck, Austria, 530-536, 2007.

[12] A. Mukherjee, U. Garain and A. Biswas, *Experimenting with Automatic Text-to-Diagram Conversion: A Novel Teaching Aid for the Blind People*, In Journal of Educational Technology & Society (ETS), (In press).

[13] A Dutta, J Gibert, J Llads, H Bunke, and U Pal, *Combination of Product Graph and Random Walk Kernel for Symbol Spotting in Graphical Documents*, In Proc. of 21st International Conference on Pattern Recognition (ICPR 2012), Tsukuba, Japan, 1663-1666, 2012.

[14] APH-Shopping, American Printing House for the Blind, Inc., http://shop.aph.org/webapp/wcs/stores/servlet/Home_10001_11051 (accessed June 30, 2013).

[15] B M Krebs, *ABCs of Braille*, Braille Institute of America, 1977.

[16] Braille Geometry Kit, http://www,mathwindow,com/brl_geometry.html (accessed June 30, 2013).

[17] Braille Graphics and Tactile Graphics from Repro-Tronics, http://www.brailler.com/repro.htm (accessed June 30, 2013).

[18] C Baral, M Gonzalez, J Dzifcak, and J Zhou, *Using Inverse Lambda and Generalization to Translate English to Formal Languages*, In Proc. of International Conference on Computational Semantics, Oxford, 2011.

[19] C Fellbaum, D Gross, and K Miller, *Adjectives in WordNet*, International Journal of Lexicography, 3, 265-277, 1990.

[20] C Fellbaum, *English Verbs as a Semantic Net*, International Journal of Lexicography, 4, 278-301, 1990.

[21] C Gouy-Pailler, S Zijp-Rouzier, S Vidal, and D Chłne, *A Haptic Based Interface to Ease Visually Impaired Pupils Inclusion in Geometry Lessons*, In Proc. of 4th International Conference on Universal Access in Human-Computer Interaction, 598-606, 2007.

[22] C K Looi and B T Tan, *A Cognitive-Apprenticeship-Based Environment for Learning Word Problem Solving*, Journal for Research in Mathematical Education, 17(4), 339-354, 1998.

[23] C R Fletcher, *Understanding and Solving Arithmetic Word Problems: A Computer Simulation*, Behavior Research Methods, Instruments, and Computers, 17, 565-571, 1985.

[24] C Winograd, *Understanding Natural Language*, Academic Press, 1972.

[25] D Dellarosa, *A Computer Simulation of Children's Arithmetic Word Problem Solving*, Behavior Research Methods, Instruments, and Computers, 18, 147-154, 1986.

[26] D F Rogers, *Procedural Elements for Computer Graphics*, USA: McGraw Hill Book Co, 1985.

[27] D G Bobrow, *Natural Language Input for a Computer Problem Solving System*, Project MAC, M.I.T, Cambridge, 1964.

[28] D L Briars and J H Larkin, *An Integrated Model of Skill in Solving Elementary Word Problems*, Cognition and Instruction, 1, 245-296, 1984.

[29] D Marples, *Argument and Technique in the Solution of Problems in Mechanics and Electricity*, Technical Report CUED/C-Educ/TRI, Dept. of Engineering, Cambridge, England, 1974.

[30] D McDermott and G J Sussman, *The Conniver Reference Manual*, A1M-259a, Artificial Intelligence Laboratory, MIT, Cambridge, 1974,

[31] D Muth, *Extraneous Information and Extra Steps in Arithmetic Word Problems*, Contemporary Educational Psychology, 17, 278-285,1992.

[32] D Parkes, *Nomad: Enabling Access to Graphics and Text-based Information for Blind and Visually Impaired and Other Disability Groups*, In Proc. of World Congress Tech. People Disability, Arlington, Virginia, 689-716, 1991.

[33] Duxbury Systems Products, http://www.duxburysystems.com/products.asp (accessed June 30, 2013).

[34] E Charniak, *CARPS, A Program Which Solves Calculus Word Problems*, Report MAC-TR-51, Project MAC, M.I.T, Cambridge, Mass, 1968.

[35] E Charniak, *Computer Solution of Calculus Word Problems*, In Proc. of International Joint Conference on Artificial Intelligence, Washington, DC, 303-316, 1969.

[36] E, Brill, *A Simple Rule-based Part of Speech Tagger*, In Proc. of the Workshop on Speech and Natural Language: HLT91, Morristown, NJ, USA, 112-116, 1992.

