

Ain Shams University  
Faculty of Computers and Information Sciences  
Computer Science Department

**Mind Map Generator**  
2nd Edition

**Done By***:* **Department***:*

Fatima Mohammad Abdullah (CS)  
Marwa Soliman Mohamed (IS)  
May Mahmoud Samy Shalaby (CS)  
Mona Othman Abdel Aziz (CS)  
Shahenda Magdy Nasr El Din (IS)

**Supervised By:**

Professor Dr. Taymour Nazmy

*Vice dean of higher studies and researches and Professor in Computer Science department, Faculty of Computer and Information Sciences, Ain Shams University*

Dr. Mohammed Abdeen

*Computer Science department, Faculty of Computer and Information Sciences, Ain Shams University*

**Assistant:**Mohammad Hamdy

Acknowledgement

We would like to acknowledge and extend our heartfelt gratitude to the following persons who have made the completion of this project possible:   
  
Our supervisors, Prof. Dr. Taymour Nazmy and Dr. Mohamed Abdeen, for their vital support and advice.

Our head of department, Prof. Dr. Mostafa Aref, for his encouragement and guidance during our seminars.  
  
Mohammad Hamdy, for his understanding and assistance, and for the constant reminders and much needed motivation.

Ahmed Salah for guiding us to this project.

Team members of the original Mind Map Generator created in 2007/2008, Asmaa Hamdy, Radwa Abdl-Hamid and Sara Samir, for coming up with the idea and proving its feasibility.

All the faculty members and Staff.

Most especially to our families and friends for their understanding and moral support, and for enduring with us the hard times, the late hours and cranky moods.  
  
And to God, who made all things possible.

Abstract

Mind mapping is a revolutionary system for capturing ideas and insights horizontally on a sheet of paper. It is a creativity- and productivity-enhancing technique that can improve the learning and efficiency of individuals. It helps people "see" a problem and its solution.

It represents associated thoughts with symbols rather than with unrelated words. The mind forms associations almost instantaneously, and "mapping" allows you to write your ideas quicker than expressing them using only words or phrases. It is a way of note taking and planning that mirrors the design of the human brain. This diagram is used to represent words, ideas, tasks, or other items linked to and arranged radially around a central key word or idea. Mind maps use mind mapping.

Mind maps can be used in nearly every activity where thought, planning, recall or creativity are involved. This means they can be used to generate, visualize, structure, and classify ideas, and as an aid in study, organization, problem solving, decision making, and writing. Now, it’s used by millions of people around the world - from the very young to the very old - whenever they wish to use their minds more effectively.

But, making mind maps manually requires reading and understanding the text well which often takes much time and effort. In addition, not all people are creative enough to draw mind maps. There are softwares that deal with mind maps but needs the user to having already read and understood whatever it is they want to create a mind map for, and design its layout, which is also not very helpful. This means that if mind maps can be generated automatically, this will save a lot of time and effort, especially to those new to the technique. That is what led to the creation of the “Mind Map Generator” in 2007/2008.

The “Mind Map Generator” is a software which takes plaintext as input and generates a mind map for that plaintext as output. But still, it had some limitations which were part of our objective with continuing on with this project. Another of our objectives is dividing the input plaintext into a multi leveled mind map, so that related topics or information appear together in a separate mind map. This process is done in a way so that the mind maps generated are automatically laid out on the screen to facilitate reading and understanding.

List of Figures

2.1 Example of Mind Map ………………………………………………………………….……………………17

3.1 System Architecture……………………………………………………………………….…………………23

3.2 Mind Map Generator Input Output…..…………………………………………….……………….24

3.3 Phases of Frontend……………………….…..…………………………………………….………………..27

3.4 Text Meaning Representation Input Output…………………………………….……………….28

3.5 Multi-level Generation Input Output..……………………………………………….……………..29

3.6 Mind Map Conversion Input Output …………………………………………………….…………..29

3.7 Automatic layout Input Output…………………………………………………………….…………..30

4.1 Frontend phases…………….…………………………………………………………………….…………..31

4.2 Tree Representation...………………………………………………………………………….…………..34

4.3 Chart...…………………………………………………………………………………………………….………..39

4.4 Arc Extension Algorithm……………………………………………………………………………........39

4.5 Bottom-up chart parsing algorithm…..................................................................39

4.6 Two parse trees for two different sentences……………………………………………………42

4.7 The modified parse tree…………………………………………………………………………………….43

4.8 WordNet troponyms…………………………………………………………………………………………47

6.1 Demonstration of why multi-level should be used…………………………………….........69

6.2 Multi-level phases………………………………………………………………………………………………70

6.3 Example of an ontology………………………………………………………………………………………73

9.1 Input form with sample text……………………………………………………………………………….90

9.2 TMR screenshot 1………………………………………………………………………………………………..91

9.3 TMR screenshot 2………………………………………………………………………………………………..91

9.4 TMR level 1 with "virus" as central topic………………………………………………………………92

9.5 TMR level 2 with "swine influenza" as central topic……………………………………………..93

9.6 TMR level 3 with "similar symptoms" as central topic………………………………………….93

9.7 TMR level 4 showing the symptoms…………………………………………………………………………94

9.8 The first mind map produced by the input text ………………………………………………………..95

9.9 The mind map produced by clicking "spread"…………………………………………………………..95

9.10 The mind map produced by clicking "Not Transmit"……………………………………………….96

9.11 The mind map produced by clicking "Swine Influenza"…………………………………………..96

9.12 The mind map produced by clicking "similar symptoms"………………………………………..97

9.13 The mind map produced by clicking "Animal symptoms"…………………………………………97

9.14 Graph showing the rate of increase of time against the number of words…………………99

Contents

**Chapter (1) Introduction**

[1.1. Motivation 9](#_Toc233190512)

[1.2. Project Definition 9](#_Toc233190513)

[1.3. Objective 9](#_Toc233190514)

[1.4. Documentation Organization and Contents 10](#_Toc233190515)

**Chapter (2) Background**

[2.1. Mind Maps and Mind Mapping 11](#_Toc233190516)

[2.1.1. Definition 11](#_Toc233190517)

[2.1.2. The History of Mind Maps ‎34 12](#_Toc233190518)

[2.1.3. Types of Mind Maps‎1 13](#_Toc233190519)

[2.1.4. Advantages and Uses of Mind Maps 15](#_Toc233190520)

[2.2. Natural Language Processing 18](#_Toc233190521)

[2.2.1. The meaning of Natural Language Processing ‎36 18](#_Toc233190522)

[2.2.2. NLP Sub-problems ‎36 19](#_Toc233190523)

[2.2.3. Major tasks in Natural Language Processing ‎36 20](#_Toc233190524)

[2.2.4. equired Knowledge ‎36 21](#_Toc233190525)

**Chapter (3) System Architecture**

[3.1. Mind Map Generator Frontend (Natural Language Processing) 24](#_Toc233190526)

[3.1.1. Morphological Analysis 24](#_Toc233190527)

[3.1.2. Parsing 24](#_Toc233190528)

[3.1.3. Syntax Analysis 25](#_Toc233190529)

[3.1.4. Discourse Analysis 25](#_Toc233190530)

[3.1.5. Word Sense Disambiguation 25](#_Toc233190531)

[3.2. Text Meaning Representation 27](#_Toc233190532)

[3.3. Multi-level Generation 28](#_Toc233190533)

[3.4. Mind Map Conversion 28](#_Toc233190534)

[3.5. Automatic Layout Allocation 29](#_Toc233190535)

**Chapter (4) Mind Map Generator frontend**

[4.1. Morphological Analysis 30](#_Toc233190536)

[4.1.1. Morphology 30](#_Toc233190537)

[4.1.2. Morphological Parsing 31](#_Toc233190538)

[4.1.3. WordNet 2.1 Morphological Analyzer 32](#_Toc233190539)

[4.2. Parsing 32](#_Toc233190540)

[4.2.1. Tree Representation 33](#_Toc233190541)

[4.2.2. Grammar 34](#_Toc233190542)

[4.2.3. Types of Parsers 34](#_Toc233190543)

[4.2.3.1. Top-down Parser 34](#_Toc233190544)

[4.2.3.2. Bottom-up Chart Parser 36](#_Toc233190545)

[4.3. Syntax Analysis 39](#_Toc233190546)

[4.3.1. Sub Modules 39](#_Toc233190547)

[4.3.1.1. Syntax Analysis Based on Grammar Rules 39](#_Toc233190548)

[4.3.1.2. Syntax Analysis Based on Words 41](#_Toc233190549)

[4.3.1.3. Syntax Analysis Based on Statistics 42](#_Toc233190550)

[4.3. Discourse Analysis 43](#_Toc233190551)

[4.4.1. “Resolution of Anaphora Procedure” Algorithm 43](#_Toc233190552)

[4.4.1.1. RAP main components 43](#_Toc233190553)

[4.4.1.2. Types of Anaphora 44](#_Toc233190554)

[4.4.1.3. Example of the RAP’s output 45](#_Toc233190555)

[4.4. Word Sense Disambiguation 45](#_Toc233190556)

[4.5.1. WordNet 45](#_Toc233190557)

[4.5.2. Lesk algorithm 47](#_Toc233190558)

**Chapter (5) Text Meaning Representation**

[5.1. Meaning Representation 51](#_Toc233190559)

[5.2. Text Meaning Representation 51](#_Toc233190560)

[5.3. Introduction to Text Meaning Representation ‎40 52](#_Toc233190561)

[5.3.1. Text Meaning Representation Structure 52](#_Toc233190562)

[5.3.2. Components of Text Meaning Representation 53](#_Toc233190563)

[5.4.Illustrating the translation from Natural Language to Text Meaning Representation ‎41 53](#_Toc233190564)

[5.5. Case Role and Circumstantial Role Definitions and Examples ‎38 58](#_Toc233190565)

[5.6. Domain Relation Definitions ‎39 62](#_Toc233190566)

[5.7. The use of Ontology in Meaning Representation 64](#_Toc233190567)

[5.8. Design and Implementation 64](#_Toc233190568)

**Chapter (6) Multilevel Generation**

[6.1. Sub-modules 69](#_Toc233190569)

[6.1.1. Weight-based Assignment 69](#_Toc233190570)

[6.1.2. Weight-based Partitioning 70](#_Toc233190571)

[6.1.3. Concept-based Partitioning 71](#_Toc233190572)

[6.2. Design and Implementation 74](#_Toc233190573)

**Chapter (7) Mind Map Conversion**

[7.1. Images in Mind Maps 78](#_Toc233190574)

[7.2. Design and Implementation 79](#_Toc233190575)

**Chapter (8) Automatic Layout Allocation**

[8.1. Applications of Graph Drawing Algorithms 80](#_Toc233190576)

[8.2. Aesthetics 81](#_Toc233190577)

[8.3. The Spring Model Algorithm 82](#_Toc233190578)

[8.4. Simulated Annealing Method 85](#_Toc233190579)

[8.5. Design and Implementation 87](#_Toc233190580)

**Chapter (9) Testing**

[9.1. Text Meaning Representation Testing 91](#_Toc233190581)

[9.2. Multilevel Generation Testing 92](#_Toc233190582)

[9.3. Mind Map Conversion and Automatic Layout Allocation Testing 94](#_Toc233190583)

[9.4. Evaluation 98](#_Toc233190584)

**Conclusion and Future Work**

[1- Conclusion 99](#_Toc233190585)

[2- Future Work 99](#_Toc233190586)

**References** 100

Chapter (1)

Introduction

# Motivation

Mind maps claim to work in a manner similar to the brain, which is not in neat lines. The mind remembers key words and images, not sentences. Mind maps use just key words and key images, allowing a lot more information to be put on a page and also quick memorizing. In addition to that, they have a number of quite intriguing advantages.

Mind Maps are simple and ease of use. They allow associations and links to be recorded and reinforced and with the relative importance of each idea clearly indicated. The organization of a mind map reflects the way your own brain organizes ideas. Starting from the center of the page rather than the top-left corner allows you to work in all directions. The center with the main, more important ideas is more clearly defined, the links between key concepts will be immediately recognized, and the addition of new information is easy. Also, visual and depict associations between keywords are much easier to recall than when expressed in linear lines.

# Project Definition

“Mind Map Generator” created in 2007/2008 was the first tool that generates mind maps automatically with so simple usage. However taking into consideration the huge amount of English language rules and the difficulties of covering them all, the software was only able to understand and process short, simple sentences. The generated mind map output was crammed on the side of the screen and needed to be manually allocated. Finally, the output of any input text was a one level map, not classified or divided in any manner, regardless of what size it might have been.

# Objective

With this project, Mind Map Generator, 2nd Edition, we want to overcome the input text limitations the original project was bound by, and produce a more structured output. We plan on doing so by having the parser cover more English grammar rules, enrich the meaning representation of the text and produce the output in a multileveled or nested mind maps automatically laid out on the screen.

* **Text Meaning Representation enrichment**   
  To overcome the language restrictions in the Mind Map Generator software, since most of the NLP techniques need extremely large information about the language processed words, rules and expressions which are hard to get and not so easily applied and implemented.
* **Multilevel Generation**  
  To enable the program to build up semantic grouping in topics in the given text content and build up relationships between them such that Mind map can be built up as Parent-children relation, in other words each mind map image can be expanded to child mind maps, narrowing down the focus of each mind map.
* **Automatic Layout Allocation**  
  To enable the program to automatically distribute the mind map elements in a clear, user-friendly context, regardless of the amount of data input, to improve the graph readability.

# Documentation Organization and Contents

This document provides a comprehensive explanation of the system. The following chapter, Chapter 2, covers some basic background about mind mapping and natural language processing. Following that, Chapter 3 gives a brief overview of the system architecture. In Chapters 4, 5, 6, 7 and 8, each describe a module of the system. The final chapter, Chapter 9, goes over testing the modules. The modules, in our project scope, tested are Text Meaning Representation Enrichment, Multilevel Mind Map Generation, Mind Map Conversion and Automatic Layout Allocation. Finally, we sum up with a conclusion and some future work recommendations.

Chapter (2)

Background

In these times of exploding data and information overload it is beneficial to reduce complexity in order to have a clear sight of issues and clearly see the solutions. With 80-90% of the information we receive being visual, there has to be means to make things easier to comprehend and understand visually.

Visualization is considered to be the means of choice to convey complex and relevant meaning. It is a member of the family of non-verbal thinking like mathematical and musical thinking. By the same token we make use of drawings, diagrams, maps, icons and the like to simplify the world in order to understand it and act accordingly.

The act of understanding and acting is clearly linked to creativity and innovation which are the two paramount virtues for every person that is dedicated to progress. Seeing the “big picture” and the relationships of causes, parameters, ideas, and solutions is what makes visualization techniques and **mind mapping** in particular so valuable today. By creating connections in a **mind map** between nodes you can easily associate meaning. It is much more efficient than linear text and it’s much more convenient to modify these associations while you are processing your thoughts.

**Mind mappin**g is one of the numerous other visual thinking techniques but it is probably the most popular right now for the many opportunities to foster collaboration and integrations into business processes it can be used in.

# Mind Maps and Mind Mapping

In this section we will discuss all there is to know about mind maps and the mind mapping technique. The history of mind maps, advantages, uses and classifications are discussed in details.

## Definition

A mindmap is a diagram used to represent words, ideas, tasks, or other items linked to and arranged around a central key word or idea. Mind maps are used to generate, visualize, structure, and classify ideas, and as an aid in study, organization, problem solving, decision making, and writing.

Mind Maps are a way of representing associated thoughts with symbols rather than with unrelated words. The mind forms associations almost instantaneously, and "mapping" allows you to write your ideas quicker than expressing them using only words or phrases. They help you avoid thinking linearly and they open you up to creativity and new ways of thinking. They are also more realistic, because most things are not orderly to begin with and they help you get the big picture. They are also helpful because linear thinking is limiting.

The elements of a given mind map are arranged intuitively according to the importance of the concepts, and are classified into groupings, branches, or areas, with the goal of representing semantic or other connections between portions of information. Mind maps may also aid recall of existing memories.

By presenting ideas in a radial, graphical, non-linear manner, mind maps encourage a brainstorming approach to planning and organizational tasks. Though the branches of a mind map represent hierarchical tree structures, their radial arrangement disrupts the prioritizing of concepts typically associated with hierarchies presented with more linear visual cues. This orientation towards brainstorming encourages users to enumerate and connect concepts without a tendency to begin within a particular conceptual framework.

The mind map can be contrasted with the similar idea of concept mapping. The former is based on radial hierarchies and tree structures denoting relationships with a central governing concept, whereas concept maps are based on connections between concepts in more diverse patterns.

Mind mapping, on the other hand, is a way of note taking and planning that mirrors the design of the human brain.

It is a creativity- and productivity-enhancing technique that can improve the learning and efficiency of individuals. Mind mapping uses pictures and/or word phrases to organize and develop thoughts in a non-linear fashion. It helps people "see" a problem and its solution.

Mind mapping –along with mind maps– can be used in nearly every activity where thought, planning, recall or creativity are involved. This means they can be used to generate, visualize, structure, and classify ideas, and as an aid in study, organization, problem solving, decision making, and writing. The following are the characteristics of mind maps:

1. Graphical (Distinguish words or ideas with colors & symbols).
2. Hierarchical or tree branch.
3. Allows for creativity.
4. Associates words with visual representations.
5. Focuses on ONLY one word or idea versus Concept maps that connect multiple words or ideas.

## The History of Mind Maps

Mind maps (or similar concepts) have been used for centuries in learning, brainstorming, memory, visual thinking, and problem solving by educators, engineers, psychologists, and others. Some of the earliest examples of mind maps were developed by Porphyry of Tyros, a noted thinker of the 3rd century, as he graphically visualized the concept categories of Aristotle. Philosopher Ramon Llull (1235 - 1315) also used mind maps.

The semantic network was developed in the late 1950s as a theory to understand human learning and developed into mind maps by Allan M. Collins and M. Ross Quillian during the early 1960s. Due to his commitment and published research, and his work with learning, creativity, and graphical thinking, Collins can be considered the father of the modern mind map.

British popular psychology author Tony Buzan claims to have invented modern mind mapping. He claimed the idea was inspired by Alfred Korzybski's general semantics as popularized in science fiction novels, such as those of Robert A. Heinlein and A. E. van Vogt. Buzan argues that while 'traditional' outlines force readers to scan left to right and top to bottom, readers actually tend to scan the entire page in a non-linear fashion. Buzan also uses popular assumptions about the cerebral hemispheres in order to promote the exclusive use of mind mapping over other forms of note making.

The mind map continues to be used in various forms, and for various applications including learning and education (where it is often taught as 'Webs', 'Mind webs', or 'Webbing'), planning, and in engineering diagramming.

When compared with the concept map (which was developed by learning experts in the 1970s) the structure of a mind map is a similar radial, but is simplified by having one central key word.

It is better to structure your knowledge with the help of clues about knowledge instead of structuring knowledge itself. Mind Maps helps to arrange your knowledge, ideas, words with the help of images those are called mind maps.

## Types of Mind Maps

There are 3 basic types of mind maps, based on their content:

**"Library" maps**  
A Library map is a collection of reference information for the purposes of

* transferring information and knowledge
* storing and recovering useful materials
* understanding or learning something

The focus of a library map is the subject. Topics can be sub-headings or statements that are expanded, and the position of a piece of information represents its relationship to the bigger subject. A common issue with Library maps is that the same piece of information "belongs" in more than one place - chairs can be classified as both furniture and as things made from wood. Not all Furniture is chairs, neither are all Wooden Things chairs. Do we put chairs under Wooden Things or furniture, or both? How do we decide? Or do we use a database instead? One solution is to decide on a major classification scheme where everything appears once, then identify other groupings with colours or icons. Other users will typically navigate the map themselves by following signposts to the resources they seek.

**"Presentation" maps**  
A Presentation map is a story or an argument, designed to

* Inform an audience in a directed fashion
* Argue a proposition or case
* Make a call to action

The focus of a presentation map is the audience themselves, although you would not necessarily write that in the central topic. The positioning of information is relative to the audience's viewpoint, and the deeper it is in the map, the more involved they are. The development of arguments away from the centre builds on inductive thinking that justifies or amplifies higher level statements, and if your audience is still with you, then you are well placed to add more detail to the foundations you have laid. Topics near to the centre are major statements that easily connect to your audience's world view. You can see this kind of architecture in most marketing materials - the opening gambit is usually to get you to identify with a problem or issue. Presentation maps should use statements rather than headings, and should retain their integrity when viewed at different levels of detail. You can then use the same map for the two-minute briefing to the board of directors, or the two-hour version for the technical nit-pickers. Presentation maps will also need to follow a sequence, for example clockwise.

**"Tunnel Timeline" maps**  
A Tunnel Timeline map is a map that is designed around delivering an outcome. The primary purpose of this kind of map is to ***visualize success***. You are drawing a picture of what success looks like, and showing the actions on the path or paths to reach it. Use Tunnel Timelines for

* Project outlines and plans
* Strategies
* Problem solving

Topics at or near the centre of the map represent the successful outcome and topics near the edges are the *next actions* to take towards those outcomes. The relationship with a "tunnel" is that the map shows your project as the light at the end of the tunnel, with your next actions around the walls nearest to you. As you make progress towards the centre, you complete the actions and decisions along the way. Major milestone topics in the map should be written as outcomes to keep you focused on achievement, e.g. "reject rate is 3%" or "client has renewed contract". This is the key benefit of visualizing projects in software mind maps - you stay focused on your objective and keep your eye on where you want to go, not on your short-term direction. Responding to changes and obstacles is easier if you are focused on the big target, and a continual visual reminder of objectives is a positive force. Participants in your project or strategy can see where it is headed, and can understand how their contribution takes it forward.

But these are not the only types of mind maps, only those based on its content and that occur frequently in business life.