[37] F Vidal-Verdu and M Hafez, *Graphical Tactile Displays for Visually Impaired People*, IEEE Transactions on Neural Systems and Rehabilitation Engineering, 15, 119-130, 2007.

[38] G A Miller, R Beckwith, C Fellbaum, D Gross, and K Miller, *Introduction to WordNet: An On-line Lexical Database*, International Journal of Lexicography, 3(4), 235-244, 1990.

[39] G A Miller, *Nouns in WordNet: a Lexical Inheritance System*, International Journal of Lexicography, 3(4), 245-264, 1990.

[40] G E Oberem, P S Shaffer, and L C McDermott, *Using a Computer to Investigate and Address Student Difficulties in Drawing Free-Body Diagrams*, In Proc. of Summer Meeting of the American Association of Physics Teachers, Boise, ID, 1993.

[41] G E Oberem, *ALBERT: A Physics Problem Solving Monitor and Coach*, In Proc. of First International Conference on Computer Assisted Learning (ICCAL'87), Calgary Alberta, Canada, 179-184, 1987.

[42] G E Oberem, *Transfer of a Natural Language System for Problem-Solving in Physics to Other Domains*, In Proc. of World Conference on Educational Multimedia and Hypermedia:ED-MEDIA 94, Vancouver, British Columbia, Canada, 424-431, 1994.

[43] G Luger, *Mathematical Model Building in the Solution of Mechanics Problems: Human Protocols and the Mecho Trace*, Cogn. Sci., 5(1) 55-77, 1981.

[44] G S Novak and W C Bulko, *Diagrams and Text as Computer Input*, Journal for Visual Languages and Computing, 4(2), 161 - 175, 1993.

[45] G S Novak, *Computer Understanding of Physics Problems Stated in Natural Language*, Ph,D, Thesis, arch 61, The University of Texas at Austin, 1976.

[46] H Minagawa and N Ohnishi, *Tactile-Audio Diagram for Blind Persons*, IEEE Trans. Rehabilitation Engineering, 4(4), 431- 437, 1996.

[47] H, Bunke, *Recent Developments in Graph Matching*, In Proc. of ICPR 2000, 2117-2124, 2000.

[48] I T Phillips and Atul K Chhabra, *Empirical Performance Evaluation of Graphics Recognition Systems*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 21(9), 849-869, 1999.

[49] J A Borges and L R Jansen, *Blind People and the Computer: An Interaction that Explores Drawing Potentials*, In Proc. of SEMENGE'99 Seminrio de Engenharia, Universidade Federal Fluminense, Brazil, 1999.

[50] J A Gardner, V Bulatov, and H Stowell, *The ViewPlus IVEO Technology for Universally Usable Graphical Information*, In Proc. of International Conference on Technology and People with Disabilities: CSUN, Los Angeles, 2005.

[51] J A Gardner, V Bulatov, M Suzuki, and K Yamaguchi, *Audio/Visual/Tactual Presentation of Scientific Graphics*, In Proc. of Joint Conference ASCM 2009 and MACIS 2009, Japan, 242-251, 2009.

[52] J De Kleer, *Multiples Representation of Knowledge in a Mechanics Problem-Solver*, In Proc. of IJCAI-77, Cambridge, Massachusetts, 299-304, 1975.

[53] J De Kleer, *Qualitative and Quantitative Knowledge in Classical Mechanics*, Artificial Intelligence Laboratory, M.I.T, 1975.

[54] J L Toennies, J Burgner, T J Withrow, and R J Webster, *Toward Haptic/Aural Touchscreen Display of Graphical Mathematics for the Education of Blind Students*, In Proc. of World Haptics Conference (WHC), IEEE, 373-378, 2011.

[55] J L Wheeler and J W Regian, *The Use of A Cognitive Tutoring System in the Improvement of the Abstract Reasoning Component of Word Problem Solving*, Computer in Human Behavior, 15, 243-254, 1999.

[56] J P Gelb, *The Computer Solution of English Probability Problems*, PhD Thesis, Computer Science Program, RPI, Troy, New York, 1971.

[57] J P Gelb, Second International Conference on the Learning Sciences: IJCAI-71, London, England, 455-462, 1971.

[58] J W Raymond and P Willet, *Maximum Common Subgraph Isomorphism Algorithms for the Matching of Chemical Structures*, Journal of Computer-Aided Molecular Design, 16, 521-533, 2002.