There is another type of classification that differentiates maps not on their content, but on their life span:

**"Brainstorming" maps  
These are often used just during one session. You dump all your ideas or even better in a group session the ideas of the whole team, structure the map, set some priorities, decide on some next actions and afterwards the map is not needed any more. It has “just” facilitated the process. The life time of those maps is in the range of a few hours.**

**"Project" maps  
These are used to plan an event, a product launch or how to approach a sales deal, live until the project is finished. Afterwards they are not needed any more. Those maps are usually used for days or a few weeks. They are updated again and again to reflect the latest status of the project.**

**"Knowledge" maps  
These contain information that you write down once and are then kept for a long time. They usually replace other types of documents like Word or PowerPoint. Those maps can live very long, even for years. Occasionally they need to be updated, but often they are never touched after their first creation.**

## Advantages and Uses of Mind Maps

We can start demonstrating the advantages of mind maps by emphasizing on the disadvantages of traditional linear notes.

**Disadvantages of traditional linear notes**

* Energy and time wasted writing down superfluous words.
* Other information may be missed while noting down one idea.
* Take longer to read and review.
* Associations and connections between key words and ideas not readily apparent.
* Attention wanders easily.
* Lack of color and other visual qualities handicap memory.
* Traditional notes aid forgetting not memory.

**Advantages of Mind Maps**

Mind maps work the way the brain works -- which is not in nice neat lines. The following are some advantages of mind maps.

**Simplicity:** Once one gets used to drawing mind maps, it becomes so simple to use. **Associability:** Memory is naturally associative, not linear. Any idea probably has thousands of links in your mind. Mind maps allow associations and links to be recorded and reinforced. The relative importance of each idea is clearly indicated. More important ideas will be nearer the centre.  
**Use of Keywords and Images:** The mind remembers key words and images, not sentences -- try recalling just one sentence from memory!   Mind maps use just key words and key images, allowing a lot more information to be put on a page.  **Easy to Recall:** Because mind maps are more visual and depict associations between key words, they are much easier to recall than linear notes. **Radial Organization:** Starting from the center of the page rather than top-left corner allows you to work out in all directions. The organization of a mind map reflects the way your own brain organizes ideas. The centre with the main idea is more clearly defined. The links between key concepts will be immediately recognized. **Ease of review:** Mind maps are easy to review. Regular review reinforces memory. Best is to try reviewing in your imagination first, then go back and check on those areas that were hazy. **Visual quality:** We remember what stands out (where were you when John Lennon was shot?). Visual quality of mind maps allows you to make key points to stand out easily.

**Uses and Applications**

Mind Mapping can be applied to every aspect of life where improved learning and clearer thinking will enhance human performance.

* **Note Taking:**Whenever information is being taken in, mind maps help organize it into a form that is easily assimilated by the brain and easily remembered. They can be used for noting anything -- books, lectures, meetings, interviews, phone conversations.
* **Recalling Information:**Whenever information is being retrieved from memory, mind maps allow ideas to be quickly noted as they occur, in an organized manner. There's no need to form sentences and write them out in full. They serve as quick and efficient means of review and so keep recall at a high level.
* **Encourage Creativity:**Whenever you want to encourage creativity, mind maps liberate the mind from linear thinking, allowing new ideas to flow more rapidly. Think of every item in a mind map as the center of another mind map.
* **Problem solving:**Whenever you are confronted by a problem -- professional or personal -- mind maps help you see all the issues and how they relate to each other. They also help others quickly get an overview of how you see different aspects of the situation, and their relative importance.
* **Planning:**Whenever you are planning something, mind maps help you get all the relevant information down in one place and organize it easily. They can be used for planning any piece of writing from a letter to a screenplay to a book (I use a master map for the whole book, and a detailed sub-map for each chapter), or for planning a meeting, a day or a vacation.
* **Preparing Presentations:**Whenever I speak I prepare a mind map for myself of the topic and its flow. This not only helps me organize the ideas coherently; the visual nature of the map means that I can read the whole thing in my head as I talk, without ever having to look at a sheet of paper.

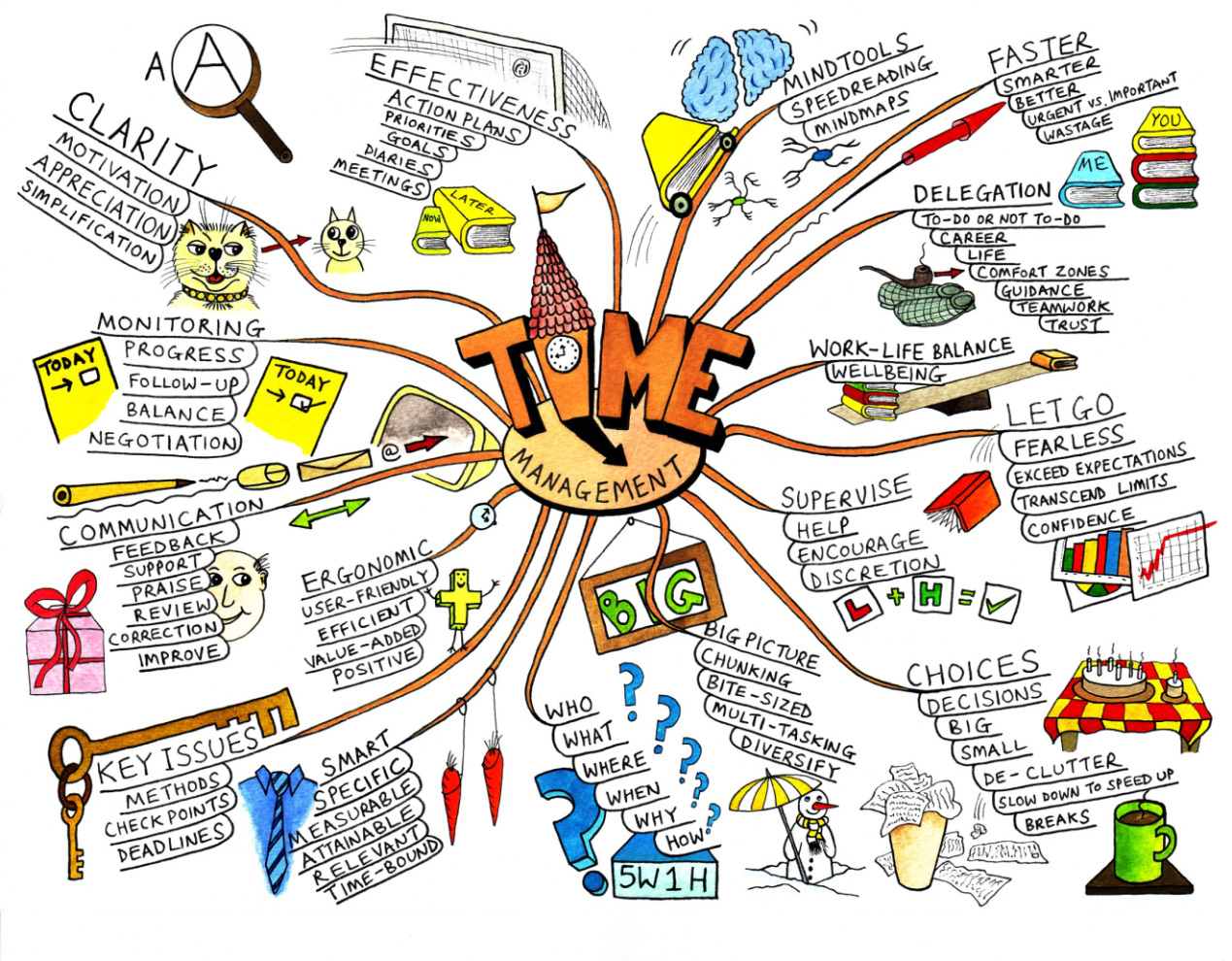
Example  
The following figure is an illustration of how a hand drawn mind map might look like.

Figure 2.1 Example of Mind Map

# Natural Language Processing

"Understanding" language means, among other things, knowing what concepts a word or phrase stands for and knowing how to link those concepts together in a meaningful way. It's ironic that natural language, the symbol system that is easiest for humans to learn and use, is hardest for a computer to master. Long after machines have proven capable of inverting large matrices with speed and grace, they still fail to master the basics of our spoken and written languages.

The challenges we face stem from the highly ambiguous nature of natural language. As an English speaker you effortlessly understand a sentence like "Flying planes can be dangerous". Yet this sentence presents difficulties to a software program that lacks both your knowledge of the world and your experience with linguistic structures. Is the more plausible interpretation that the pilot is at risk, or that the danger is to people on the ground? Should "can" be analyzed as a verb or as a noun? Which of the many possible meanings of "plane" is relevant? Depending on context, "plane" could refer to, among other things, an airplane, a geometric object, or a woodworking tool. How much and what sort of context needs to be brought to bear on these questions in order to adequately disambiguate the sentence?

## The meaning of Natural Language Processing

Natural language processing (NLP) is a field of computer science and linguistics concerned with the interactions between computers and human (natural) languages. Natural language generation systems convert information from computer databases into readable human language. Natural language understanding systems convert samples of human language into more formal representations such as parse trees or first order logic that are easier for computer programs to manipulate. Many problems within NLP apply to both generation and understanding; for example, a computer must be able to model morphology (the structure of words) in order to understand an English sentence, and a model of morphology is also needed for producing a grammatically correct English sentence.

NLP has significant overlap with the field of computational linguistics, and is often considered a sub-field of artificial intelligence. The term natural language is used to distinguish human languages (such as Spanish, Swahili or Swedish) from formal or computer languages (such as C++, Java or LISP). Although NLP may encompass both text and speech, work on speech processing has evolved into a separate field.

Natural-language understanding is sometimes referred to as an AI-complete problem, because natural-language recognition seems to require extensive knowledge about the outside world and the ability to manipulate it. The definition of "understanding" is one of the major problems in natural-language processing.

The goal of NLP evaluation is to measure one or more qualities of an algorithm or a system, in order to determine whether (or to what extent) the system answers the goals of its designers, or meets the needs of its users. Research in NLP evaluation has received considerable attention, because the definition of proper evaluation criteria is one way to specify precisely an NLP problem, going thus beyond the vagueness of tasks defined only as language understanding or language generation. A precise set of evaluation criteria, which includes mainly evaluation data and evaluation metrics, enables several teams to compare their solutions to a given NLP problem.

## NLP Sub-problems

Natural language processing can be divided into the following problems:

**Speech Segmentation**   
In most spoken languages, the sounds representing successive letters blend into each other, so the conversion of the analog signal to discrete characters can be a very difficult process. Also, in natural speech there are hardly any pauses between successive words; the location of those boundaries usually must take into account grammatical and semantic constraints, as well as the context.

**Text Segmentation**  
Some written languages like Chinese, Japanese and Thai do not have single-word boundaries either, so any significant text parsing usually requires the identification of word boundaries, which is often a non-trivial task.

**Part of Speech Tagging**  
Is the process of marking up the words in a text (corpus) as corresponding to a particular part of speech, based on both its definition, as well as its context —i.e. relationship with adjacent and related words in a phrase, sentence, or paragraph.

**Word Sense Disambiguation**  
Many words have more than one meaning; we have to select the meaning which makes the most sense in context.

**Syntactic Ambiguity**  
The grammar for natural languages is ambiguous, i.e. there are often multiple possible parse trees for a given sentence. Choosing the most appropriate one usually requires semantic and contextual information.

**Speech Acts and Plans**  
A sentence can often be considered an action by the speaker. The sentence structure alone may not contain enough information to define this action. For instance, a question is actually the speaker requesting some sort of response from the listener. The desired response may be verbal, physical, or some combination. For example, "Can you pass the class?" is a request for a simple yes-or-no answer, while "Can you pass the salt?" is requesting a physical action to be performed. It is not appropriate to respond with "Yes, I can pass the salt," without the accompanying action (although "No" or "I can't reach the salt" would explain a lack of action).

## Major tasks in Natural Language Processing

The following are considered to be the most important tasks in the Natural Language Processing field:

**Automatic summarization**The creation of a shortened version of a text by a computer program. The product of this procedure still contains the most important points of the original text.   
  
**Foreign language reading aid**A computer program that assists a non-native language user to read properly in their target language. The proper reading means that the pronunciation should be correct and stress to different parts of the words should be proper.  
 **Foreign language writing aid**A computer program that assists a non-native language user in writing decently in their target language.

**Information extraction**A type of information retrieval whose goal is to automatically extract structured information, i.e. categorized and contextually and semantically well-defined data from a certain domain, from unstructured machine-readable documents.

**Information retrieval (IR)**IR is concerned with storing, searching and retrieving information. It is a separate field within computer science (closer to databases), but IR relies on some NLP methods (for example, stemming). Some current research and applications seek to bridge the gap between IR and NLP.

**Machine translation**Automatically translating from one human language to another.

**Named entity recognition (NER)**Given a stream of text, determining which items in the text map to proper names, such as people or places. Although in English, named entities are marked with capitalized words, many other languages do not use capitalization to distinguish named entities.

**Natural language generation**Thenatural language processingtask of generatingnatural languagefrom a machine representation system such as aknowledge baseor a logical form.

**Natural language understanding**Natural Language Understanding deals with machinereading comprehension. Reading comprehension is defined as the level of understanding of a writing.

**Optical character recognition**The mechanical or electronic translation of images of handwritten, typewritten or printed text (usually captured by a scanner) into machine-editable text.

**Anaphora resolution**refers to the way in which a word or phrase relates to other text.

**Question answering**Given a human language question, the task of producing a human-language answer. The question may be a closed-ended (such as "What is the capital of Canada?") or open-ended (such as "What is the meaning of life?").

**Speech recognition**Given a sound clip of a person or people speaking, the task of producing a text dictation of the speaker(s). (The opposite of text to speech.)

**Spoken dialogue system**  
Spoken dialogue system is a dialog system delivered through voice.

**Text simplification**Text simplification is an operation used to modify, enhance, classify or otherwise process an existing corpus of human-readable text in such a way that the grammar and structure of the prose is greatly simplified, while the underlying meaning and information remains the same.

**Text-to-speech**  
Speech synthesis is the artificial production of human speech.

**Text-proofing**  
Proof-reading traditionally means reading a proof copy of a text in order to detect and correct any errors.

## Required Knowledge

A natural language-system must use considerable knowledge about the structure of the language itself, including what the words are, how words combine to form sentences, what the words mean, how word meanings contribute to sentence meanings, and so on. However, we cannot completely account for linguistic behavior without also taking into account another aspect of what makes humans intelligent — their general world knowledge and their reasoning abilities.

The following are some of the different forms of knowledge relevant for natural language understanding:

**Phonetic and phonological knowledge -** concerns how words are related to the sounds that realize them.

**Morphological knowledge** - concerns how words are constructed from more basic meaning units called morphemes. A mor­pheme is the primitive unit of meaning in a language (for example, the meaning of the word "*friendly"* is derivable from the meaning of the noun "*friend"* and the suffix "*-ly",* which transforms a noun into an adjective).

**Syntactic knowledge** - concerns how words can be put together to form correct sentences and determines what structural role each word plays in the sentence and what phrases are subparts of what other phrases.

**Semantic knowledge** - concerns what words mean and how these meanings combine in sentences to form sentence meanings. This is the study of context-independent meaning - the mean­ing a sentence has regardless of the context in which it is used.

**Pragmatic knowledge** - concerns how sentences are used in different situations and how use affects the interpretation of the sentence.

**Discourse knowledge -** concerns how the immediately preceding sentences affect the interpretation of the next sentence. This information is especially important for interpreting pronouns and for interpreting the temporal aspects of the information conveyed.

**World knowledge** - includes the general knowledge about the struc­ture of the world that languages users must have in order to, for example, maintain a conversation. It includes what each lan­guage user must know about the other user’s beliefs and goals.

These definitions are imprecise and are more characteristics of knowledge than actual distinct classes of knowledge. Any particular fact might include aspects from several different levels, and an algorithm might need to draw from several different levels simultaneously.

Chapter (3)

System Architecture

The text input by the user will the system's frontend, where it will undergo some natural language processing procedures. The output of the frontend is a parse tree for each sentence entered, to which discourse analysis and word sence disambiguation are applied. This output is used to generate text meaning representation for the inout text. The text meaning representation is then passed on to the multi-level generation phase, where it undergoes some semantic grouping and restructuring, so that when the text meaning representation is converted to mind map, the mind map produced may be multi-leveled or nested with the most relevant keywords at the highest level, and more details can be seen as we go deeper into the map. The final phase, before displaying the mind map to the user is, to automatically lay out the map onto the screen.

Figure 3.1 System Architecture

# Mind Map Generator Frontend (Natural Language Processing)

It is the first phase in the system which is responsible for understanding the entered text. It mainly consists of the efforts put into this project, in its first attempt, the previous year.

Figure 3.2 MMG frontend Input Output

This phase consists of five sub-phases of natural language processing

## Morphological Analysis

Morphology is about the inner structure of words. It is interested in what are the smallest units in a word that bear some meaning and how can they be combined to form words. It is the identification, analysis and description of the structure of words. Morphology is the branch of linguistics that studies patterns of word formation within and across languages, and attempts to formulate rules that model the knowledge of the speakers of those languages.

The goal of is to find out what morphemes a given word is built from. For example, a morphological parser should be able to tell us that the word *cats* is the plural form of the noun stem *cat*, and that the word *mice* is the plural form of the noun stem *mouse*.

## Parsing

Parsing is a term for the diagramming of sentences of natural languages. It considers methods of describing the structure of sentences and explores ways of characterizing all the legal structures in a language. The most common way of representing how a sentence is broken into its major subparts and how those subparts are broken up in turn, is as a tree.

To construct a tree structure for a sentence, you must know what structures are legal for English. A set of rewrite rules describes what tree structures are allowed. These rules say that a certain symbol may be expanded in the tree by a sequence of other symbols. In order to parse natural language data, researchers must first agree on the grammar to be used. The choice of syntax is affected by both linguistic and computational concerns.

Human sentences are not easily parsed by programs, as there is substantial ambiguity in the structure of human language, whose usage is to convey meaning (or semantics) amongst a potentially unlimited range of possibilities but only some of which are germane to the particular case.

## Syntax Analysis

Syntax Analyzer is responsible for the production of the final parse trees of the input text. It is responsible for the modification, addition, deletion or selection of parse trees produced by the Parser, and based on the Morphological Analyzer results.

It has three sub-modules or parts based on the source of the rules used in each part. The first includes the removal of incorrect parse trees. These incorrect parse trees result from being unable to include the context-free grammar rules that prevent these parse trees. The source of the rules of this module is the English grammar rules or recommendations. The second includes the addition, modification or deletion of the parse trees based on the nature of the words used in the sentence. This includes the lexical items that consist of two or more words, such as phrasal verbs or compound pronouns, and checking the syntax structure of some words. The third includes the selection of some parse trees and removal of the rest based on statistical techniques. This part is responsible for the selection of parse tree structures that are more likely to be mentioned in the text.

## Discourse Analysis

Discourse Analysisis concerned with assigning each pronoun to the noun which this pronoun refers to. To do this, the “Resolution of Anaphora Procedure” algorithm is used. The “Resolution of Anaphora Procedure” algorithm is an algorithm for identifying both intra-sentential and inter-sentential antecedents of pronouns in text. . It relies on measures of salience derived from syntactic structure and a simple dynamic model to select the antecedent noun phrase of a pronoun from a list of candidates. It does not employ semantic conditions or real-world knowledge in evaluating candidate antecedents; nor does it model intentional or global discourse structure.

## Word Sense Disambiguation

Word Sense Disambiguation is concerned with assigning the most proper sense for each word according to the formulation of the sentence. For example, the word "ball" has several senses, including a round object used in games, a formal dance, and a pitch in baseball that is not a strike. A precise method for quantifying how similar two word senses are is called a measure of semantic relatedness.

Word morphemes

Figure 3.3 Phases of Frontend

Text Meaning Representation

Pronoun references

Word Senses

Modified Parse trees

Word Sense Disambiguation

Discourse Analysis

Syntax Analysis

Parse trees

Parsing

Morphological Analysis

Input text

# Text Meaning Representation

It is responsible for putting the text in a form that best represents its meaning, as it makes the computer able to understand the meaning of all the words and the relation between the words of the whole text.

Figure 3.4 Text Meaning Representation Input Output

Designing a meaning representation for NLP involves determining its content and its representation. We address the issue of content in terms of the nature of primitives used to share knowledge between linguistic and world knowledge representations. The structure of the representation is shown to be based on the needs for composing the primitives in different ways, the expressiveness of which is determined by the machine translation task and its needs for linguistic and world knowledge.

It is a language-neutral description of the meaning conveyed in a natural language text, and is derived by syntactic, semantic, and pragmatic analysis of the text. Because the TMR is intended to be language neutral, it is also deliberately syntax neutral, and avoids using terminology like clause, proposition, tense, etc., which are associated more closely with the syntactic structure of a particular language. In addition to providing information about the lexical-semantic dependencies in the text, the TMR represents stylistic factors, discourse relations, speaker attitudes, and other pragmatic factors present in the discourse structure. In doing so, the TMR captures not only the meaning of individual elements in the text, but also the relations between those elements, and captures both propositional and non-propositional components of textual meaning.

# Multi-level Generation

The Multi-level Generation phase is responsible for generating a multi-leveled text meaning representation. Every set of related actions and information -in the original Text Meaning Representation-are grouped into one common concept to help organize the TMR into an easily read form

Figure 3.5 Multi-level generation Input Output

The algorithm used in the multi-level generation phase goes through three steps before producing the final output of this phase. These three steps are: Weight Assignment, Weight-Based Partitioning, and Concept-Based Partitioning.

Weight assignment is concerned with the assignment of weights to noun and verb frames to indicate their level of importance. Weight-Based Partitioning is carried out is to identify the main noun frames present in the TMR. Concept-Based Partitioning groups related information under one common concept to provide a better layout for the text meaning representation and make it easier to follow and understand.

# Mind Map Conversion

The mind map conversion phase is the last phase that takes place before displaying the final output to the reader. This phase is concerned with the conversion of the obtained multileveled text meaning representation into an automatically laid out mind map that contains images to attract the user’s attention and aid in remembering and understanding the input text.

Figure 3.6 Mind Map Conversion Input Output

# Automatic Layout Allocation

Automatic Layout Allocation is done mainly in response to requirements of data visualization.

Figure 3.7 Automatic Layout Input Output

The quality or usefulness is highly dependent on its application domain. Therefore a graph drawing algorithm must take into account *aesthetics*: criteria for making salient characteristics of the graph easily readable. Readability and “salient characteristics” are highly subjective and dependent on the purpose for which the drawing is generated. Some aesthetic criteria include:

* minimize the number of edge crossings
* draw edges as straight as possible
* vertices should be evenly distributed
* the majority directed edges should be drawn pointing in the same direction
* minimize bends in the edges
* minimize the area of the area drawing
* maximize display of symmetries
* maximize *angular resolution*

Chapter (4)

Mind Map Generator Frontend   
(previous work)

This phase, the frontend, takes in the text input by the user onto which it carries out some natural language processing. The sub-phases that take place are as shown in the diagram below; morphological analysis is the first to take place, followed by parsing, where every possible parse tree for each sentence is built. The syntax analysis is where the correct parse tree for each sentence is chosen. Following that, discourse analysis and word sense disambiguation take place. In this chapter we give a brief overview for each.

Figure 4.1 Frontend phases

# Morphological Analysis

Morphological Analysis is concerned by how words are constructed from more basic units called morphemes.

For example: Friendly (Adjective) = friend (noun) + ly (suffix)

## 4.1.1. Morphology

Morphology is about the inner structure of words. It is interested in what are the smallest units in a word that bear some meaning and how can they be combined to form words.

For example the word “*rabbits”* has two units which contribute to the meaning of the word: ‘*rabbit’* contributes the main meaning, and ‘s’ adds the information that the word is plural. The smallest unit in a word that bears some meaning, such as *rabbit* and *s*, are called *morphemes*.

Morphology is the identification, analysis and description of the structure of words. While words are generally accepted as being the smallest units of syntax, it is clear that in most (if not all) languages, words can be related to other words by rules. For example, English speakers recognize that the words *dog*, *dogs*, and *dog catcher* are closely related. English speakers recognize these relations from their tacit knowledge of the rules of word formation in English. They infer intuitively that *dog* is to *dogs* as *cat* is to *cats*; similarly, *dog* is to *dog catcher* as *dish* is to *dishwasher*. The rules understood by the speaker reflect specific patterns (or regularities) in the way words are formed from smaller units and how those smaller units interact in speech. In this way, morphology is the branch of linguistics that studies patterns of word formation within and across languages, and attempts to formulate rules that model the knowledge of the speakers of those languages.

Then, how are morphemes combined to form words that are legal in some language. Two kinds of processes can be distinguished here. They are called *inflection* and *derivation*. Inflection is usually taken to be the process of adding a grammatical affix to a word stem, forming a word of the same class as the stem. Adding plural *s* to a noun stem, for example, is an inflectional process. Derivation is when adding an affix to a word stem results in a word with a class different from that of the stem. Making a noun out of a verb by adding *ation* to the verb is an example: “*realize”* + “*ation”* gives us “*realization”*.

Let's look at the example of inflection of nouns in English in more detail. Basically, English nouns only come in two forms: singular and plural. In the standard case, you get the plural form by adding an *s* to the end of the noun stem, which, at the same time, is the singular form of the noun: *rabbit* vs. *rabbits*. However, there are some exceptions, where the plural is not built by simply adding an *s* to the stem, but rather by changing the stem: *foot* vs. *feet*. So, valid English nouns consist of either the stem of a regular noun, or the singular stem of an irregular noun, or the plural stem of an irregular noun, or the stem of a regular noun plus an s.

The nice thing about these morphological rules is that they can be captured using finite state techniques, which means that you can draw a finite state automaton describing the inflectional morphology of English noun phrases. There is one more complication, however, and that is that sometimes, when two morphemes are combined, additional changes happen at the boundary. When combining the noun stem “*fox”*, for instance, with the plural morpheme “*s”* we get “*foxes*“ instead of “*foxs”*.

## 4.1.2. Morphological Parsing

The goal of *morphological parsing* is to find out what morphemes a given word is built from. For example, a morphological parser should be able to tell us that the word *cats* is the plural form of the noun stem *cat*, and that the word *mice* is the plural form of the noun stem *mouse*. So, given the string *cats* as input, a morphological parser should produce an output that looks similar to *cat NPL*.

Here are some more examples:

Mouse 🡪 Mouse N SG  
 Mice 🡪 Mouse N PL  
 Foxes 🡪 Fox N PL

Morphological parsing yields information that is useful in many NLP applications. In parsing, e.g., it helps to know the agreement features of words. Similarly, grammar checkers need to know agreement information to detect such mistakes. But morphological information also helps spell checkers to decide whether something is a possible word or not, and in information retrieval it is used to search not only *cats*, if that's the user's input, but also for *cat*.

To get from the surface form of a word to its morphological analysis, we are going to proceed in two steps. First, we are going to split the words up into its possible components. So, we will make *cat + s* out of *cats*, using *+* to indicate morpheme boundaries. In this step, we will also take spelling rules into account, so that there are two possible ways of splitting up *foxes*, namely *foxe + s* and *fox + s*. The first one assumes that *foxe* is a stem and *s* the suffix, while the second one assumes that the stem is *fox* and that the *e* has been introduced due to the spelling rule that we saw above.

In the second step, we will use a lexicon of stems and affixes to look up the categories of the stems and the meaning of the affixes. So, *cat + s* will get mapped to *cat NP PL*, and *fox + s* to *fox N PL*. We will also find out now that *foxe* is not a legal stem. This tells us that splitting *foxes* into *foxe + s* was actually an incorrect way of splitting *foxes*, which should be discarded. But note that for the word *houses* splitting it into *house + s* is correct.

## 4.1.3. WordNet 2.1 Morphological Analyzer

The morphological analyzer of WordNet.Net 2.1 consists of a set of rules for the regular verbs, nouns, adjectives and adverbs. For irregular verbs, nouns and adjectives and adverbs, WordNet.Net 2.1 has an exceptional list for each part of speech that contains the word and its main morpheme.

# Parsing

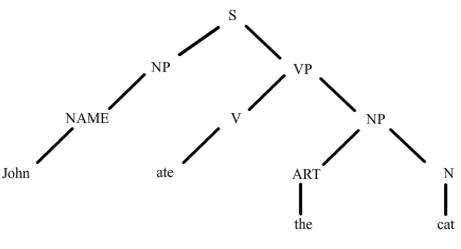
Parsing is a term for the diagramming of sentences of natural languages. It considers methods of describing the structure of sentences and explores ways of characterizing all the legal structures in a language.

## 4.2.1. Tree Representation

The most common way of representing how a sentence is broken into its major subparts and how those subparts are broken up in turn, is a tree. The tree representation for the sentence "*John ate the cat"* is shown in Figure 4.2. This illustration can be read as follows: The sentence (S) consists of an initial noun phrase (NP) and a verb phrase (VP). The initial noun phrase is made of the simple (NAME) "*John".* The verb phrase is composed of a verb (V) "*ate"* and an NP, which consists of an article (ART) "*the"* and a common noun (N) "*cat".* In list notation this same structure could be represented as

**(S (NP (NAME John))** **(VP (V ate) (NP (ART the)** **(N cat))))**

For example the tree representation for the sentence “*John ate the cat” is*



*Figure 4.2 Tree Representation*

To construct a tree structure for a sentence, you must know what structures are legal for English. A set of rewrite rules describes what tree structures are allowed. These rules say that a certain symbol may be expanded in the tree by a sequence of other symbols.

In some machine translation and natural language processing systems, human languages are parsed by computer programs. Human sentences are not easily parsed by programs, as there is substantial ambiguity in the structure of human language, whose usage is to convey meaning (or semantics) amongst a potentially unlimited range of possibilities but only some of which are germane to the particular case.

In order to parse natural language data, researchers must first agree on the grammar to be used. The choice of syntax is affected by both linguistic and computational concerns; for instance some parsing systems use lexical functional grammar, but in general, parsing for grammars of this type is known to be NP-complete. Head-driven phrase structure grammar is another linguistic formalism which has been popular in the parsing community, but other research efforts have focused on less complex formalisms.

## 4.2.2. Grammar

Grammars consisting entirely of rules with a single symbol on the left-hand side, called the mother, are called context-free grammars (CFGs). Context-free grammars are a very important class of grammars for two reasons: The formalism is powerful enough to describe most of the structure in natural languages, yet it is restricted enough so that efficient parsers can be built to analyze sentences. Symbols that cannot be further decomposed in a grammar, namely the words in the preceding example, are called terminal symbols. The other symbols, such as NP, VP, and S, are called non-terminal symbols. The grammatical symbols such as N and V that describe word categories are called lexical symbols. Of course, many words will be listed under multiple categories. For example, *"can"* would be listed under V and N.

The following is a sample grammar, the statements with which the parse tree in Figure 4.2 was built.

1. S 🡪 NP VP
2. VP 🡪 V NP
3. NP 🡪 NAME
4. NP 🡪 ART N
5. NAME 🡪 John
6. V 🡪 ate
7. ART 🡪 the
8. N 🡪 cat

## 4.2.3. Types of Parsers

The task of the parser is essentially to determine if and how the input can be derived from the start symbol of the grammar. This can be done in essentially two ways; top-down parser and bottom-up parser. Top-down parsing can be viewed as an attempt to find left-most derivations of an input-stream by searching for parse trees using a top-down expansion of the given formal grammar rules, while a bottom-up parser starts with the input and attempts to rewrite it to the start symbol.

### 4.2.3.1. Top-down Parser

Like the parser used in our project, a top-down parser starts with the S symbol and attempts to rewrite it into a sequence of terminal symbols that matches the classes of the words in the input sentence.

The state of the parse at any given time can be represented as a list of symbols that are the result of operations applied so far, called the symbol list. For example, the parser starts in the state (S) and after applying the rule S 🡪 NP VP the symbol list will be (NP VP). If it then applies the rule NP 🡪 ART N, the symbol list will be (ART N VP), and so on.

The parser could continue in this fashion until the state consisted entirely of terminal symbols, and then it could check the input sentence to see if it matched. But this would be quite wasteful, for a mistake made early on is not discovered until much later. A better algorithm checks the input as soon as it can. In addition, rather than having a separate rule to indicate the possible syntactic categories for each word, a structure called the lexicon is used to efficiently store the possible categories for each word.

With a lexicon specified, a grammar need not contain any lexical rules. Given these changes, a state of the parse is now defined by a pair: a symbol list similar to before and a number indicating the current position in the sentence. Positions fall between the words, with 1 being the position before the first word. For example, here is a sentence with its positions indicated:

**1 The 2 dogs 3 cried 4**

**1. S -> NP VP**

**2. NP -> ART N**

**3. NP -> ART ADJ N**

**4. VP -> V**

**5. VP -> V NP**

A typical parse state would be ((N VP) 2), indicating that the parser needs to find an N followed by a VP, starting at position two. New states are generated from old states depending on whether the first symbol is a lexical symbol or not. If it is a lexical symbol, like N in the preceding example, and if the next word can belong to that lexical category, then you can update the state by removing the first symbol and updating the position counter. In this case, since the word "*dogs"* is listed as an N in the lexicon, the next parser state would be ((VP) 3) which means it needs to find a VP starting at position 3. If the first symbol is a non-terminal, like VP, then it is rewritten using a rule from the grammar. For example, using rule 4, the new state would be ((V) 3) which means it needs to find a V starting at position 3. On the other hand, using rule *5,* the new state would be ((V NP) 3).

A parsing algorithm, that is guaranteed to find a parse if there is one, must systematically explore every possible new state. One simple technique for this is called backtracking. Using this approach, rather than generating a single new state from the state ((VP) 3), you generate all possible new states. One of these is picked to be the next state and the rest are saved as backup states. If you ever reach a situation where the current state cannot lead to a solution, you simply pick a new current state from the list of backup states. Following is the algorithm in a little more detail.

### 4.2.3.2. Bottom-up Chart Parser

The main difference between top-down and bottom-up parsers is the way the grammar rules are used. For example, consider the rule :

**NP -> ART ADJ N**

In a top-down system you use the rule to find an NP by looking for the sequence ART ADJ N. In a bottom-up parser you use the rule to take a sequence ART ADJ N that you have found and identify it as an NP. The basic operation in bottom-up parsing then is to take a sequence of symbols and match it to the right-hand side of the rules. You could build a bottom-up parser simply by formulating this matching process as a search process. The state would simply consist of a symbol list, starting with the words in the sentence. Successor states could be generated by exploring all possible ways to

* rewrite a word by its possible lexical categories
* replace a sequence of symbols that matches the right-hand side of a grammar rule by its left-hand side symbol

Unfortunately, such a simple implementation would be prohibitively expensive, as the parser would tend to try the same matches again and again, thus dupli­cating much of its work unnecessarily. To avoid this problem, a data structure called a chart is introduced that allows the parser to store the partial results of the matching it has done so far so that the work need not be reduplicated.

Matches are always considered from the point of view of one constituent, called the key. To find rules that match a string involving the key, look for rules that start with the key, or for rules that have already been started by earlier keys and require the present key either to complete the rule or to extend the rule. For instance, consider The Grammar shown below.

**1. S -> NP PVP**

**2. NP -> ART ADJ N**

**3. NP -> ART N**

**4. NP -> ADJ N**

**5. VP -> AUX VP**

**6. VP -> V NP**

Assume you are parsing a sentence that starts with an ART. With this ART as the key, rules 2 and 3 are matched because they start with ART. To record this for analyzing the next key, you need to record that rules 2 and 3 could be continued at the point after the ART. You denote this fact by writing the rule with a dot (o), indicating what has been seen so far. Thus you record

**2'. NP -> ART oADJ N**

**3'. NP -> ART o N**

If the next input key is an ADJ, then rule 4 may be started, and the modi­fied rule 2 may be extended to give

**2''. NP -> ART ADJ o**

The chart maintains the record of all the constituents derived from the sentence so far in the parse. It also maintains the record of rules that have matched partially but are not complete. These are called the active arcs. For example, after seeing an initial ART followed by an ADS in the preceding example, you would have the chart shown in Figure 4.3. You should interpret this figure as follows. There are two completed constituents on the chart: ART1 from position 1 to 2 and ADJ1 from position 2 to 3. There are four active arcs indi­cating possible constituents. These are indicated by the arrows and are interpreted as follows (from top to bottom). There is a potential NP starting at position 1, which needs an ADJ starting at position 2. There is another potential NP starting at position 1, which needs an N starting at position 2. There is a potential NP starting at position 2 with an ADS, which needs an N starting at position 3. Finally, there is a potential NP starting at position 1 with an ART and then an ADJ, which needs an N starting at position 3.

The basic operation of a chart-based parser involves combining an active arc with a completed constituent. The result is either a new completed constituent or a new active arc that is an extension of the original active arc. New completed constituents are maintained on a list called the agenda until they themselves are added to the chart. This process is defined more precisely by the arc extension algorithm shown in Figure 4.4. Given this algorithm, the bottom-up chart parsing algorithm is specified in Figure 4.5.

Do until there is no input left:

1. If the agenda is empty, look up the interpretations for the next word in the input and add them to the agenda.

2. Select a constituent from the agenda (let’s call it constituent C from position p1 to p2).

3. For each rule in the grammar of form **X -> C X1 ... Xn**, add an active arc of form **X -> C o C o X1.... Xn** from position p1 to p2.

4. Add C to the chart using the arc extension algorithm above.

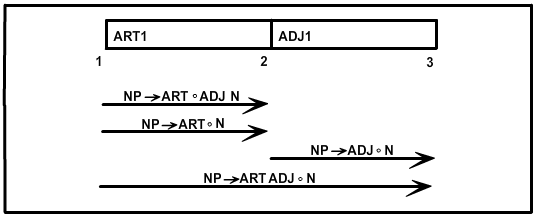
4.5. Bottom-up chart parsing algorithm

Figure 4.4. Arc extension algorithm

To add a constituent C from position p1 to p2:

1. Insert C into the chart from position p1 to p2.
2. For any active arc the form X 🡪 X1 …º C … Xn from position p0 to p1. add a new active arc X 🡪 X1 … C º … Xn from position p0 to p2.
3. For any active arc of the form X 🡪 X1 … Xn º C from position P0 to P1. then add a new constituent of type X from P0 to P2 to the agenda.

Figure 4.3.: The chart after seeing an ADJ in position 2



# Syntax Analysis

Syntax Analyzer is the module of Natural Language Processing programs which produce the final parse trees of the input text. Some books include the Morphological Analyzer and the Parser in the Syntax Analyzer.

The Syntax Analyzer mentioned here is the module responsible for the modification, addition, deletion or selection of parse trees produced by the Parser, and based on the Morphological Analyzer results. Filtering is done based on a score given to each parse tree based on its internal structure.

## 4.3.1. Sub Modules

The Syntax Analyzer is dependent mainly on the possible parse trees structures produced by the Parser. It has three sub-modules or parts based on the source of the rules used in each part.

First module includes the removal of incorrect parse trees. These incorrect parse trees result from being unable to include the context-free grammar rules that prevent these parse trees. The source of the rules of this module is the English grammar rules or recommendations.

Second module includes the addition, modification or deletion of the parse trees based on the nature of the words used in the sentence. This includes the lexical items that consist of two or more words, such as phrasal verbs or compound pronouns, and checking the syntax structure of some words.

Third module includes the selection of some parse trees and removal of the rest based on statistical techniques. This part is responsible for the selection of parse tree structures that are more likely to be mentioned in the text.

### 4.3.1.1. Syntax Analysis Based on Grammar Rules

Along with the correct parse trees produced by the parse for a sentence, the parser output may also include wrong or awkward parse trees. This can be due to several reasons some of which may later be discussed.

The grammar rules are covered using context free grammar (CFG), which is the dominant form of grammar rules used in NLP. So covering some restrictions in the natural language grammar may increase the number of CFG rules or lead to usage of complex notations which make parsing more difficult. Using some checking rules that can be applied after parsing can solve this problem.

Here is an example for a situation where this problem may occur and how it can be solved using simple rules after parsing. Assume we have the following piece of CFG rules as our parsing grammar:

XS 🡪 ADVP + CMA + SS  
ADVP 🡪 ADV  
ADVP 🡪 PRP  
ADVP 🡪 INPH

where these notations have the following meanings:

XS: complex sentence  
ADVP: adverb phrase  
CMA: comma  
SS: simple sentence  
ADV: adverb  
PRP: prepositional phrase  
INPH: infinitive phrase

It is simply noted that this simple grammar can produce wrong parse trees in spite of the definition of ADVP is correct and the definition of XS is also correct. The source of error is that simple sentences can't be preceded by infinitive phrase. A solution using CFG is to produce two versions of ADVP, the original version and a restricted version to come before the simple sentence. Applying this to a complete English CFG can duplicate its size tens of times which will result in an inefficient parsing. Another solution is the introduction of new notations in the grammar that prevent occurrence of this situation so that the intended grammar can be as follows:

XS 🡪 ( ADVP – INPH ) + CMA + SS  
ADVP 🡪 ADV  
ADVP 🡪 PRP  
ADVP 🡪 INPH

This solution is considered better than the first solution but using additional notations makes parsing a more complex process.

Let's look at the following solution for this problem. No more CFG rules are used, no more notations are used. Wrong parse trees are generated, so a checking module is run on the parse trees after parsing, if the wrong situation is found, the parse tree is deleted. Next processing is performed on the rest of the parse trees.

Using this simple solution need no modification on the parsing algorithm and is the most efficient solution of the three introduced solutions. Implementation of this approach can be simply done by using a hash table containing these restrictions and a small module to traverse the parse tree searching for the occurrence of these situations.

Not only grammatical restrictions can be achieved using this approach, but also grammatical recommendations can be applied and bad parse trees that contains specific features can be deleted if a better parse tree without these features is available

### 4.3.1.2. Syntax Analysis Based on Words

This part differs from the first part in that the source of modifications is not grammatical rules, but the words of the sentence or their lexical entry. It means that it depends mainly on the lexicon and the sentence itself.

Let's look at the following sentences:

*John looked up the word in the dictionary*

*He reached up to there*

It is obvious that the verb of the first sentence is *"looked up"* not "*looked"* and that the preposition *"up to"* in the second is a single preposition not two prepositions. Most parsers treat with this condition correctly by searching for compound lexical items in the sentence and treating them each as one word. This produces two parse trees for each of the two mentioned sentences.

In certain situations, the parser can't deal with compound words and let's take an example which illustrates this situation.

*John looked the word up in the dictionary*

*John looked the word in the dictionary up*

It is impossible for the parser to know that the intended verb in these two sentences is *look up* not *look*. Here comes the role of the syntax analyzer to solve this problem. Syntactic issues like this need traversal of the parse trees searching for some predefined patterns and doing the intended action for every found pattern (see Figure 4.6).

*John looked the word up in the dictionary*

Figure 4.6: Two parse trees for two different sentences

*John looked the word in the dictionary up*

Let's take a look at simple parse trees for the two mentioned sentences in the previous figure.

*John looked up the word in the dictionary*

Figure 4.7: The modified parse tree

The syntax analyzer is intended to provide a modification for these parse trees or addition of a new parse tree so as to let "*look up"* be the sentence verb instead of "*look*" being the verb and "*up"* being an adjective. The parse tree introduced by the syntax analyzer should be something like the one in Figure 4.7.