[59] K D Forbus, C K Riesbeck, L Birnbaumi, K Livingston, A Sharma, and L Ureel, *Integrating Natural Language, Knowledge Representation and Reasoning,and Analogical Processing to Learn by Reading*, In Proc. of 22nd Conference on Artificial Intelligence (AAAI-07), Vancouver, 2007.

[60] K E Chang, Y T So, and H F Lin, *Computer-Assisted Learning for Mathematical Problem Solving*, Computers & Education, 2005.

[61] K R Koedinger and E L F Sueker, *PAT Goes to College: Evaluating a Cognitive Tutor for Developmental Mathematics*, In Proc. of Second International Conference on the Learning Sciences, 180-187, 1996.

[62] K R Koedinger, J R Anderson, W H Hadley, and M A Mark, *Intelligent Tutoring Goes to School in the Big City*, In Proc. of International Journal of Artificial Intelligence in Education, 8, 30-43, 1997.

[63] K Rassmus-Grohn, C Magnusson, and H Eftring, *User Evaluations of a Virtual Haptic-Audio Line Drawing Prototype*, Haptics and Audio Interaction Design, 4129, 81-91, 2006.

[64] L Bussel, *Touch Tiles: Elementary Geometry Software with a Haptic and Auditory Interface for Visually Impaired Children*, In Proc. of EuroHaptics, 512515, 2003.

[65] L Casely-Hayford and P Lynch, *A Review of Good Practice in ICT and Special Educational Needs for Africa*, www.comp.dit.ie/dgordon/Publications/Contributor/Ghana/SENPHASE1FINAL.pdf (accessed June 30, 2013).

[66] L Sterling, A Bundy, L Byrd, R OKeefe, and B Silver, *Solving Symbolic Equations with Press*, Edited by L Sterling, A Bundy, L Byrd, R OKeefe, and B Silver, ComputerAlgebra, Lecture Notes in Computer Science,Springer, 144, 109-116, 1982.

[67] L Wenyin and D Dori, *A Protocol for Performance Evaluation of Line Detection Algorithms*, Machine Vision and Applications, 9(5/6), 240-250, 1997.

[68] M B Dias, M K Rahman, S Sanghvi, and K Toyam, *Experiences with Lower-Cost Access to Tactile Graphics in India*, In Proc. of First ACM Symposium on Computing for Development, USA, 2010.

[69] M Batusic and F Urban, *Preparing Tactile Graphics for Traditional braille Printers with BrlGraphEditor*, Computers Helping People with Special Needs, Lecture Notes in Computer Science (Springer), 2398, 261-274, 2002.

[70] M Horstmann et al., *Automated Interpretation and Accessible Presentation of Technical Diagrams for Blind People*, The New Review of Hypermedia and Multimedia, 10(2), 141-163, 2004.

[71] M Ilieva and H Boley, *Representing Textual Requirements as Graphical Natural Language for UML Diagram Generation*, In Proc. of Twentieth International Conference on Software Engineering & Knowledge Engineering (SEKE'2008), San Francisco, USA, 478-483, 2008.

[72] M K Hesham and A L James, *The Integrated Communication 2 Draw (IC2D): A Drawing Program for the Visually Impaired*, CHI '99: Extended Abstracts on Human Factors in Computing Systems, Pittsburgh, Pennsylvania, 222-223, 1999.

[73] M Kurze, *TDraw: A Computer-based Tactile Drawing Tool for Blind People*, In Proc. of Second Annual ACM Conference on Assistive Technologies, Vancouver, British Columbia, Canada, 131-138, 1996.

[74] M L Williams, R C Wilson, and E R Hancock, *Deterministic Search for Relational Graph Matching*, Pattern Recognition, 32(7), 1255-1271, 1999.

[75] M Minsky, *A Framework for Representation of Knowledge*, AIM-306, Artificial Intelligence Laboratory, M.I.T, Cambridge, 1973.

[76] M Steele and J Steele, *DISCOVER: An Intelligent Tutoring System for Teaching Students with Learning Difficulties to Solve Word Problems*, Journal of Computers in Mathematics and Science Teaching, 18(4), 351-359, 1999.

[77] N 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 Proc. of International Conference on Artificial Intelligence and Education (ICAIE), 324 - 330, 2010.

[78] Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay, *Learning to Automatically Solve Algebra Word Problems*, In Proc. of the Conference of the Association for Computational Linguistics (ACL),2014.