Another role of syntax analyzer is to delete some parse trees that don't cope with the syntax structure of the words in the sentence. Not all lexicons contain a syntax structure entry for their words. If the syntax structure entry is available, it can be used to check parse trees for the syntax structure.

The following examples illustrate the mentioned idea:

1. The teacher made the students interested in the subject.  
 2. The teacher made the students learn well.  
 3. The teacher went to the class.  
 4. \* The teacher made  
 5. \* The teacher went the garden

The first three sentences are completely correct sentences, but the two sentences preceded by \* are wrong. The verb *made* in the fourth sentence can't come in the intransitive form, so this sentence is wrong. The verb *went* in the fifth sentence must be followed by a prepositional phrase.

### 4.3.1.3. Syntax Analysis Based on Statistics

The third part of the syntax analyzer may be included in a NLP system or not. Its main functionality is the selection of one or more of the parse trees depending on the shapes of the available parse trees. It is some sort of evaluating the parse tree based on its structure and contained nodes.

Several ideas can be used to achieve this idea:

* Each possible shape of the parse tree can have a predetermined probability. Parse trees with big probabilities are the parse trees that survive and parse trees with smaller probabilities are deleted.
* Finding certain constructs in the parse tree increase or decrease the parse tree chance of surviving.
* Choosing parse trees that minimizes or maximizes certain criteria.

Let's take a look at the following example:

*This is the biggest garden of the gardens of Cairo.*

The prepositional phrase *"of Cairo"* acts as an adjective. It can be describing the noun "*gardens"* or the noun "*garden"*. It is obvious that the correct interpretation of this sentence entails choosing that "*of Cairo"* describes "*gardens*" not "*garden"*. A syntactic rule could be added that prefers the usage of the good parse tree and deleting the bad parse tree.

Another example is introduced to assure good understanding of the idea:

*I prepared to play in the garden*

The prepositional phrase "*in the garden*" can be a complement for the verb *"prepared*" or the verb "*play"*. It's obvious that it is more correct to treat *"in the garden"* as a complement for "*play"*. Adding a rule that prefers complements to modify last mentioned verb does the intended action.

# Discourse Analysis

Discourse Analysis is concerned with assigning each pronoun to the noun which this pronoun refers to.

## 4.4.1. “Resolution of Anaphora Procedure” Algorithm

It is an algorithm for identifying both intra-sentential and inter-sentential antecedents of pronouns in text. RAP applies to the syntactic structures of McCord's (1990, 1993) Slot Grammar parser. It relies on measures of salience derived from syntactic structure and a simple dynamic model of attentional state to select the antecedent noun phrase (NP) of a pronoun from a list of candidates. It does not employ semantic conditions or real-world knowledge in evaluating candidate antecedents; nor does it model intentional or global discourse structure.

## RAP main components

RAP contains the following main components:

* An intra-sentential syntactic filter for ruling out anaphoric dependence of a pronoun on an NP on syntactic grounds.
* A morphological filter for ruling out anaphoric dependence of a pronoun on an NP due to non-agreement of person, number, or gender features.
* A procedure for identifying pleonastic (non-anaphoric, semantically empty) pronouns.
* An anaphor binding algorithm for identifying the possible antecedent binder of a lexical anaphor within the same sentence.
* A procedure for assigning values to several salience parameters (grammatical role, parallelism of grammatical roles, and frequency of mention, proximity, and sentence recency) for an NP. Higher salience weights are assigned to (i) subject over non-subject NPs, (ii) direct objects over other complements, (iii) arguments of a verb over adjuncts and objects of prepositional phrase adjuncts of the verb, and (iv) head nouns over complements of head nouns.
* A procedure for identifying anaphorically linked NPs as an equivalence class for which a global salience value is computed as the sum of the salient values of its elements.
* A decision procedure for selecting the preferred element of a list of antecedent candidates for a pronoun.

## Types of Anaphora

Anaphora is cohesion (presupposition) which points back to some previous item. There are various types of anaphora.

**Pronominal anaphora:** The most widespread type of anaphora which is realized by anaphoric pronouns.  
Example: “They took extensive notes”.  
It should be pointed out that not all pronouns in English are anaphoric. For instance, "it" can often be non-anaphoric such as in the case of the previous sentence. Other examples of non-anaphoric “it” include expressions such as "It is important", "It is necessary", "It has to be taken into account". A non-anaphoric "it" is termed pleonastic.

**Definite noun phrase anaphora:** Typical cases of definite noun phrase anaphora is when the antecedent is referred by a definite noun phrase representing either same concept (repetition) or semantically close concepts (e.g. synonyms, superordinates).  
Example: The participants found it hard to cope with the speed of the presentation.

**One-anaphora:** One-anaphora is the case when the anaphoric expression is realized by a "one" noun phrase.  
Example: If you cannot attend a tutorial in the morning, you can go for an afternoon one.  
Finally, we distinguish *intra-sentential* anaphors (referring to an antecedent which is in the same sentence as the anaphor) and *inter-sentential* anaphors (referring to an antecedent which is in a different sentence from that of the anaphor).

### 4.4.1.3. Example of the RAP’s output

John talked to Bill about himself.

Antecedent NP--lexical anaphor pairs: John 1.1 (1.1: "John" is the first word in the first sentence) – himself 1.6 (1.6: "himself" is the sixth word in the first sentence), Bill 1.4 – himself 1.6

Anaphor--Antecedent links: himself. (1.6) to John. (1.1)

In this example, john. (1.1) was preferred to Bi11.(1.4) owing to its higher salience weight.

# Word Sense Disambiguation

Word Sense Disambiguation is concerned with assigning the most proper sense for each word according to the formulation of the sentence. For example, the word "ball" has several senses, including a round object used in games, a formal dance, and a pitch in baseball that is not a strike. A precise method for quantifying how similar two word senses are is called a measure of semantic relatedness.

4.5.1. WordNet  
  
WordNet is a machine readable dictionary created at the Cognitive Science Laboratory at Princeton University. Unlike most dictionaries, WordNet contains only open-class words (nouns, verbs, adjectives, and adverbs). WordNet does not contain closed-class words such as pronouns, conjunctions, and prepositions. WordNet groups sets of synonymous word senses into synonym sets or synsets.

A word sense is a particular meaning of a word. For example, the word *"walk*" has several meanings; as a noun, it can refer to traveling by foot or it can refer to a base on balls in baseball. A synset contains one or more synonymous word senses. For example, base on balls, walk, pass is the synset for the second sense of the noun walk. The synset is the basic organizational unit in WordNet.

Each synset has a gloss (definition) associated with it. The gloss for the synset base on balls, walk, pass is “(baseball) an advance to first base by a batter who receives four balls.” Many synsets also have an example in addition to the gloss. For example, “he worked the pitcher for a base on balls.”

The sense numbers in WordNet are assigned according to the frequency with which the word sense occurs in the SemCor corpus (i.e., the first sense of a word is usually more common than the second). Word senses that do not appear in SemCor are assigned sense numbers in a random order. Word senses can be represented as strings in a specific format, using the word form, a single letter representing the part of speech, and a sense number, such as walk#n#2, which represents the second sense of the noun walk. The part of speech letter is "n" for nouns, "v" for verbs, "a" for adjectives, and "r" for adverbs. Terms consisting of more than one word are often joined by underscores instead of spaces. The synset in the previous paragraph can be written as base on balls#n#1, walk#n#2, pass#n#1.

WordNet defines relations between synsets and relations between word senses. A relation between synsets is a semantic relation, and a relation between word senses is a lexical relation. The distinction between lexical relations and semantic relations is somewhat subtle. The difference is that lexical relations are relations between members of two different synsets, but semantic relations are relations between two whole synsets. Some examples of semantic relations are the hypernym, hyponym, meronym, and holonym relations. A hypernym of a synset is a generalization of that synset. The hyponym relation is the inverse of the hypernym relation. The hypernym and hyponym relations represent is-a relationship between nouns. If A and B are nouns, and an A is a B, then A is a hyponym of B and B is a hypernym of A. For example, {organism, being} is the hypernym of {plant, flora} because a plant is an organism. Since the hyponym relation is the opposite of the hypernym relation, {plant, flora} is a hyponym of {organism, being}. For verbs, the term troponym is used instead of hyponym. A troponym is a way of doing something else. For example, soar is a troponym of fly, wing because soaring is a way of flying. WordNet still uses the term hypernym as the inverse of tryponym; therefore, {fly, wing} is a hyponym of {soar}.

{travel, go, move, locomote}

{walk}

{fly, wing}

{ride}

{soar}

{hover}

Figure 4.8: WordNet troponyms

A meronym is a word that is a part of whole (e.g., {wheel} is a meronym of {wheeled vehicle}). The inverse is the holonym relation (e.g., {wheeled vehicle} is a holonym of {wheel}).

The hypernym, hyponym, and troponym relations are particularly interesting. These relations form is-a taxonomies with the noun and verb synsets. These taxonomies have a tree-like structure. All noun and verb synsets belong to at least one taxonomy. Because multiple inheritance is allowed, some synsets belong to more than one taxonomy. There is no unique root node that links all noun synsets together or all verb synsets together. Instead, there are multiple taxonomies. Some examples of lexical relations are the antonym relation and the derived form relation. For example, the antonym of the tenth sense of the noun light (light#n#10) in WordNet is the first sense of the noun dark (dark#n#1). The synset to which light#n#10 belongs is {light#n#10, lighting#n#1}. Clearly it makes sense that light#n#10 is an antonym of dark#n#1, but lighting#n#1 is not a antonym of dark#n#1; therefore, the antonym relation needs to be a lexical relation, not a semantic relation because the relationship is between individual word senses, not whole synsets.

* **Measures of Relatedness**

The relatedness between two words can be computed using any of the following three measures:

1. **Path measure**: the distance between the words' synsets in the is-a tree in the WordNet.
2. **Information content**: Information content is a measure of specificity. The information content of a concept is inversely related to the frequency with which the concept is expected to occur. A concept that rarely occurs would have a high information content, and a concept that frequently occurs would have a low information content. Mathematically, the information content of a concept is IC(c) = -log P(c) where P(c) is the probability of the concept c.
3. **Glosses** : gloss overlaps are a very promising means of measuring relatedness, since they can be used to make comparisons between concepts of different parts of speech. For example, this might include comparing nouns with verbs, or verbs with adjectives. Measures that are based on paths in *is–a* hierarchies tend to be limited to making comparisons between concepts with the same part of speech, since these hierarchies do not include multiple parts of speech. The only other measure capable of mixed part of speech comparisons is that of Hirst and St. Onge, which is dependent on the existence of specific links between concepts.

## 4.5.2. Lesk algorithm

The Lesk algorithm is an example of the glosses relatedness measure. The original Lesk algorithm disambiguates words in short phrases. Given a word to disambiguate, the dictionary definition or gloss of each of its senses is compared to the glosses of every other word in the phrase. A word is assigned that sense whose gloss shares the largest number of words in common with the glosses of the other words. The algorithm begins anew for each word and does not utilize the senses it previously assigned.

The Lesk demonstrates this algorithm on the words *pine cone* using the Oxford Advanced Learner’s Dictionary.

It finds that the word *"pine"* has two senses:

Sense 1: kind of **evergreen tree** with needle–shaped leaves

Sense 2: waste away through sorrow or illness.

The word *"cone"* has three senses:

Sense 1: solid body which narrows to a point

Sense 2: something of this shape whether solid or hollow

Sense 3: fruit of certain **evergreen tree**

Each of the two senses of the word "*pine"* is compared with each of the three senses of the word "*cone"* and it is found that the words *"evergreen tree"* occurs in one sense each of the two words. These two senses are then declared to be the most appropriate senses when the words "*pine"* and "*cone"* are used together.

Similarly the algorithm has been shown to correctly disambiguate the word "*flies"* in both *"time flies like an arrow"* and "*fruit flies like a banana"*. Given a phrase like "*time flies like an arrow",* the algorithm compares the glosses of all the sense of *"time"* to the glosses of all the senses of *"fly"* and "*arrow"*, and assigns to "*time"* that sense that leads to the greatest number of overlapped words. Next it compares the glosses of *"fly"* with those of *"time"* and "*arrow*", and so on.

However, it's recognizable that glosses are by necessity short, and may not provide sufficient information on their own to make judgments about relatedness. For example, the gloss of "*canoe"* is "*small and light boat pointed at both* *ends propelled with a paddle"*. It has no gloss overlaps with either *bank*1: *financial institution that accepts deposits and channels the money into lending* *activities*, or with *bank*2: *sloping land especially beside a body of water*. Thus in the sentence *"The canoe was near the bank*", a simple gloss overlap measure finds no relation between *canoe* and either sense of *bank*.

Two different measures have been developed to address this issue. In the extended gloss overlap measure, we also make comparisons between glosses of words that are related according to WordNet. In the gloss vector measure simplify ideas from both Wilks, et. al. and Sch¨utze to create a relatedness measure based on dictionary gloss co–occurrence statistics.

**Extended Gloss Overlaps (Adapted Lesk)**

The extended gloss overlap measure was developed to overcome the limitations of short definitions. Lesk’s gloss overlaps are adapted to a networked resource such as WordNet by finding overlaps not only between the definitions of the two concepts being measured, but also among those concepts to which they are related.

This is motivated by the idea that semantic relations (such as *is–a* and *has–part*) specified in WordNet do not capture all the possible relations betweenconcepts. For example, there are no explicit relations between *boat: a small vessel for travel on water*, and *bank*2*: sloping land especially beside a body of water*. Even the shortest *is-a* path between them in WordNet is not particularlyindicative of their relatedness, since it includes the higher level concept *physical object*.

However, despite the lack of a path in WordNet (direct or indirect) we observe that *boat* and *bank*2 are related. One can launch a boat from a bank, for example, or run a boat aground on a bank. The glosses of these two concepts share the word *water* which hints at their relatedness. The fact that concepts to which each of these is related also share overlaps adds to that conclusion. Thus, in general we believe that there are relations between concepts that are implicit but can be found via gloss overlaps.

For the extended gloss overlap measure, we consider the glosses of all the concepts that are directly connected to a concept by a relation when finding overlaps. The process of finding and scoring overlaps can be described as follows: When comparing two glosses, we define an overlap between them to be the longest sequence of one or more consecutive words that occurs in both glosses such that neither the first nor the last word is a function word, that is a pronoun, preposition, article or conjunction. If two or more such overlaps have the same longest length, then the overlap that occurs earliest in the first string being compared is reported. Given two strings, the longest overlap between them is detected, removed and in its place a unique marker is placed in each of the two input strings. The two strings thus obtained are then again checked for overlaps, and this process continues until there are no longer any overlaps between them. The sizes of the overlaps thus found are squared and added together to arrive at the score for the given pair of glosses.

The original Lesk Algorithm compares the glosses of a pair of concepts and computes a score by counting the number of words that are shared between them. This scoring mechanism does not differentiate between single word and phrasal overlaps and effectively treats each gloss as a bag of words. For example, it assigns a score of 3 to *bank*2*: (sloping land especially beside a body of* *water)* and *lake: (body of water surrounded by land)*, since there are 3 overlapping words: *land, body, water*. Note that stop words are removed, so *of* is not considered an overlap.

However, there is a relationship between the lengths of phrases and their frequencies in a large corpus of text. The longer the phrase, the less likely it is to occur multiple times in a given corpus. A phrasal *n*–word overlap is a much rarer occurrence than a single word overlap. Therefore, we assign an *n* word overlap the score of *n*2. This gives an *n*–word overlap a score that is greater than the sum of the scores assigned to those *n* words if they had occurred in two or more phrases, each less than *n* words long. This is true since the square of a sum of positive integers is strictly greater than the sum of their squares. That is, (*a*0 + *a*1 + *...* + *an*) 2 *>* (*a*0) 2 + (*a*1)2 + *...* + (*a n*) 2, where *ai* is a positive integer. For the above gloss pair, we assign the overlap *land* a score of 1 and *body of water* a score of 9, leading to a total score of 10.

**Path length similarity measure**

The Path measure in WordNet::Similarity is a simple measure that uses the path length distance to measure the similarity of synsets. Within the package it is called WordNet::Similarity::path. The distance between two synsets is measured using node counting. Similarity is defined as

Simpath(s1; s2) = 1/distnode(s1; s2) (1)

such that distnode(s1; s2) is the distance between synset s1 and synset s2 using node counting. The distance between {person} and {living thing, animal} is three, so the similarity score is 1=3.

Since node counting is used, the distance between two synsets is always greater-than or equal-to 1. For example, the distance between {person} and {person} is 1. Therefore, similarity is always greater-than 0 but less-than or equal-to 1.

If a unique root node is being used, then there will always exist a path between any two noun synsets or any two verb synsets. If, however, a unique root node is not being used, then it is possible and in the case of verbs likely that there will not be a path between two synsets. In such a case, the similarity of the two synsets is undefined.

Since some synsets in WordNet have more than one hypernym, all measures of semantic similarity must take multiple inheritance into account. In the case of this measure, when there is more than one path between two synsets as a result of multiple inheritance, the shortest path is used.

One perceived drawback of a simple edge or node counting measure is that links in a taxonomy like WordNet can represent different distances between synsets. Some links may represent a large difference in meaning, while other links may represent only a small refinement in meaning. Typically, links that are high in taxonomy (closer to the root), represent a greater semantic distance, and links low in taxonomy represent a smaller semantic distance. There is a greater semantic distance {object, physical object} and {land, dry land, earth} than there is between {island} and {land, dry land, earth}.

Chapter (5)  
Text Meaning Representation

Meaning Representation is part of Semantic Analysis concerned with extracting the meaning from the input text. The semantic analyzer performs three sub-tasks; Word Sense Disambiguation, Discourse Analysis and Text Meaning Representation.

1. **Word Sense Disambiguation** (WSD): This task is concerned with assigning the most proper sense (meaning) for each word according to the formulation of the sentence.
2. **Discourse analysis**: This task is basically concerned with assigning each pronoun to the noun which this pronoun refers to.
3. **Meaning Representation**: putting the text in a form which represents its meaning and simply querying the text.

The chapter discusses the meaning representation. This sub-task is responsible for converting the text into a form that represents its meaning.

# Meaning Representation

Designing a meaning representation for NLP involves determining its content and its representation. We address the issue of content in terms of the nature of primitives used to share knowledge between linguistic and world knowledge representations. The structure of the representation is shown to be based on the needs for composing the primitives in different ways, the expressiveness of which is determined by the machine translation task and its needs for linguistic and world knowledge.

# Text Meaning Representation

A text meaning representation (TMR) is made up of particular instances of meanings represented in the ontology and the lexicon. To distinguish between concepts and instances, we introduce an *instantiation operator* that works as follows. An instance has all the properties that its concept has either in the ontology or in the lexicon for which a value is available. Instances only have value facets in their properties. Values are found as follows: any value represented in the lexicon supersedes any value in the ontology. If no value is specified in either the lexicon or the ontology, a default filler from the ontology is the value for the instance.

In addition, the following enhancements are made to MR to get the TMR language:

* Properties in TMRs often need to refer to chunks bigger than individual instances in the TMR. For example, in ``John said that Bill's obsession with guns sent him to prison,'' what was said was more than a single object or event. In order to support such scoping needs, we introduce a ``super structure'' called *proposition.* A proposition has a head that must be a concept. In addition, it has a limited set of properties including time, aspect, and attitude. Propositions group word meanings into bigger structural units to represent meanings of entire sentences. A proposition is to TMR what a sentence is to a text.
* An instance often needs to refer to a particular slot of another instance. We introduce a *reification* operator that raises a slot in an instance to the TMR level by making it an individual instance in the TMR.
* In order to capture co-references in a text, we introduce a *co-reference* relation that identifies two or more instances in the TMR as referring to one and the same entity in the world.
* Certain other linguistic embellishments are also added to the TMR. For example, *time relations* and *quantitative relations* are used to represent the corresponding information.

# Introduction to Text Meaning Representation

A text meaning representation (TMR) is a language-neutral description of the meaning conveyed in a natural language text, and is derived by syntactic, semantic, and pragmatic analysis of the text. Because the TMR is intended to be language neutral, it is also deliberately syntax neutral, and avoids using terminology like clause, proposition, tense, etc., which are associated more closely with the syntactic structure of a particular language. In addition to providing information about the lexical-semantic dependencies in the text, the TMR represents stylistic factors, discourse relations, speaker attitudes, and other pragmatic factors present in the discourse structure. In doing so, the TMR captures not only the meaning of individual elements in the text, but also the relations between those elements, and captures both propositional and non-propositional components of textual meaning.

## Text Meaning Representation Structure

The TMR is divided into seven sections which combine to convey the overall meaning of the original text. The Table of Contents provides a summary of the heads (roughly, the predications), speech acts, attitudes, relations, focus, and stylistic factors found in the text, and is followed by a Speech Act section where the type and scope of each speech act, the speaker/writer, hearer/reader, time of the speaking/writing, etc. are given.

The results of analyzing the individual sentences in a natural language text are represented in the TMR body. A clause in the natural language is typically represented by an EVENT or PROPERTY concept from the ontology; this concept is referred to as an interlingual head in the TMR, and contains a number of modifying roles (such as case and circumstantial roles) that further define it.

The TMR also contains sections in which information on Domain Relations, and Temporal Relations are conveyed. In the final section of the TMR, the co-reference section, separate occurrences of the same object or event are matched up.

## Components of Text Meaning Representation

These seven sections, previously mentioned, combine to convey the overall meaning of the original text. The first section of a TMR is a "table of contents" which, in practice, is the last section to be filled in. The table of contents provides a summary of the predicates, relations, and stylistic factors found in the text. This section is followed by a "statement" section where the scope of the text, the speaker/sriter, hearer/reader, time of the speaking/writing, etc. are given. Next comes the "TMR body", where sentences in a natural language text are represented a language-neutral format. The text is translated, generally clause by clause, from the original natural language into the interlingua. A clause typically equates to an interlingual head, which can be an event, a property, an attribute, or an object concept. Most heads have agents (subjects) and themes (objects). Information about the head is given in a slot-filler format. Heads can have other slots (e.g. COTHEME, ACCOMPANIER, BENEFICIARY, PURPOSE, MANNER, ATTITUDE, LOCATION, FOCUS, etc.), as needed to convey the meaning of the original text. Next, the "temporal relations" section documents of a temporal nature between clauses. This is followed by the domain relations section, where relations are made between syntactic elements. The final section of the TMR is the "co-reference section". Here separate references in the TMR body to the same object or event are matched.

# Illustrating the translation from Natural Language to Text Meaning Representation

It is easiest to beginbyfirst separating the sentences of your text into clauses. A clause, generally speaking, is a subject and finite verb (a verb which changes tense with the subject). So there may be one, or several, clauses in a sentence.

Let's look at an example. Consider the following text:

"Kawasaki Steel Corp. has reached agreement with Mitsubishi Corp. and CIA Vale Do Rio Doce, Brazil's state-run mining corporation, to set up a joint venture in Brazil possibly this summer to produce ferrosilicon, reported Nikkan Kogyo."

To separate this text into clauses, first highlight or pull out the verbs or verb phrases of each sentence:

Kawasaki Steel Corp. **has reached agreement** with Mitsubishi Corp. and CIA Vale Do Rio Doce, Brazil's state-run mining corporation, **to set up** a joint venture in Brazil possibly this summer**to produce** ferrosilicon, **reported** Nikkan Kogyo.

so we have our clauses:

* Kawasaki Steel Corp. **has reached agreement** with Mitsubishi Corp. and CIA Vale Do Rio Doce, Brazil's state-run mining corporation
* **to set up** a joint venture in Brazil possibly this summer
* **to produce** ferrosilicon
* **reported** Nikkan Kogyo

Notice that although this text is only one sentence long, there are at least four clauses in it. Also notice that when you have a verb, a subject is implicit, even if it is not mentioned explicitly in the text (an action infers someone or something is *doing* the action). It can be overwhelming to attempt to transcribe an entire sentence, so it is important at this stage to only transcribe one clause at a time.

***First clause:***

**Selecting the Head**

A verb is a common head in a TMR frame. In the case of our example, the verb phrase ***has reached agreement*** presents itself as a possible head. Consequently, ***agree*** is the verb we should use as the head of the current clause.

**Filling the Slots for Case Roles**  
The next step to representing the first clause is to fill the **case roles**, which are the arguments a predicate can take. These include:

* agent
* theme
* co-theme
* accompanier
* beneficiary
* experience
* instrument
* source
* destination
* path
* degree
* means
* manner
* purpose

In addition, the circumstantial roles must be filled in if they are appropriate to the clause. These are roles which relate events to more circumstantial pieces of information that describe them, such as:

* location
* time

But what about the clause we're working on? To refresh your memory, here it is again:

"Kawasaki Steel Corp. has reached agreement [our head: agree] with Mitsubishi Corp. and CIA Vale Do Rio Doce, Brazil's state-run mining corporation"

Going down the list of available case roles, let's begin by seeing if agree has an **agent.**

**Agent**  
To find out if agree has an agent, ask "who or what is agreeing?" The answer to that in this case is Kawasaki Steel Corp., Mitsubishi Corp. and CIA Vale Do Rio Doce. So we have three "agents" of agree.

If only things were that simple! But there is more to the text than that. Note that in the original clause, "Kawasaki Steel Corp." is given special mention; it has "reached agreement" with the other two companies, which are placed in a clause-end prepositional phrase. So, in fact, "Kawasaki Steel Corp." is the "agent," while the other two companies fill the case slot **accompanier**. The Accompanier slot in this way conveys the idea that the emphasis in this clause is placed on Kawasaki Steel Corp.

**Theme**  
A very common pairing of case roles finds **agent** and **theme** occurring together.

To discover if agree requires a **theme** case role, ask yourself: what does the action of agreeing affect? Often a **theme** will be an entity of some kind, but in this case, agree affects the next clause, that headed by create:

Kawasaki Steel Corp. has reached agreement **[agreed]** [...] **to set up** a joint venture [...]

#### So it is the action of setting up which is affected by agree. This is a very common use of the **theme** slot--filling it with an action that is "nested" within another action

There are no other case roles which need to be invoked for the first clause, but there are circumstantial roles which are part of every predicate frame.

**Property Frames** *There is one portion of the text in the first clause which has been left wholly unrepresented:*

*Kawasaki Steel Corp. has reached agreement with Mitsubishi Corp. and CIA Vale Do Rio Doce,****Brazil's state-run mining corporation****[...]*

*The only "verb-like" word in this text segment is*state-run*; one option might be to decompose the word into the verb*run*. However, there is a better way to represent this text than to decompose the verb-like phrase into a head of its own.*

*Because*state-run*and*mining*are properties, they more appropriately belong to a****property frame****. A property frame is headed by an object, and has special slots suited to describing common object properties.*

***Second clause:***

*We initially divided the second clause as:*

*...to set up a joint venture in Brazil possibly this summer...*

*However, if we look ahead to the next clause:*

*...to produce ferrosilicon...*

*we may want to ask ourselves, is*possibly this summer*meant to modify*set up a joint venture*or*to produce ferrosilicon*? One may make arguments for either choice--welcome to the wonderful world of text meaning representation!*

*We will take the position that it is the*setting up*of the joint venture which will take place*possibly this summer*instead of the producing of ferrosilicon.*

**Filling the slots for Case Roles**

*Because we needed to put the head of the second clause in as filler for the****theme****slot of the previous clause, we already know that the head of the second clause is*set up*.*

*It is only necessary to look again at the definitions and examples of each to determine which case roles are needed.*

***Agent*** *Although the****agent****doesn't occur in the second clause, it can be traced back to the first:*

*First Clause*

*Kawasaki Steel Corp. has reached agreement with Mitsubishi Corp. and CIA Vale Do Rio Doce, Brazil's state-run mining corporation...*

*Second Clause*

*...to set up a joint venture in Brazil...*

*Asking, "who or what is*setting up*something?", we get the answer--Kawasaki Steel Corp., Mitsubishi Corp. and CIA Vale Do Rio Doce.*

***Theme*** *The entity whose state is affected by the action of*setting-up*is clearly*joint venture*.*

*However, there is more to the "joint venture issue" than meets the eye. Since*setting-up a joint venture*can be glossed as*creating a tie-up*, we can double back and change the head to*create*. (This also necessitates changing the****theme****of the first clause to*create*, which is easily done.)*

***Finishing the Predicate Frame: Circumstantial Roles*** *Are there any other roles which should be represented in this clause of the TMR? Look at the circumstantial roles and their definitions to discover if there are any appropriate to represent this clause.*

**Location** *The circumstantial role****location****probably caught your eye; a prepositional phrase such as "*in Brazil"*is a giveaway that the****location****slot is needed*

**Purpose** *To find out if there are additional roles, it is often necessary to look ahead to the clauses which follow. They are:*

*Third Clause*

***to produce****ferrosilicon*

*Fourth Clause*

***reported****Nikkan Kogyo*

*It is clear from the text that the****purpose****of*creating the tie-up*is in the third clause: to*produce ferrosilison*; in fact, the insertion of the words "in order to" make this relation clear--*to set-up [create] a joint venture [tie-up] in Brazil possibly this summer [in order to] produce ferrosilicon*. So the****purpose****slot should be added, with the head of the next clause as its filler.*

***Third clause:***

*Because we needed to put the head of the second clause in as filler for the****theme****slot of the previous clause, we already know what the head of the second clause is:*

*to produce ferrosilicon*

*Although it may seem that the****agent****of produce would be the same set as in the previous clause, a closer look is in order. The previous clause, joined to our third clause, yields:*

*to set up****[create]****a joint venture [tie-up]  
possibly this summer to****produce****ferrosilicon*

*So we see that it is the newly-created joint venture which will produce ferrosilicon. We must instantiate a new "tie-up," and co-reference.*

**Theme** *This time, instead of filling the****theme****slot with another head, we will fill it with ferrosilicon*

***Fourth clause:***

*The final clause is a speech act clause which is reporting what proceeded in the sentence:*

*Kawasaki Steel Corp. has reached agreement with Mitsubishi Corp. and CIA Vale Do Rio Doce, Brazil's state-run mining corporation, to set up a joint venture in Brazil possibly this summer to produce ferrosilicon,****reported Nikkan Kogyo****.*

*This is similar to the speech acts one finds in novels ("I told you not to come,"****said Pierre****.") and even in everyday conversation (****My dad said****he's going to to kill me if I wreck the car again.). Of course, they are very often found in newspaper articles, which is where our sentence comes from.*

*The****theme****, in this case, is the first predicate of the sentence; imagine that the fourth clause was at the beginning of the sentence--Nikkan Kogyo reported that Kawasaki Steel Corp. has reached agreement with... So we insert into the****theme****slot the first head of the sentence, agree.*

# Case Role and Circumstantial Role Definitions and Examples

A tentative list of case roles and circumstantial roles for use in TMR writing. Definitions and examples are given to provide a common starting point. Both case roles and circumstantial roles would be defined as properties (specifically, as relations) in the ontology.

**Case Roles**Case Roles are the arguments or typical roles that a predicate can take; they will appear as properties of events in the TMR.  
The following is a list of the case roles used in the text meaning representation (TMR).

* Agent
* Theme
* Experiencer
* Beneficiary
* Source
* Destination
* Location
* Path
* Co-theme
* Accompanier
* Purpose
* Degree
* Means
* Manner

The following is a brief explanation and example for each of the case roles just mentioned above.

**Agent** - the entity that causes or is responsible for an action. (the subject in a transitive sentence is often, but not always, the agent)

* Kathy [agent] ran to the store [destination].
* Joey [agent] ate the cookie [theme].
* Du Pont Co. [agent] said it agreed to form a joint venture in gas separation technology with L'Air Liquide S.A., an industrial gas company based in Paris.

**Theme** - the entity whose state or location is being described, or whose state is affected by an action (direct object of an action; subject in an intransitive sentence).

* John [agent] kicked the ball [theme].
* The price [theme] is high.
* The ball [theme] rolled down the hill [path].
* Bridgestone Sports Co. has set up a company [theme] in Taiwan with a local concern and a Japanese trading house.

**Experiencer** – the entity that undergoes psychological experience (perception, cognition); (e.g., passive role in involuntary perceptual event)

* I [experiencer] realized that . . .
* John [experiencer] heard the music [theme]. (vs. John [agent] listened to the music [theme].
* However, recently the natural rubber market [experiencer] has experienced a slump and is at 60 percent of production capacity.

**Beneficiary** - the entity that benefits from an action

* I [agent] did it [theme] for Mary [beneficiary].
* Cindy [agent] lent me [beneficiary? or destination?] some money [theme].

The venture, which is contingent on a definitive agreement, intends to develop, manufacture and market equipment based on polymeric membrane technology and in some instances perform the gas separation for customers [beneficiary].

Issue: Should beneficiary be used only as a "circumstantial" role, leaving source and destination to handle verbs of transfer (give to, buy from, etc.). See more examples under 5 and 6 below.

**Source** - conceptual places where various types of movement and transfer start (used for direction in verbs of motion)

* The goods [theme] will be shipped from Japan [source].
* Susan [agent] bought the book from Jane [source].
* BROKEN HILL PROPRIETARY CO., Australia's biggest company, said it soon will receive 133 million Australian dollars ($94.7 million) from the sale [source] of domestic assets.

**Destination** - an endpoint for actions & processes involving change of location, transfer.

* John [agent] took his mother [theme] to the theater [destination].
* Jill [agent] went to the store [destination].
* Cindy [agent] brought the money to me [destination]
* Hilda [agent] gave John [destination] a present [theme].
* Bridgestone Sports Co. said Friday it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan [destination].

Issue: For verbs of transfer (buy, sell, give, take, exchange, lend, borrow, etc.), other roles are sometimes used, such as beneficiary, or recipient. I suggest that source/destination can be used to cover verbs of transfer as well as verbs of motion. Note that "Hilda" may also be regarded as a source, as well as an agent. I prefer to ignore this conflation, but it may be relevant for languages other than English.

**Location** - the place where an event takes place or where an object exists

* The milk [theme] is in the refrigerator [location].
* The play [theme] will be performed at the Shakespeare Theater [location].
* Kawasaki Steel intends to secure a stable supply of ferrosilicon by producing the material in Brazil [location] where power rate is less than one-third of that of Japan [location].

Note: Location can be both a case role and a circumstantial role, in that any event can take place at a location and at a time. However, some predicates require that a location be expressed, as in the first example above.

**Path** - the route along which an entity (i.e., a theme) travels

* Mary [agent/theme] ran down the hill [path].
* The train [theme] traveled along the track [path].
* The plane [theme] took the polar route [path] from Korea [source] to Chicago [destination].

Issue: Role conflation: "Mary" is both the agent and the theme of "ran".

**Co-theme** - an entity whose state is described in relation to another

* John [theme] is a lawyer [co-theme].

Co-theme is used together with theme to represent events such as: `designate X as Y', `christen X as Y', `name X as Y', `mark X as Y', `sell X as Y', `pronounce X as Y', `dub X as Y', and `elect X as Y'.

**Accompanier** - an entity which joins the agent in the event, but is not the initiator of the event

* John [agent] went with Mary [accompanier].
* Kawasaki Steel Corp. has agreed with Mitsubishi Corp.[accompanier] and Cia Vale Do Rio Doce [accompanier], Brazil's state-run mining corporation, to set up a joint venture in Minas Gerais State in Brazil possibly this summer to produce ferrosilicon, reported Nikkan Kogyo.

**Purpose** - a goal; the reason for which something is done

* Nomura Shoken and Credit 109 [agents] have tied up to issue the "Nomura Tokyu Top Card" [purpose].
* Dresser Industries Inc. and Komatsu Ltd. of Japan said they signed a memorandum of understanding \*to join their construction equipment businesses in North, Central and South America\* [purpose].

**Degree** - the extent to which something occurs or is done.

* The development of joint products [theme] by the nonlife insurance industry and the securities companies [agents] is expanding a step [degree].

**Means** - the method or way in which something is accomplished

* Armco [agent] will establish a new company [theme] by spinning off its general steel department [means].
* The venture, which is contingent on a definitive agreement, intends to develop, manufacture and market equipment based on polymeric membrane technology and in some instances perform the gas separation for customers [beneficiary].

**Manner**- the style in which something is done.

* Fujitsu and Telecom [agents] will jointly [manner] set up Information Switching Technology [theme].

**Circumstantial Roles**  
Roles which relate events to more circumstantial pieces of information that describe them, such as location, time, etc. These roles will also appear as properties of events in the TMR. Here is a tentative list:

**Location** (see above, under case roles, no. 7)

**Time** - the time at which an event takes place

* John [agent] ate dinner at five o'clock [time].
* Bridgestone Sports Co. said Friday [time] it has set up a joint venture in Taiwan with a local concern and a Japanese trading house to produce golf clubs to be shipped to Japan.

Other tentative circumstantial roles that have been suggested in previous work are listed below:

* condition
* cause
* effect
* result
* mode
* quantity
* quality

# Domain Relation Definitions

The following list provides brief definitions of the Domain Relations. We are in the process of revising the definitions and collecting examples of these relations. We hope to provide a more comprehensive working aid in the future.

**Casual Domain Relations**  
Relations of dependence among events, states, and objects. The following list shows the casual domain relations:

* Volitional
* Non-volitional
* Enablement
* Purpose
* Condition
* Reason

Following is an explanation to what each of the relations stand for:

**Volitional** - The relation between a deliberate, intentional action of an intelligent agent, and its consequence

**Non – volitional** - The relation between a non-intentional action or a state of an intelligent agent and its consequence

**Enablement**  - An event enables an event or a state when it removes the obstacles that were preventing the latter from occurring

**Purpose** - Event A is a purpose for event or state B if A describes a goal which an intelligent agent tries to achieve by performing B

**Condition** - Event or state A is a condition for event or state B if A is a cause, reason, enablement or purpose of A and A is an event or a state which has not actually happened and is, thus, hypothetical

**Reason** - The relation between an event or state, and a deliberate, intentional action by an agent. Often (but not always) lexically realized in English through "because," "since" or "for the reason that."

**Conjunction Domain Relations**  
Relations among adjacent elements that are components of a larger textual element. The following shows a list of conjunction domain relations:

* Addition
* Enumeration
* Contrast
* Adversative
* Concessive
* Comparison

Following is a brief explanation to what each of the relations stands for:

**Addition** - A relation in which one (or more) of the conjuncts are set apart from others, sometimes for rhetorical purposes

**Enumeration** - A relation in which all of the conjuncts have equal status

**Contrast** - A relation which connects conjuncts whose difference is stressed

**Adversative** - A relation which connects conjuncts whose differences are stressed in the utterance

(Note: The difference between Adversative and Contrast is unclear. We need examples in order to determine whether they can be collapsed.)

**Concessive** - Event or state A stands in a concessive relation to event or state B if A is typically not believed to be a result of B. Often introduced in English by "(even) though."

**Comparison** - Entity A stands in the relation of comparison to entity B if the speaker believes that A and B are in some sense similar.

**Particular/Representative Domain Relations**

**Particular** - Relates two textual elements (sentences, paragraphs, etc.) one of which is a special case of the other

**Representative** - Relates two textual elements (sentences, paragraphs, etc.), one of which is an example of the other

**Alternative Domain Relations**  
Relations that are used in situations of choice, parallel to the logical connector "OR."

**Inclusive-or** - The Inclusive-or relation occurs when any of the elements joined by "or" can apply

**Exclusive-or** - The Exclusive-or relation occurs when only one of the elements joined by "or" can apply

**Co reference Domain Relations**  
The relation established among textual references to an object, an event, or a state

**Temporal Domain Relations**

**At** - Two events happen at the same time (the events can be either momentary or prolonged)

**Before/After** - One event happens after/before another in time

**During** - One event takes place after the beginning and before the end of another event

# The use of Ontology in Meaning Representation

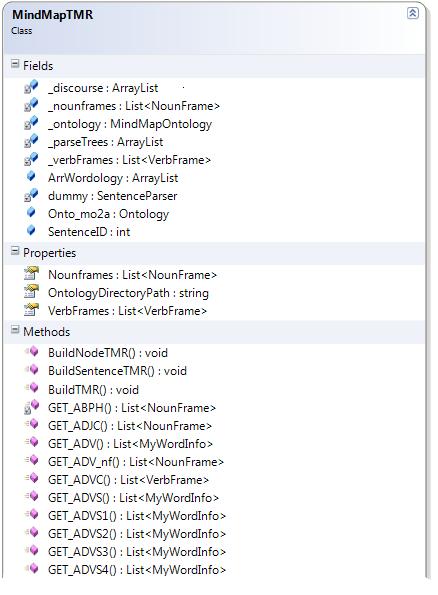
Ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts. A concept is a cognitive unit of meaning – an abstract idea or a mental symbol sometimes defined as a "unit of knowledge", built from other units which act as a concept's characteristics. It, the ontology, is used to reason about the properties of that domain, and may be used to define the domain. Ontology will be further discussed in the following chapter.

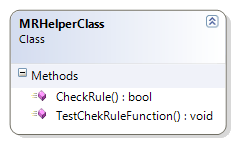
Our use of ontology in meaning representation is to help define the relations between the text elements, previously discussed, in certain cases.

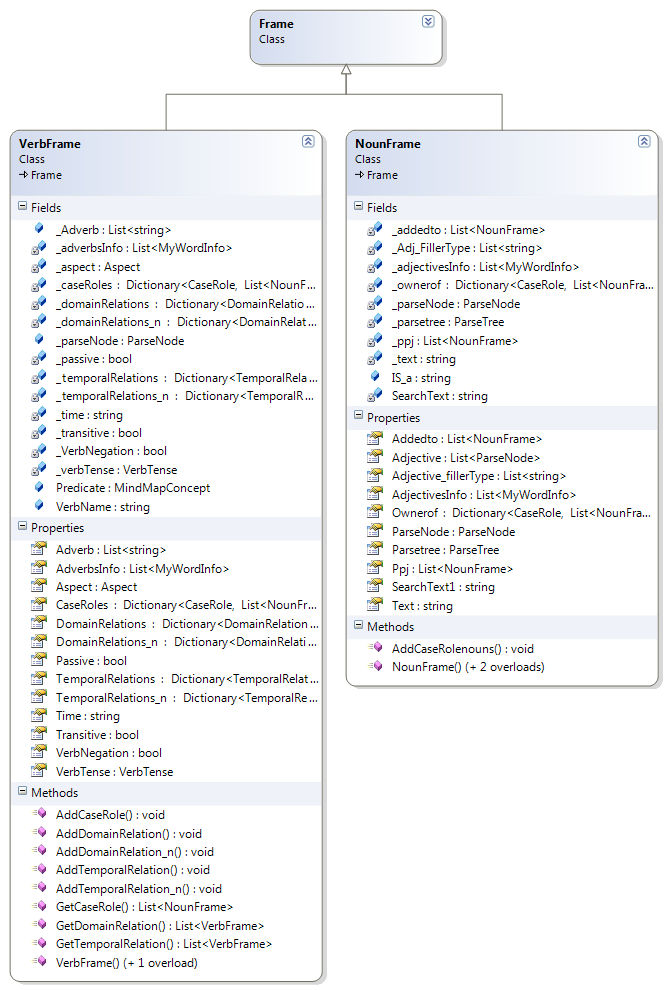
When encountered by a sentence, such as "Mozart gained popularity by performing in concerts and operas", we need to know certain properties of the word following the preposition, in order to determine the relation type to add. A prepositional phrase such *as* "by performing" is a giveaway that the **means** slot is needed, another such as "in concerts and operas" is a giveaway that the **location** slot is needed. But how do we determine which? This can simply be answered by properties in the concept of the words "performing" and "concert", where it will be obvious that that they are an event and location respectively.