[79] O Lahav and D Mioduser, *Haptic-Feedback Support for Cognitive Mapping of Unknown Spaces by People Who are Blind*, International Journal of Human-Computer Studies, 66, 2335, 2008.

[80] O Hori and D Doermann, *Quantitative Measurement of the Performance of Raster-to-Vector Conversion Algorithms*, In Proc. of 1st Workshop on Graphics Recognition: Methods and Applications- Selected Papers, R Kasturi and K Tombre, eds., Lecture Notes in Computer Science (Springer-Verlag LNCS), 1072, 57-68, 1996.

[81] P Albert, *Math Class: an Application for Dynamic Tactile Graphics*, In Proc. of Computers Helping People with Special Needs, Lecture Notes in Computer Science, Springer, 4061(2006), 1118-1121, 2006.

[82] P Bhowmick and B B Bhattacharya, *Fast Polygonal Approximation of Digital Curves Using Relaxed Straightness Properties*, IEEE Trans. Pattern Anal. Mach. Intell., 29(9), 1590-1602.

[83] P Roth, L Petrucci, and T Pun, *From Dots to Shape : An Auditory Haptic Game Platform for Teaching Geometry to Blind Pupils*, In Proc. of ICCHP 2000, Germany, 603-610, 2000.

[84] PictureBraille, http://www.pentronics.com.au/index_files/Picturebraille.htm (accessed June 30, 2013).

[85] R Beckwith, G A Miller, and R Tengi, *Design and Implementation of the WordNet Lexical Database and Searching Software*, Princeton University, 1993.

[86] R Davis and B G Buchanan, *Meta-level Knowledge: Overview and Applications*, In Proc. of IJCAI77, 920-927, 1977.

[87] R E Ladner et al., *Automating Tactile Graphics Translation*, In Proc. of 7th International ACM SIGACCESS Conference on Computers and Accessibility, Baltimore, 2005.

[88] R F Simmons, *Natural Language Question-Answering Systems: 1969*, Communications of the ACM, 13(1), 15-30, 1970.

[89] R F Simmons, *Semantic Networks: Their Computation and Use in Understanding English Sentences*, Computer Models of Thought and Language, edited by Schank and Colby, Freeman and Co., 1973.

[90] R K Lindsay, *Inferential Memory as the Basis of Machines Which Understand Natural Language*, Computers and Thought, New York: McCraw-Hill, 1963.

[91] S E Krufka and K E Barner, *Automatic Production of Tactile Graphics from Scalable Vector Graphics*, In Proc. of 7th International ACM SIGACCESS Conference on Computers and Accessibility, 2005.

[92] S K Guha and S Anand, *Computer as a Group Teaching Aid for Persons Who are Blind*, Journal of Rehabilitation Research and Development, 29(3), 57-63, 1992.

[93] S Landau and K Gourgey, *A New Approach to Interactive Audio/Tactile Computing: The Talking Tactile Tablet*, In Proc. of Technology and Persons with Disabilities Conference, California State Univ, Northridge, USA, 2003.

[94] S P Marshall, *Schemas in Problem Solving*, M.I.T, Cambridge: Cambridge University Press, 1995.

[95] S Ramani, *A Language Based Problem Solver*, In Proc. of IJCAI-71, London, England, 463-473, 1979.

[96] S Ramani, *Language Based Problem-Solving*, Ph,D, Thesis, Computer Group, Tata Institute of Fundamental Research, 1969.

[97] S Rouzier, B Hennion, T Perez-Segovia, and D Chłne, *Touching Geometry for Visually Impaired Pupils*, In Proc. of EuroHaptics, Germany, 104-109, 2004.

[98] S Sharmin, G Evreinov, and R Raisamo, *Non-Visual Feedback Cues or Pen Computing*, In Proc. of World Haptics, 625-628, 2005.

[99] S Zijp-Rouzier and E Petit, *Teaching Geometry to Visually Impaired People Using Haptics and Sound*, Human-Computer Interaction, 2005.

[100] T Barbieri, L Mosca, and L Sbattella, *Learning Math for Visually Impaired Users*, In Proc. of Computers Helping People with Special Needs (ICCHP 08), Lecture Notes in Computer Science, Springer, 5105(2008), 907-914, 2006.

[101] T P Way and K E Barner, *Automatic Visual to Tactile Translation, Part II: Evaluation of the Tactile Image Creation System*, IEEE Transactions on Rehabilitation Engineering, 5, 1997.