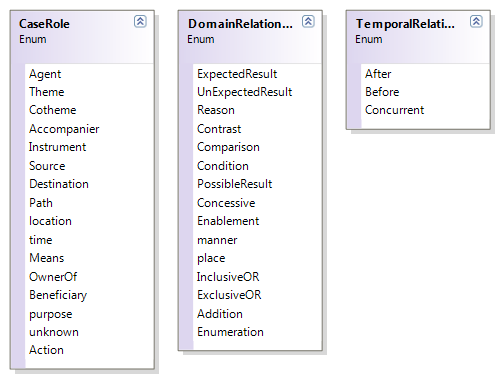
# Design and Implementation

1. The **MindMapTMR** class is sent the parse trees and the discourse.
2. **BuildTMR()** function in **MindMapTMR** is then invoked.
3. **BuildTMR()** function builds the text meaning representation of each sentence.
4. The text meaning representation is built by matching the parse tree of each sentence with functions in the class in which the text meaning representation data structure is built.









Chapter(6)

Multi-level Generation

The goal of this phase is to group each set of related actions or information in the input text into a common concept so that it’s easier for the reader to identify and understand.

Without this phase, if an input of a quite significant size that mentions more than one main concept with a lot of detail is entered, the output would be quite unclear and unorganized like the following figure which defies the purpose of comprehension speed, simplicity and clarity.

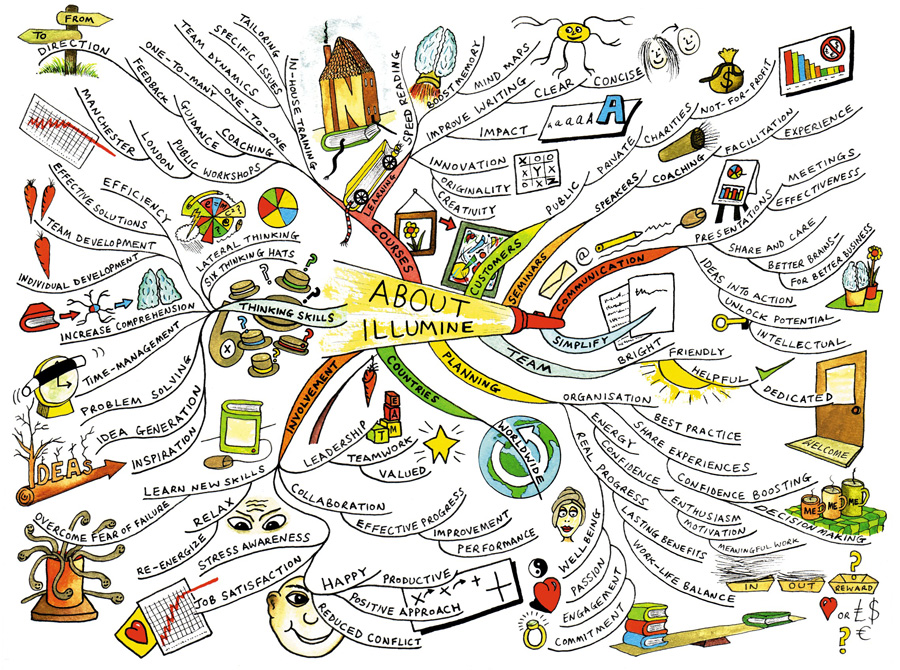


Figure 6.1 Demonstration of why multi-level should be used

Multilevel generation enables the program to build up semantic grouping in topics in the given text content and build up relationships between them such that Mind map can be built up as Parent-children relation, in other words each mind map image can be expanded to child mind maps. So, through a series of steps, the output from the text meaning representation is transformed into an organized multileveled representation where the highest level consists of a summarized form of the map.

# 6.1. Sub-modules

The following is an algorithm we developed for extracting relevant segments out of large description logic ontologies for the purposes of increasing tractability for both humans and computers. This technique takes advantage of the detailed semantics captured within an ontology to produce highly relevant segments.

Figure 6.2 Multi-level phases

# Weight-based Assignment

In this step, weights for the noun frames and verb frames that result from the text meaning representation phase are assigned based on predefined values.

These predefined values were determined based on a series of test cases and assumptions. The goal of this step is to determine the importance of each noun frame or verb frame based on their associations with other frames of the sentence. The weight assigned to each frame indicates its level of importance. The higher the weight, the more important the frame is.

Each of the relations of the case roles and domain relations are given values. The weight per noun frame is the sum of the values of its relations. The weight per verb frame is then deduced from the weights of its surrounding associated nouns.

# Weight-based Partitioning

The goal of this step is to identify the main noun frames present in the current text meaning representation.

The noun frame weights obtained from the previous step are partitioned into clusters using *K-Means++*. The **K-Means++** is obtained by augmenting *K-Means* with a very simple, randomized seeding technique.

The **K-Means** method is a widely used clustering technique that seeks to minimize the average squared distance between points in the same cluster. Although it offers no accuracy guarantees, its simplicity and speed are very appealing in practice.

Clustering is one of the classic problems in machine learning and computational geometry. In the popular k-means formulation, one is given an integer k and a set of n data points in Rd. The goal is to choose k centres so as to minimize φ, the sum of the squared distances between each point and its closest centre. Solving this problem exactly is NP-hard, even with just two clusters, but twenty-five years ago, Lloyd proposed a local search solution that is still very widely used today. Indeed, a recent survey of data mining techniques states that it “is by far the most popular clustering algorithm used in scientific and industrial applications”. Usually referred to simply as K-Means, Lloyd’s algorithm begins with k arbitrary centres, typically chosen uniformly at random from the data points. Each point is then assigned to the nearest centre, and each centre is recomputed as the centre of mass of all points assigned to it. These two steps (assignment and centre calculation) are repeated until the process stabilizes. One can check that the total error φ is monotonically decreasing, which ensures that no clustering is repeated during the course of the algorithm. Since there are at most kn possible clusterings, the process will always terminate. In practice, very few iterations are usually required, which makes the algorithm much faster than most of its competitors. Unfortunately, the empirical speed and simplicity of the k-means algorithm come at the price of accuracy.

There are many natural examples for which the algorithm generates arbitrarily bad clusterings (i.e., φ / (φ OPT) is unbounded even when n and k are fixed). Furthermore, these examples do not rely on an adversarial placement of the starting centres, and the ratio can be unbounded with high probability even with the standard randomized seeding technique.

The K-Means method is a simple and fast algorithm that attempts to locally improve an arbitrary K-Means clustering. It works as follows.

1. Arbitrarily choose k initial centers C = {c1, . . . , ck}.
2. For each i Є {1, . . . , k}, set the cluster Ci to be the set of points in X that are closer to ci than they are to cj for all j ≠ i.
3. For each i Є {1, . . . , k}, set ci to be the center of mass of all points in Ci: ci = 1/|Ci|. Σx ЄCi x.
4. Repeat Steps 2 and 3 until C no longer changes.

The K-Means++ algorithm begins with an arbitrary set of cluster centres. We propose a specific way of choosing these centres. At any given time, let D(x) denote the shortest distance from a data point x to the closest centre we have already chosen. Then, we define the following algorithm, which is known as K-Means++.

1. Choose an initial centre c1 uniformly at random from X.
2. Choose the next centre ci, selecting ci = x’ ∈ X with probability D(x’)2/∑x∈X  D(x)2
3. Repeat Step 1b until we have chosen a total of k centres.

Then proceed as with the standard K-Means algorithm.

*Evaluating k-means++*

Speed:

* + Initialization similar to 1 iteration of Lloyd’s method
  + In practice:
    - Uses fewer iterations than Lloyd’s method
    - Runs *faster* than Lloyd’s method

Simplicity:

* + Only marginally harder to implement than Lloyd’s method
  + Easy to understand intuitively

So, after using K-Means++ to cluster the noun frames’ weights, the noun frames corresponding to the cluster with the greatest centroid are marked as the main noun frames in the current text meaning representation whose associated actions will be semantically grouped using the concept-based partitioning.

# Concept-based Partitioning

The goal is to group related information under one common concept to facilitate the comprehension of the converted text and provide an organized layout of the map.

The grouping of the noun frames or the verb frames is achieved according to *an ontology*.  An ontology is a formal representation of a set of concepts within a domain and the relationships between those concepts. It is used to reason about the properties of that domain, and may be used to define the domain. It enables knowledge sharing and re-use.

An ontology describes the concepts and relationships that are important in a particular domain, providing a vocabulary for that domain as well as a computerized specification of the meaning of terms used in the vocabulary. Ontologies range from taxonomies and classifications, database schemas, to fully axiomatized theories. In recent years, ontologies have been adopted in many business and scientific communities as a way to share, reuse and process domain knowledge.

Ontologies are used in artificial intelligence, the Semantic Web, software engineering, biomedical informatics, library science, and information architecture as a form of knowledge representation about the world or some part of it.

Common components of ontologies include:

* Individuals: instances or objects (the basic or "ground level" objects)
* Classes: sets, collections, concepts, types of objects, or kinds of things.
* Attributes: aspects, properties, features, characteristics, or parameters that objects (and classes) can have.
* Relations: ways in which classes and individuals can be related to one another.
* Function terms: complex structures formed from certain relations that can be used in place of an individual term in a statement.
* Restrictions: formally stated descriptions of what must be true in order for some assertion to be accepted as input.
* Rules: statements in the form of an if-then (antecedent-consequent) sentence that describe the logical inferences that can be drawn from an assertion in a particular form.
* Events: the changing of attributes or relations.

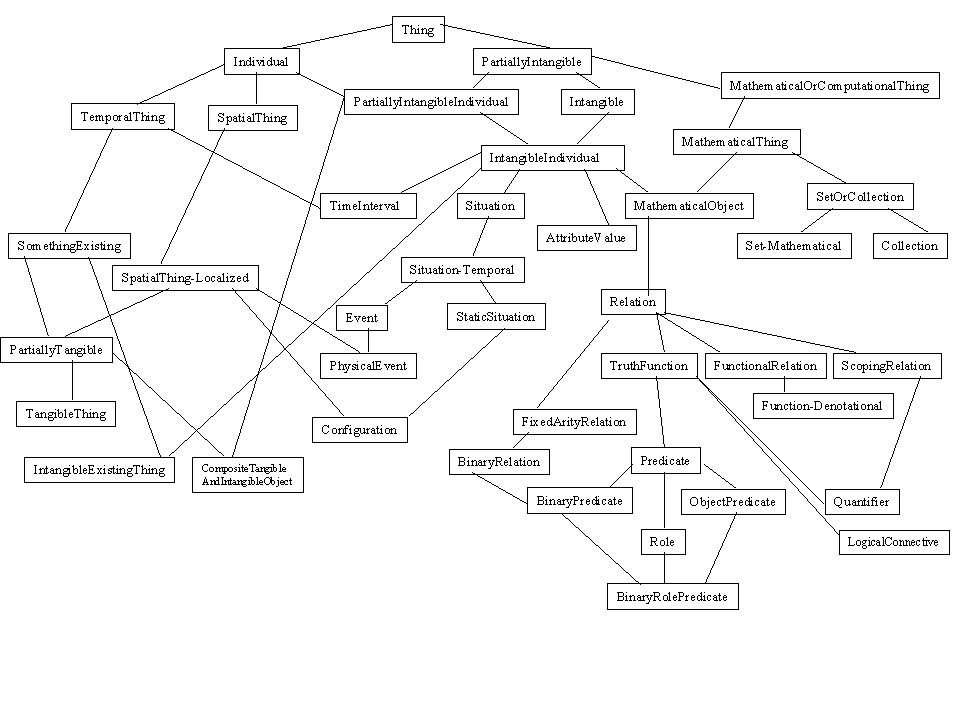


Figure 6.3. Example of an ontology

Ontologies are commonly encoded using *ontology languages*.

Ontology languages are formal languages used to construct ontologies. They allow the encoding of knowledge about specific domains and often include reasoning rules that support the processing of that knowledge.

Examples of ontology languages:

* CycL
* DOGMA (Developing Ontology-Grounded Methods and Applications)
* OWL (Web Ontology Language)

The data described by an OWL ontology is interpreted as a set of "individuals" and a set of "property assertions" which relate these individuals to each other. An OWL ontology consists of a set of axioms which place constraints on sets of individuals (called "classes") and the types of relationships permitted between them. These axioms provide semantics by allowing systems to infer additional information based on the data explicitly provided. For example, an ontology describing families might include axioms stating that a "hasMother" property is only present between two individuals when "hasParent" is also present, and individuals of class "HasTypeOBlood" are never related via "hasParent" to members of the "HasTypeABBlood" class. If it is stated that the individual Harriet is related via "hasMother" to the individual Sue, and that Harriet is a member of the "HasTypeOBlood" class, then it can be inferred that Sue is not a member of "HasTypeABBlood".

We used OWL to encode the ontology according to which the grouping of the noun frames and the verb frames is achieved.

We also used *Protégé* to edit the ontology.  **Protégé** is a free, open-source platform that provides a growing user community with a suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats. Protégé can be customized to provide domain-friendly support for creating knowledge models and entering data. Further, Protégé can be extended by way of a plug-in architecture and a Java-based Application Programming Interface (API) for building knowledge-based tools and applications.

Each noun frame or verb frame is mapped to a concept in the ontology that defines that noun frame or verb frame. For a set of noun frames or verb frames, the concept-based partitioning determines the frames that are semantically related, i.e., the frames whose concepts are close to one another in the ontology, and groups them by getting their nearest parent concept in the ontology.

For example, if the frames’ concepts are like the following:

COUGH – SNEEZE – HEAL – ITCHING – CURE – PILL – TREAT – DRUG – DIE – BLEED – SLEEP.

The grouping will be like the following:

ANIMAL-SYMPTOM {COUGH – SNEEZE – ITCHING}

MEDICAL-EVENT {HEAL – CURE – TREAT}

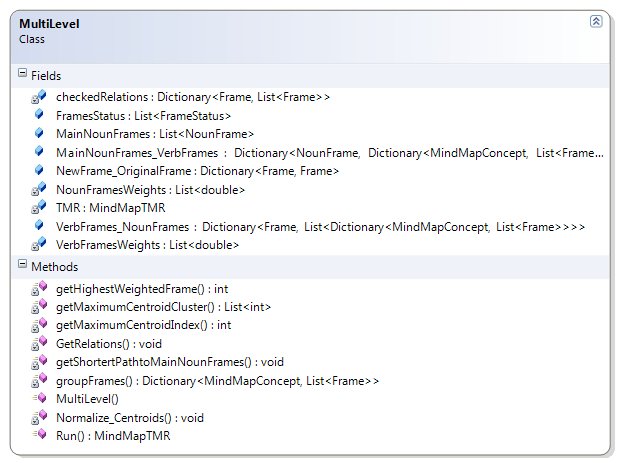
MEDICAL-ARTIFACT {PILL – DRUG}

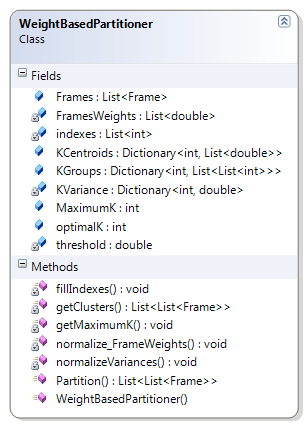
BIOLOGICAL-FUNCTION {DIE – BLEED – SLEEP}

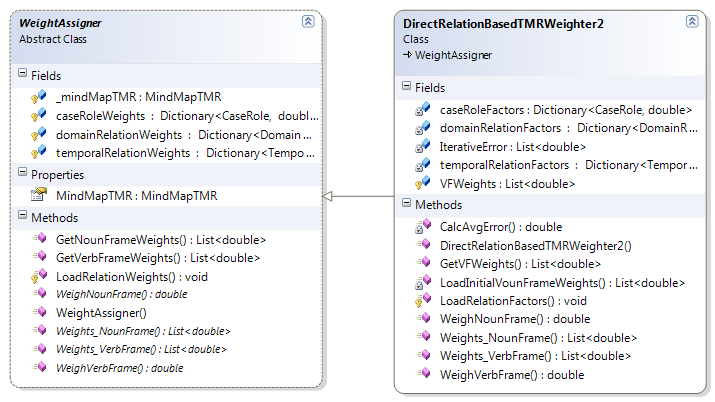
The associated actions of each main noun frame in the text meaning representation are passed to the concept partitioning and a list of concepts with their corresponding frames is returned. If a verb frame of a main noun frame was not grouped with any of the other verb frames, the associated noun frames of that verb frame are grouped using the concept-based partitioning. So, the new text meaning representation will contain the main noun frames, the grouped concepts associated with them as new verb frames, the ungrouped verb frames around the main noun frames and the grouped concepts of noun frames around them as new noun frames.

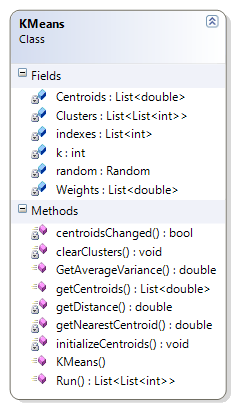
# Design and Implementation

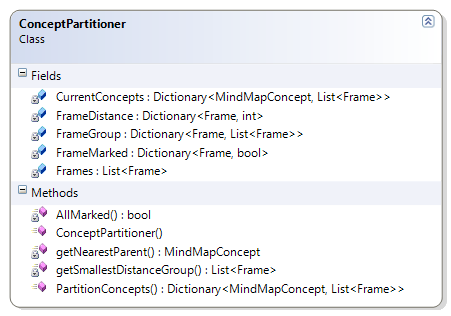
1. The constructor of class **Multilevel** takes **TMR** as an input.
2. TheMultilevel class constructor initializes an instance of **DirectRelationBasedTMRWeighter2**
3. TheMultilevel class constructor calls functions **GetNounFrameWeights()** and **GetVerbFrameWeights()** in class DirectRelationBasedTMRWeighter2 to get the weights of the noun frames and the verb frames.
4. The Multilevel class constructor then calls function **Run()** to run the algorithm.
5. The function Run() initializes an instance of **WeightBasedPartitioner**.
6. The WeightBasedPartitioner class constructor takes a **List<Frame> Frames** and a **List<double> FramesWeights** as an input.
7. The function **Run()** in the **Multilevel** class then calls function Partition in the **WeightBasedPartitioner** class.
8. The function **Partition()** in the **WeightBasedPartitioner** class returns a List<List<Frame>>.
9. The function **Run()** in the **Multilevel** class initializes an instance of **ConceptPartitioner** for each main noun frame.
10. The ConceptPartitioner constructor takes a **List<Frame> Frames** as an input.
11. The function **Run()** in the **Multilevel** class then calls function **Partition** in the **ConceptPartitioner** class which returns a **Dictionary<MindMapConcept, List<Frame>>**.
12. The function **Run()** in the **Multilevel** class then returns the new **TMR**.











Chapter (7)

Mind Map Conversion

The mind map conversion phase is the last phase that takes place before displaying the final output to the reader. This phase is concerned with the conversion of the obtained multileveled text meaning representation into a mind map that contains images, to attract the user’s attention and aid in remembering and understanding the input text.

# Images in Mind Maps

The ability to interpret and make meaning from information presented in the form of an image is known as “Visual literacy”. Visual literacy is based on the idea that pictures can be “read” and that meaning can be communicated through a process of reading. It stresses the importance of active reading based on information visualisation to capture attention and reinforce knowledge.

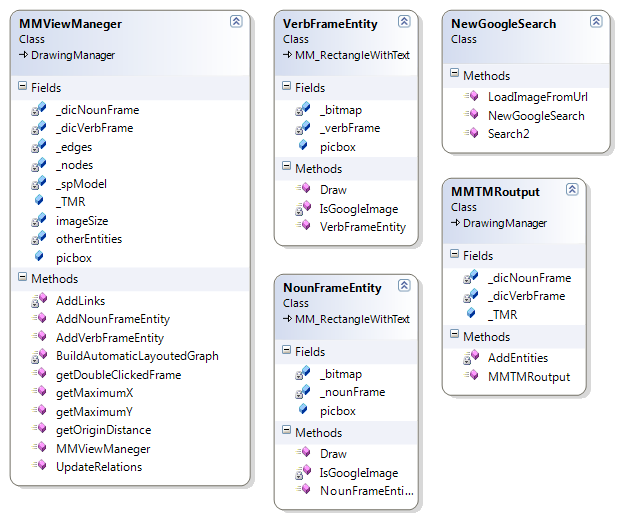
Images add visual interest to your mind maps. They can be utilized as the central topic of your map, or attached to other topics, and add a wealth of meaning and context to your maps**.**Images are used to reinforce the information, clarify a word or phrase, and create excitement. Clear pictures multiply the reader's level of understanding of the material presented. Images add impact and interest to the mind map. They enable you to appeal to more than one sense at the same time, thereby increasing the reader's understanding and retention level. With pictures, the concepts or ideas presented are no longer simply words - but words plus images. Images help to focus the brain**,** which prefers colorful objects to black-and-white ones. They leverage your brain’s powerful associative capabilities**.** Images tap into our imagination at a much deeper level than words alone can.Words and images together multiply your intellectual power.It’s the combination of the two that make mind maps such a rich visual medium for creative expression**.** Images are an excellent memory aid.Simply put, they tend to be more memorable to your brain than words do.

The cliché “a picture is worth a thousand words” is truer than ever before. For example a phrase like: “small red ball ”, in the text meaning representation ”ball” is a noun frame node connected to two other nodes, one represents the adjective “small” and the other represents the adjective “red”. When this representation is converted into a mind map, only one node with an image of a small red ball is displayed. This helps the mind map reader to get a better understanding of the text with much less effort from his side. The reader won’t have to go through a lot of connections to get to the desired information, a simple image can get the information through to the user in a shorter time.

The images displayed in the mind map are obtained using Google’s image search. The Google Image Search API is essentially a tool that you can use if you need to perform an image search against Google programmatically. Since it parses the HTML returned by Google, if the format of this HTML changes, the library's implementation will have to change accordingly.

# Design and Implementation

1. The Text Meaning Representation, containing the NounFrames, VerbFrames and their relations, is sent along with the form onto which the mind map will be displayed to the **MMVManager** component.
2. In the **MMVManager**, a node for each NounFrame and VerbFrame, and edges to represent their associations are constructed.
3. For each Noun Frame, the noun name and the adjectives related to this noun are used as a search parameter.
4. This parameter is used to obtain a suitable image through Google's image search.
5. The images obtained are in bitmap format and the mind map is then drawn using the **MMVManager** component.



Chapter (8)

Automatic Layout Allocation

Embeddings of graphs have been of interest to theoreticians for some time, in particular those of planar graphs and graphs that are close to being planar. One definition of a planar graph is one that can be drawn in the plane with no edge crossings. While working on the four-colour problem, Wagner(1936) was the first to show that every planar graph has a straight-line embedding. Tutte (1960, 1963) showed that every 3-connected planar graph has a convex embedding. While graph theory was originally an artifact from mathematics, it has become quite prevalent as a means of solving problems or representing data. With automatically generated data sets being represented as graphs, came the need to automatically generate embeddings of graphs in a 2-dimensional space, such as a computer terminal or a sheet of paper. A wide variety of fields each with their own requirements utilize automatic graph drawing algorithms. As a result, many different algorithms have been developed over the last decade. Within the last five years, there have been annual conferences on graph drawing (Di Battista, et al., 1993; Tamassia and Tollis, 1995; Brandenberg, 1996), special issues of journals on the topic and this year a monograph was published. (Cruz and Tamassia, 1994) Two different algorithms from the field will be presented briefly : the spring model algorithm and the simulated annealing method.

# Applications of Graph Drawing Algorithms

Work in this area was done mainly in response to requirements of data visualization techniques and interactive computer systems. Many fields in computer science, such as software engineering, electronic circuit design and database design, have found it useful represent data as graphs, with vertices denoting elements and edges denoting relations between them. These graphs are normally generated by software tools based on information in the system. As the size of a graph generated from data or constraints grew, so has the sophistication of embedding algorithms.

In software engineering, the architecture of a large software system can be visualized as a directed graph with vertices representing modules and edges denoting various use relations between them. These systems are often hierarchical in nature and their drawings should reflect this. Furthermore, this information can be used to make the graph drawing task easier.

Computer hardware and microchips are now sufficiently complex that they are designed using CAD tools. It is then the responsibility of the tool to create a layout of the logical gates and the connections between them on microchips and circuit boards. This layout should be a *grid* drawing. An *orthogonal* drawing is one in which an edge is a chain of horizontal and vertical segments. A *grid* drawing is an orthogonal drawing in which all the vertices and bends of edges have integer coordinates. There are many other examples of applications which use graph drawing algorithms.

Entity-relationship diagrams in database design can have a visual representation or an algebraic one. One of the requirements of systems analysis and design tools is that a database description need only be entered once in either format and the alternate format will be generated. There is a project management technique that uses PERT charts (Project Evaluation and Review Technique) to track dependencies among tasks. These dependencies form a directed graph from which other information can be derived, such as a project critical path. One technique used by the Human Genome Project analyses the gene structure by representing raw data as a directed graph.

# Aesthetics

It is not difficult to design a naïve algorithm to display a graph. A random layout places vertices randomly within a finite space. Edges can be drawn as minimum length straight lines between vertices or they may be *polylines*, that is, lines with bends in them to avoid drawing elements. A circular layout algorithms places the vertices along the perimeter of a circle and edges are drawn across the circle. A similar strategy places vertices at the intersections of an *n**n* grid along the main diagonal.

The circular layout method is particularly effective for representing cliques as it emphasizes the regularity of the graph. Unfortunately, most graphs are not cliques. None of the above methods pays much attention to the readability of the resulting graph, either as a collection of lines and dots or the data that they represent. From a graph theoretic point of view, isomorphic graphs should look similar. Also, planar graphs should be drawn without edge crossings so it is easy to visually confirm the planarity of the graph. Consequently, the drawings generated by the above strategies may not be very informative.

The quality or usefulness of a particular embedding is highly dependent on its application domain. Therefore a graph drawing algorithm must take into account *aesthetics*: criteria for making salient characteristics of the graph easily readable. Readability and “salient characteristics” are highly subjective and dependent on the purpose for which the drawing is generated. Some aesthetic criteria include:

minimize the number of edge crossings;

draw edges as straight as possible;

vertices should be evenly distributed;

the majority directed edges should be drawn pointing in the same direction;

in polyline drawings, minimize bends in the edges;

minimize the area of the area drawing;

maximize display of symmetries;

maximize *angular resolution*.

The angular resolution of a line drawing of a graph is the smallest angle formed by two edges incident on the same vertex. If this angle is too small, it may be beyond the resolution of graphic display device or even the human eye, and the two incident edges end up looking like a fuzzy blob.

In general, it is not possible to optimize two criteria simultaneously. For example, in the two embeddings of K4 below, the one on the left minimizes the number of edge crossings, whereas the one on the right maximizes the display of symmetries. (Cruz and Tamassia,1984)

Many of these optimization problems are either NP-hard or NP-complete. Minimizing the number of crossings in an embedding is NP-complete, even if the graph is hierarchical with only two layers. Minimizing the area of a grid drawing is NP-hard, as is minimizing the length of the maximum edge length.

Polytime graph drawing algorithms tend to either use heuristics to approximate an NP-hard optimization, use innovative techniques to manipulate the layout or some combination of the two. The more complex algorithms that yield more aesthetically pleasing graphs are usually able to approximate optimizations of several criteria. The specific criteria used are often determined by the specific application domain or graph type for which the algorithm was developed.

# The Spring Model Algorithm

There are many different strategies that can be used to draw a general undirected graph. One method is to use a planar embedding algorithm because a planar embedding can be constructed in linear time. The first step is to test for planarity and if the test is positive construct the embedding. If the graph is not planar, then it can be *planarized* by a variety of techniques such as deleting edges, splitting dummy vertices or adding dummy vertices at edge crossings. Planarity testing can be done in linear time. Planarization is an NP-hard problem but can be done using heuristics in O(n2) or less. So overall, this method is relatively efficient.

Another method is to orient the edges, say using an Eulerian walk and use a directed graph drawing algorithm. Finally, there is a family of force-directed algorithms which involves transforming the vertices and edges into a system of forces and finding the minimum energy state of the system. This state is found either by solving differential equations or by running a simulation of the forces. The spring model is the most popular algorithm in this family. The spring model was originally developed by Eades, 1984. The version presented here is a descendant of that algorithm developed by Kamada (1989).

Conceptually, the spring embedder works by replacing edges with springs unit some natural length. It also adds springs with larger natural lengths between non-adjacent vertices. The vertices are initially placed randomly and a system of differential equations is solved to find the system state with minimum energy. By nature, springs attract their endpoints when stretched and repel their endpoints when compressed. Vertices that are adjacent are kept close to each other by shorter springs. While the longer springs between non-adjacent vertices keep them apart, and at the same time they limit the overall size of the embedding.

The total energy of the system is represented by the following summation:

n-1 n

E = ∑ ∑ ½ kij (|pi - pj| - lij)2

i=1 j=i+1

*p1,  p2, pn* are particles on a plane representing vertices *v1,  v2,  v*n

*l i j* is the natural length of the spring between *vi* and *vj* and is defined as *li j* =L\*dij, where *L* is the desired length of the edge in the embedding and *dij* is the shortest length path between *vi* and *vj* in the graph. *L* is sometimes chosen as a function of the diameter of the graph and the available embedding area. *kij* is the strength of the spring between *vi* and *vj* and is defined as *kij* = K/*dij* , where *K* is a constant and *dij* is as defined above.

Rewriting the energy summation in terms of xy-coordinates, we get:

n-1 n

E = ∑ ∑ ½ kij( (xi-xj)2 + (yi-yj)2 + lij2 -2l √(xi-xj)2 + (yi-yj)2)

i=1 j=i+1

To find local minima of this summation, we need to take partial derivatives of this and solve to find local minima for each vertex. This results in 2*n* non-linear non-independent equations. This system of equations is rather difficult to solve, so the problem is considered one particle at a time. The particle in the system with the highest energy is identified and moved to a stable location with low energy. Let this particle be *pm* , with energy function:

Δm=√(∂E/∂X)2+(∂E/∂Y)2

Pm is moved in a stepwise manner to minimize Δm. This location corresponds to where ∂E/∂Xm = ∂E/∂Ym = 0 is satisfied. This is done by iteratively solving the following two linear equations for *x* and *y*, where t is the current iteration and adding them to *Xm* and *Ym* respectively. These steps are repeated until the new energy function stops decreasing.

∂2E/∂X2m (Xtm , Ytm) δX + ∂2E/∂Xm ∂Ym (Xtm , Ytm) δY = -∂E/∂Xm(Xtm , Ytm)

∂2E/∂Xm ∂Ym (Xtm , Ytm) δX + ∂2E/∂Y2m (Xtm , Ytm) δY = -∂E/∂Ym(Xtm , Ytm)

The method chosen to calculate the shortest path in the spring model is the Floyd Warshall algorithm.

The Floyd-Warshall algorithm compares all possible paths through the graph between each pair of vertices. It is able to do this with only *V*3 comparisons. This is remarkable considering that there may be up to *V*2 edges in the graph, and every combination of edges is tested. It does so by incrementally improving an estimate on the shortest path between two vertices, until the estimate is known to be optimal.

Consider a graph *G* with vertices *V*, each numbered 1 through N. Further consider a function shortestPath(*i*,*j*,*k*) that returns the shortest possible path from *i* to *j* using only vertices 1 through *k* as intermediate points along the way. Now, given this function, our goal is to find the shortest path from each *i* to each *j* using only nodes 1 through *k* + 1.

There are two candidates for this path: either the true shortest path only uses nodes in the set (1...*k*); or there exists some path that goes from *i* to *k* + 1, then from *k* + 1 to *j* that is better. We know that the best path from *i* to *j* that only uses nodes 1 through *k* is defined by shortestPath(*i*,*j*,*k*), and it is clear that if there were a better path from *i* to *k* + 1 to *j*, then the length of this path would be the concatenation of the shortest path from *i* to *k* + 1 (using vertices in (1...*k*)) and the shortest path from *k* + 1 to *j* (also using vertices in (1...*k*)).

Therefore, we can define *shortestPath*(*i*,*j*,*k*) in terms of the following recursive formula:

shortestPath(i,j,k) = min(shortestPath(i, j, k-1),shortestPath(i, k, k-1) + shortestPath(k, j, k-1));

shortestPath(i, j, 0) = edgeCost(i, j);

This formula is the heart of Floyd Warshall. The algorithm works by first computing *shortestPath*(*i*,*j*,1) for all (i,j) pairs, then using that to find *shortestPath*(*i*,*j*,2) for all (*i*,*j*) pairs, etc. This process continues until k=n, and we have found the shortest path for all (*i*,*j*) pairs using any intermediate vertices.

The pseudocode for the Floyd-Warshall algorithm is as follows:

1 /\* Assume a function *edgeCost*(i,j) which returns the cost of the edge from i to j

2 (infinity if there is none).

3 Also assume that **n** is the number of vertices and *edgeCost*(i,i)=0

4 \*/

5

6 double path[][];

7 /\* A 2-dimensional matrix. At each step in the algorithm, path[i][j] is the shortest path

8 from i to j using intermediate vertices (1..k-1). Each path[i][j] is initialized to

9 *edgeCost*(i,j) or infinity if there is no edge between *i* and *j*.

10 \*/

11

12 **procedure** *FloydWarshall* ()

13 **for** *k*: = 1 **to** *n*

14 **for each** (*i*,*j*) **in** {1,..,*n*}2

15 path[i][j] = min ( path[i][j], path[i][k]+path[k][j] );

The Spring model algorithm is summarized by the following pseudocode.

1. compute *dij* for 1 *i* *j* *n* ;

2. compute *lij* for 1 *i* *j* *n* ;

3. compute *kij* for 1 *i* *j* *n* ;

4. initialize *p1, p2,  pn*

5. while (max *i* )

6. let *pm* be the particle satisfying *m* = max *i* ;

7. while (*m*>)

8. compute *x* and *y* ;

9.  *Xm* *Xm +* δX

10. YmYm + δY

11. end while

12. end while

When a local minimum has been found, each pair of particles are exchanged and the energy of the system is tested. If a swap results in a lower energy state, the energy minimization process restarted with the new configuration as a starting state. This exchange and compare process provides a means of escaping large local minima.

Eventually the system converges to a global minimum. It takes O(n3) time to find all pairs shortest paths. O(n) time is needed to compute each of *m* , *x* and *y* during each iteration. With some bookkeeping, max *i* can be found in O(1) time. The time required by the energy minimization process is O(Tn), where T is the total number of inner loops. T is difficult to characterize beyond this because it depends on the value of , the initial position of the vertices, and the graph itself.

This algorithm reduces the number of edge crossings as their presence increases the energy in the system. Also, drawings produced by this method display symmetries present in the graph. The same graph with different initial conditions will converge to the same drawing. Graphs with similar structures will also be drawn similarly. This method can be extended to layered hierarchical graphs by assigning vertices in the same level a fixed *y* value and allowing only the *x* position to vary.

# Simulated Annealing Method

In the most general terms, annealing is the process by which some substance is heated until it is a liquid and then slowly cooled until it is a solid. The slow cooling allows the molecules to organize themselves into a crystal, a totally ordered form. This process is often used in the manufacturing of products such as steel. Simulations of annealing were developed to analyze the efficiency of these manufacturing processes. Simulated annealing is used as a problem solving technique where the potential solution space is large and a combinatorial search is infeasible. It has been applied with some success to problems such as VLSI circuit design, graph partitioning and the traveling salesman problem. Davidson and Harel (1996) have applied this technique to drawing undirected graphs with straight-line edges. One big advantage of this algorithm is that the relative importance of different aesthetic criteria can be varied.

The basic shape of a simulated annealing algorithm is given in the pseudocode below.

1. Initialize temperature T;  
2. Initialize configuration ;  
3. E cost of ;  
4. while(min T not reached)  
5. while(termination rule is not satisfied)  
6. choose a new configuration from the neighbourhood of ;  
7. Ecost of ;  
8. if ((E< E) OR (*random* < e(E-E’)/T )) then  
9. ;  
10. E E;  
11. end if  
12. decrease temperature T  
13. end while  
14. end while

A *configuration* is a proposed embedding of the input graph. The graph can be entered either as a set of adjacencies or as a hand drawing. If the input is a set of adjacencies then the initial configuration is randomly generated.

A *neighbourhood* of a configuration , is the set of configurations that differ from by the location of a single vertex. is generated by taking a vertex and placing it on the perimeter of a circle drawn around its original location. To simulate the behaviour of molecules in the physical annealing process, the size of this circle is initially large and decreases with T.

The *cost* of a configuration must be carefully chosen so that it reflects the desired aesthetics of the final graph and is not overly computationally intensive to compute.

1. To ensure that vertices are evenly distributed to avoid overcrowding, the term *aij=λ1/d2ij* is added to the cost function pair of vertices, *i, j*. *d2ij*is the Euclideanstraight line distance between *i* and *j*. 1 is a weighting factor whose valuedepend in the importance of this criteria relative to the others in the cost function.
2. A completely minimized cost function will spread out the vertices indefinitely.  
   To ensure that vertices are drawn on the display area, the following term is added to the cost function for each node *i*:  
    *mi=* 2 (1/ri2 +1/li2+1/ti2+1/bi2), where ri, li , ti  and *bi* are the straight-line distances between the vertex and the right, left, top and bottom borders of the display area. Again, 2 is a weighting factor.
3. To avoid overly long edges, the following term is added for each edge *k* of length *d2k* with weighting factor *ck =* *d2k*
4. To reduce the number of edge crossings  is added to the cost function for pair of edges that cross.
5. To prevent edges from being drawn too close together, particularly those that intersect at a vertex, the following term is added for every vertex *i*, edge *k* with distance *gik* and weighting factor : *hik =* */ g2ik*

The relative importance of each of the criteria can be adjusted by varying the weighting factors. The fifth factor is normally used only during an optional *fine tuning* stage of the algorithm. The main part of the algorithm is run with only the first four factors until a termination condition is reached. Then the drawing may be adjusted further by using the larger cost function.

The actual value of the *initial temperature* depends on how many iterations are desired in the annealing process. At each temperature level, approximately 30*n* perturbations should be performed. The cooling function should be geometric, i.e. *Tp+1 =**Tp*, with 0.6 0.95. A value of 0.75 gives a relatively rapid cooling with the resulting graphs giving good aesthetic properties.

The *termination condition* can be a fixed number of iterations, say 10, or when the proposed embedding stops changing significantly over two to three iterations.

The simulated annealing algorithm runs in O(n2m) time. The number of temperature changes required to terminate the outer loop is constant in the sense that it does not depend on the input graph. O(n) perturbations are performed in the inner loop, say 30*n*. It takes O(nm) time to update the cost function, which is more efficient that calculating it from scratch. Since the location of only one vertex is changed in a new configuration, n vertices need to be updated and for each of these at most m edges will also need to be updated.

This technique produces drawings that are comparable to those generated by the spring method. This algorithm does not produce conventional looking graphs for a several categories of graphs. One such group is a large cycle with no chords. While this is normally drawn as a large circle, this algorithm tends to draw the cycle curled around itself so there is not a large empty space in the middle. Simulated annealing is also not very good at bringing out the regularity of a graph such as a clique because it tends to draw all edges with comparable length. Although this has not yet been rigorously shown, this algorithm appears to be more successful with more complex graphs than spring algorithms. One useful property of simulated annealing algorithms is that they are easily parallelized.

# Design and Implementation

The algorithm chosen to generate the automatically allocated graphs was the spring model algorithm as it produced the best results suitable for our goal.

The vertices are initially placed randomly and a system of differential equations is solved to find the system state with minimum energy.

The initialization of the spring model algorithm takes place by one of three modes:

* the Random mode
* the Circular mode
* the User-Defined mode

The random mode involves randomly changing nodes' positions till a certain number of iterations is reached or the system’s energy is minimized.

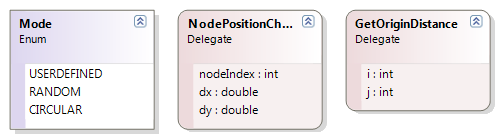
In the circular mode, initially the node with the largest number of neighbours is placed in the center of the graph with its neighbours in a circle around it.

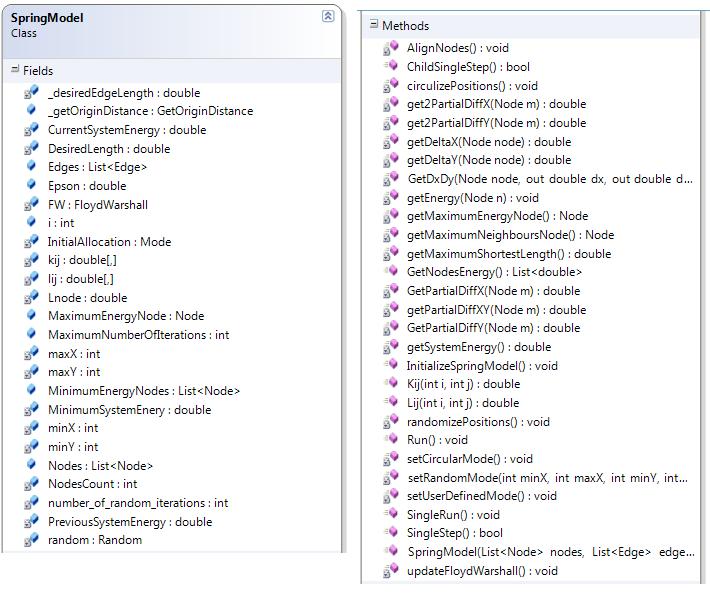
In the User-Defined mode, the user chooses the initial positions of the nodes in the graph. This mode was mostly used during early testing phases.

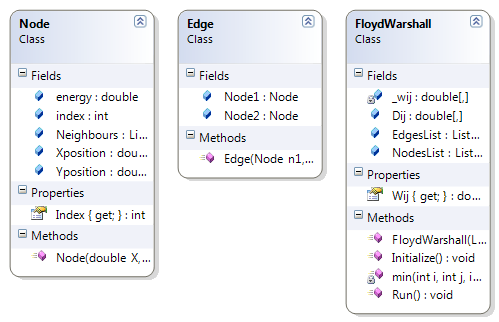
In our final implementation, the random mode is the method chosen to obtain the laid out graphs. The best results are obtained using this mode, but it takes longer time to obtain the solution.

The **SpringModel** class calls the main function **Run()** that runs the algorithm ten times using the random mode and a near optimal solution is obtained.

The **FloydWarshall** class is responsible for calculating the shortest path between each pair of nodes.







Chapter (9)

Testing

This is the phase where the testing of the project takes place. Working on a sample text, we test the text meaning representation, multilevel generation and the mind map conversion along with the automatic layout allocation. Following in figure 9.1 is a snapshot of the sample input text.