[102] T Watanabe, M Kobayashi, S Ono, and K Yokoyama, *Practical Use of Interactive Tactile Graphic Display System at a School for the Blind*, Current Developments in Technology-Assisted Education, 1111-1115, 2006.

[103] V K Chaudhri, B Bredeweg, R Fikes, S McIlraith, and M P Wellman, *A Categorization of KRR Methods for Requirement Analysis of a Query Answering Knowledge Base*, In Proc. of Sixth International Conference on Formal Ontology in Information Systems (FOIS 2010), 158-171, 2010.

[104] ViewPlus Tiger Software Suite, http://www.viewplus.com/products/ software/braille-translator/ (accessed June 30, 2013).

[105] W A Woods, *Transition Network Grammare for Natural Language Analysis*, In Proc. of CACM, 1970.

[106] W K 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:Elsevier, 48(4), 582-601, 2007.

[107] W L Hsu, S H Wu, and Y S Chen, *Event Identification Based on the Information Map INFOMAP*, In Proc. of Systems, Man, and Cybernetics Conference, Tuscon, Arizona, USA: IEEE, 1661-1672, 2001.

[108] W Lu, H T Ng, W S Lee, and L Zettlemoyer, *A Generative Model for Parsing Natural Language to Meaning Representations*, In Proc. of EMNLP-08, 2008.

[109] WordNet: A Lexical Database for the English Language, Cognitive Science Laboratory, Princeton University, http://wordnet.princeton.edu/

[110] Y Bakman, *Robust Understanding of Word Problems with Extraneous Information*, http://aps.arxiv.org/abs/math.GM/0701393 (accessed June 30, 2013).

List of Publications

Journal:

1. **A Mukherjee** and U Garain, *A Review of the Methods for Automatic Understanding of Natural Language Mathematical Problems*, Artificial Intelligence Review, **29**(2), 93-122, 2008.

2. **A Mukherjee**, U Garain, and A Biswas, *Experimenting with Automatic Text-to-Diagram Conversion: A Novel Teaching Aid for the Blind People*, Journal of Educational Technology & Society, (In press).

Conference Proceedings:

1. **A Mukherjee**, U Garain, and M Nasipuri, *On Construction of a GeometryNet*, In Proc. of IASTED International Conference on Artificial Intelligence and Applications, Innsbruck, Austria, 530-536, 2007.

2. **A. Mukherjee** and U. Garain, *Automatic Diagram Drawing based on Natural Language Text*, In Proc. of 5th International Conference (Diagrams 2008), Herrsching, Germany, 398-400, 2008.

3. **A Mukherjee** and U Garain, *Understanding of Natural Language Text for Diagram Drawing*, In Proc. of IASTED International Conference on Artificial Intelligence and Soft Computing, Palma De Mallorca, Spain, 138-145, 2009.

4. **A Mukherjee** and U Garain, *Intelligent Tutoring of School Level Geometry Using Automatic Text to Diagram Conversion Utility*, In Proc. of 2nd International Conference on Emerging Applications of Information Technology (EAIT), Kolkata, India: IEEE, 45-48, (the proceedings available in IEEE Xplorer), 2011.

5. **A Mukherjee**, S Sengupta, D Chakraborty, A Sen, and U Garain, *Text to Diagram Conversion: a Method for Formal Representation of Natural Language Geometry Problems*, In Proc. of IASTED International Conference on Artificial Intelligence and Applications, Austria, 137-144, 2013.

6. **A Mukherjee**, U Garain, and A Biswas, *Evaluation of the Graphical Representation for Text-to-Graphic Conversion Systems*, In 10th IAPR International Workshop on Graphics Recognition (GREC 2013), Bethlehem PA, USA, 2013.

Post-Proceedings:

A Mukherjee, U Garain, and A Biswas, *Evaluation of Diagrams Produced by Text-to-Graphic Conversion Systems*, In GREC 2013 - Graphics Recognition: Current Trends and Challenges; Selected papers from the Tenth IAPR International Workshop on Graphics Recognition, vol. 8746, Springer LNCS, 2014.

Book Chapter:

A Mukherjee, U Garain, and A Biswas, *Diagram Drawing Using Braille Text: A Low Cost Learning Aid for Blind People*, In Global Trends in Intelligent Computing Research and Development, Chapter 14, 384-406, Eds. B.K. Tripathy and D.P. Acharjya, Publisher: IGI Global, USA, 2013.