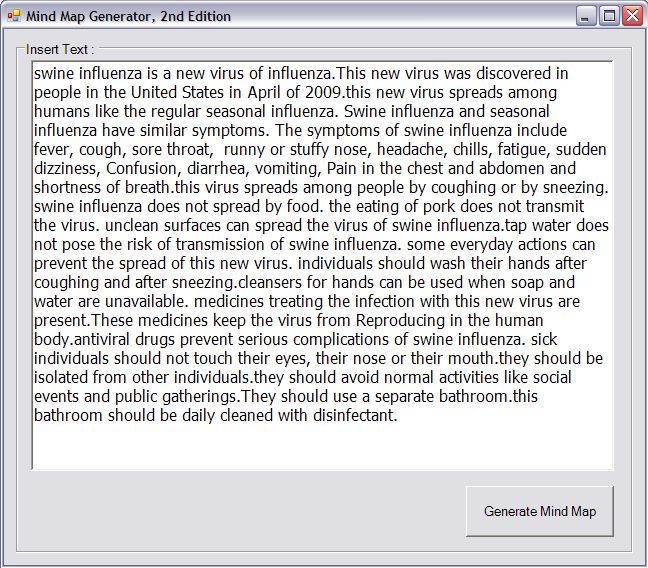


Figure 9.1 Input form with sample text

# Text Meaning Representation Testing

The project was successfully able to produce the text meaning representation for the sample text input in figure 9.1. Figures 9.2 and 9.3 shown below show a graphical representation of the text meaning representation of the sample input.

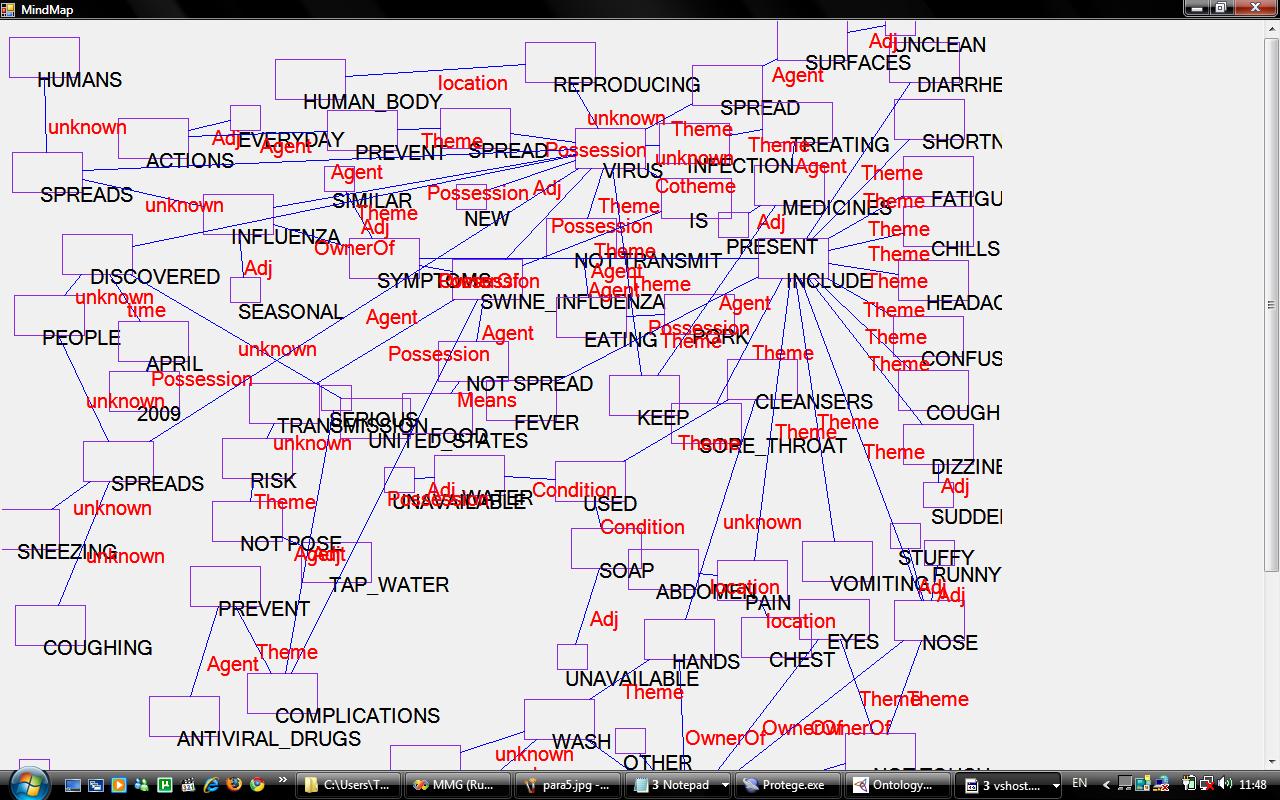


Figure 9.2 TMR screenshot 1

Scrolling down a bit to view the rest of the text meaning representation is shown in Figure 9.3

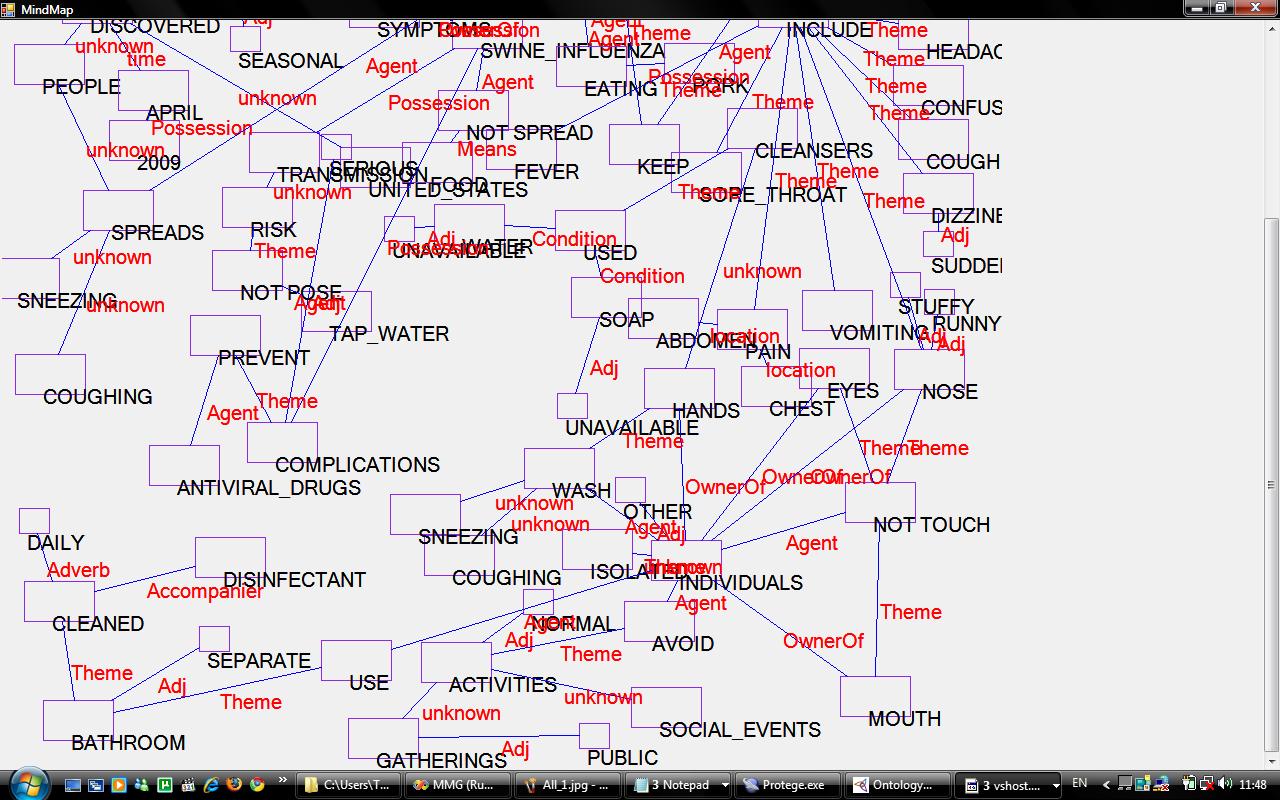


Figure 9.3 TMR screenshot 2

As can be seen, the output of the Text meaning representation, all viewed in a single level, is too much for one screen. It is also difficult to understand the meaning of anything in the sample text in that form, and the automatic layout allocation was not efficient on so many nodes. That is why we introduce multilevel text meaning representation.

# Multilevel Generation Testing

To make the output more comprehensible, simpler and easier on the eye, and also to enable automatic layout allocation to work as intended, multileveling of the text meaning representation takes place.

For example, for the sample input, the multilevel generation module was able to identify that "virus" was the main topic, and so made it the central point of the first level of text meaning representation of the first level mind map, as shown in Figure 9.4. In that level, and every other level, the central point along with its main associations are included.

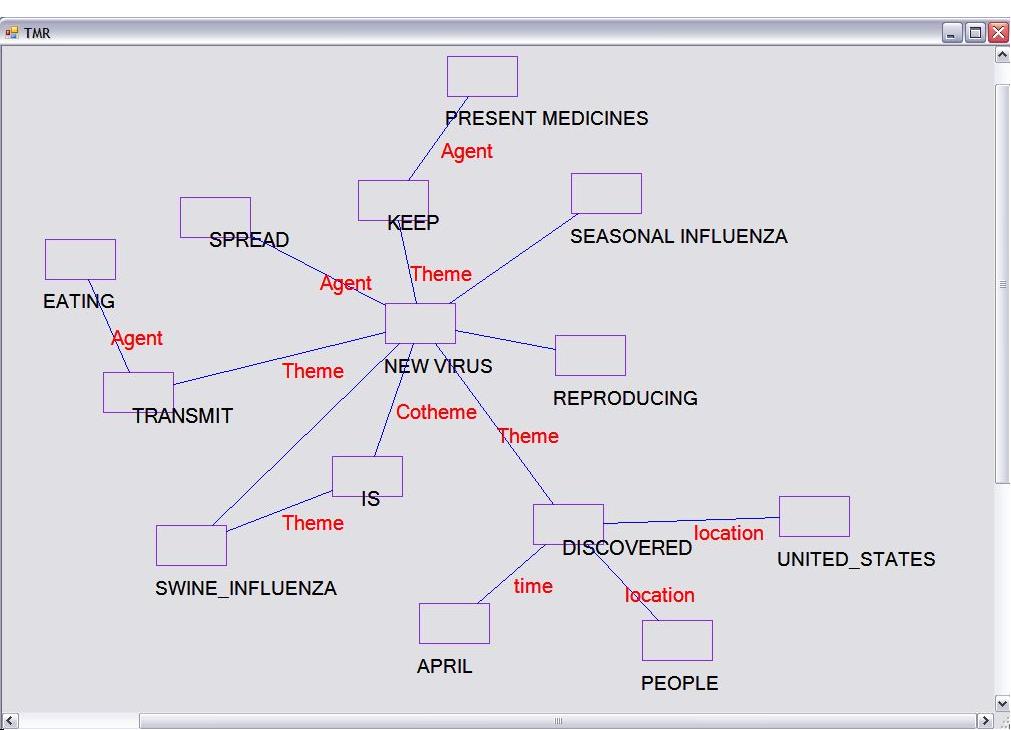


Figure 9.4 TMR level 1 with "virus" as the central topic

Looking at both sample input and the text meaning representation shown above, it is obvious that the main issues related to "virus" are present in this level. To elaborate on a sub-topic, for example Swine influenza, you have only to click on the node to view "Swine influenza" as the central topic of another mind map.

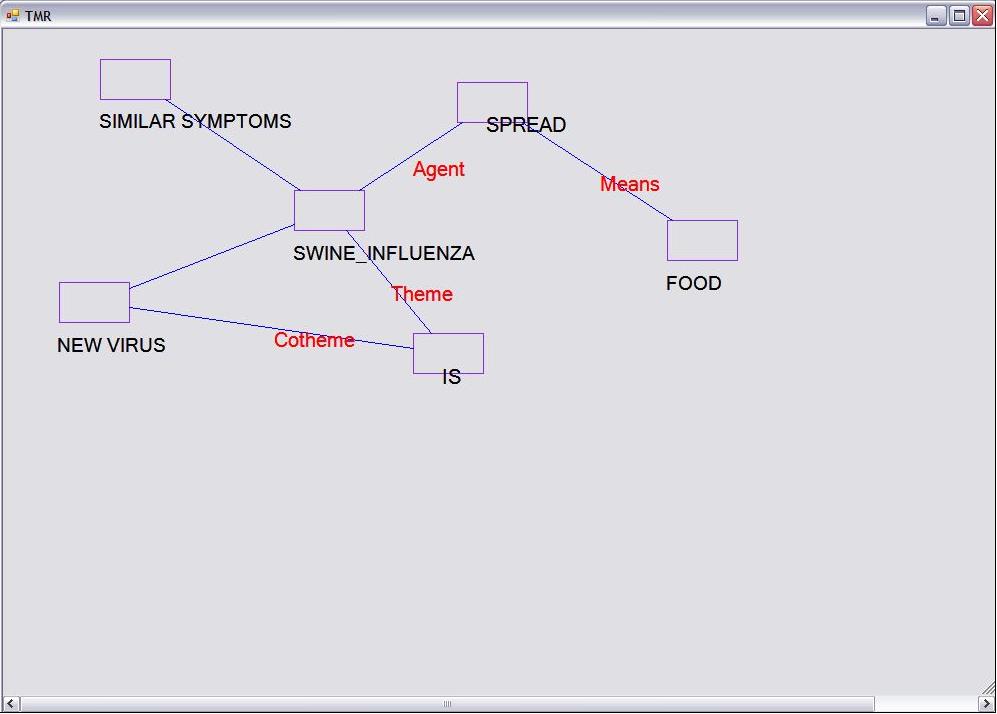


Figure 9.5 TMR level 2 with "Swine influenza" as the central topic

Clicking on "similar symptoms" in level 2, we get the text meaning representation shown in Figure 9.6.

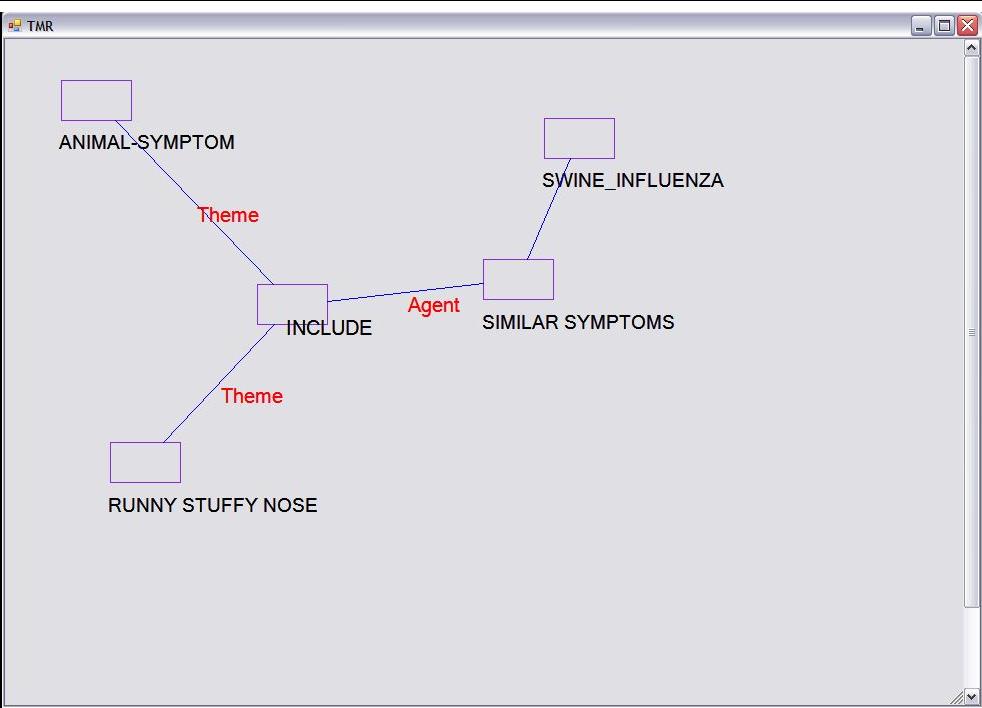


Figure 9.6 TMR level 3 with "similar symptoms" as the central topic

For example, to find out what those "animal symptoms" are, clicking on that node will result in the text meaning representation in Figure 9.7.

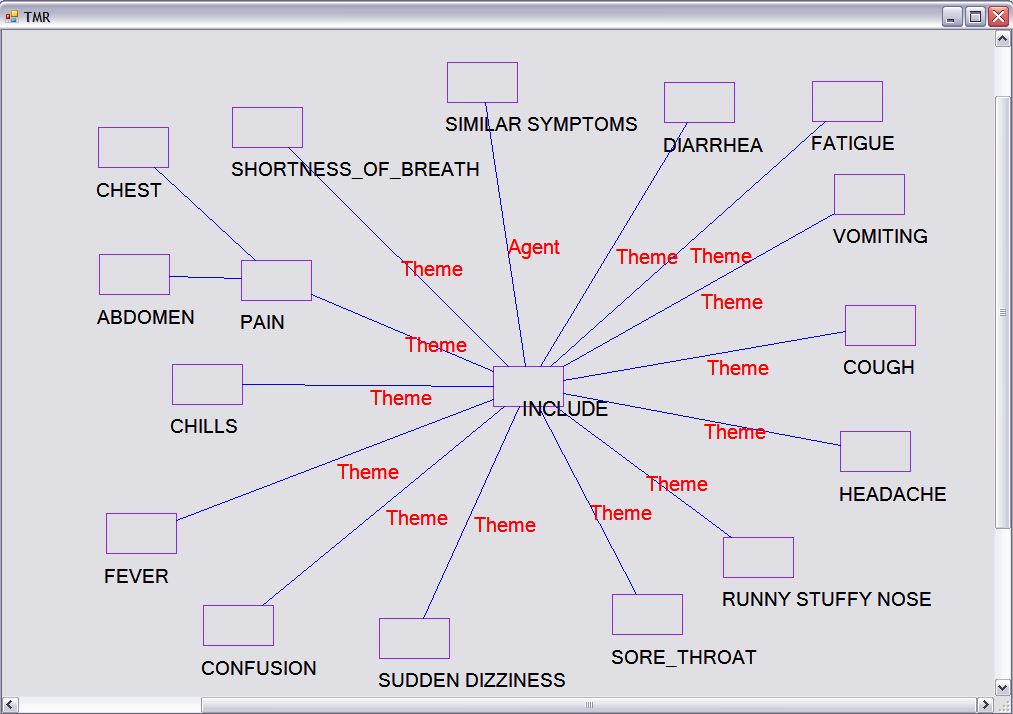


Figure 9.7 TMR level 4 showing the symptoms

This process of revealing more levels, per node, can go on until the node being clicked cannot expand to another mind map. For example, the input text mentioned that one of the symptoms was "Chills", but did not elaborate on it. Therefore, the clicking on "chills" will not result in a mind map with "chills" as the central topic.

All that we have seen above was the text meaning representation of the sample input text, which is not intended for the user to see. The mind maps into which the meaning representation is converted is what the user is supposed to see.

# Mind Map Conversion and Automatic Layout Allocation Testing

The mind maps intended for the user should have pictures, be automatically laid out on the screen and not display any text meaning representation relations. The following figure, figure 9.8, is the display of the first level mind map.

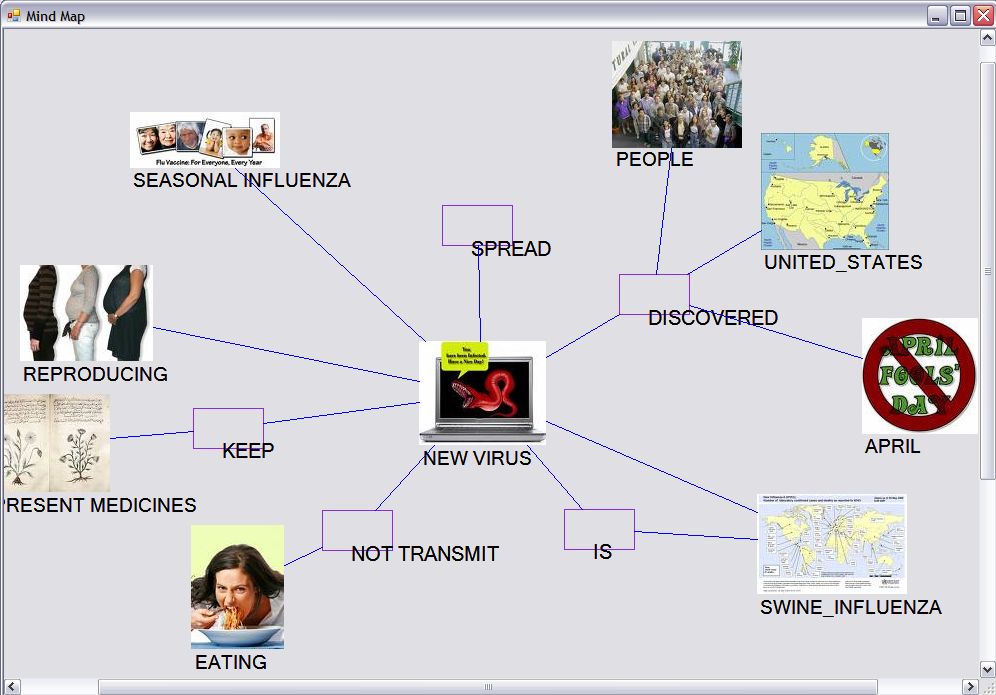


Figure 9.8 the first mind map produced by the input text.

Clicking on "Spread", will produce the mind map shown in Figure 9.9.

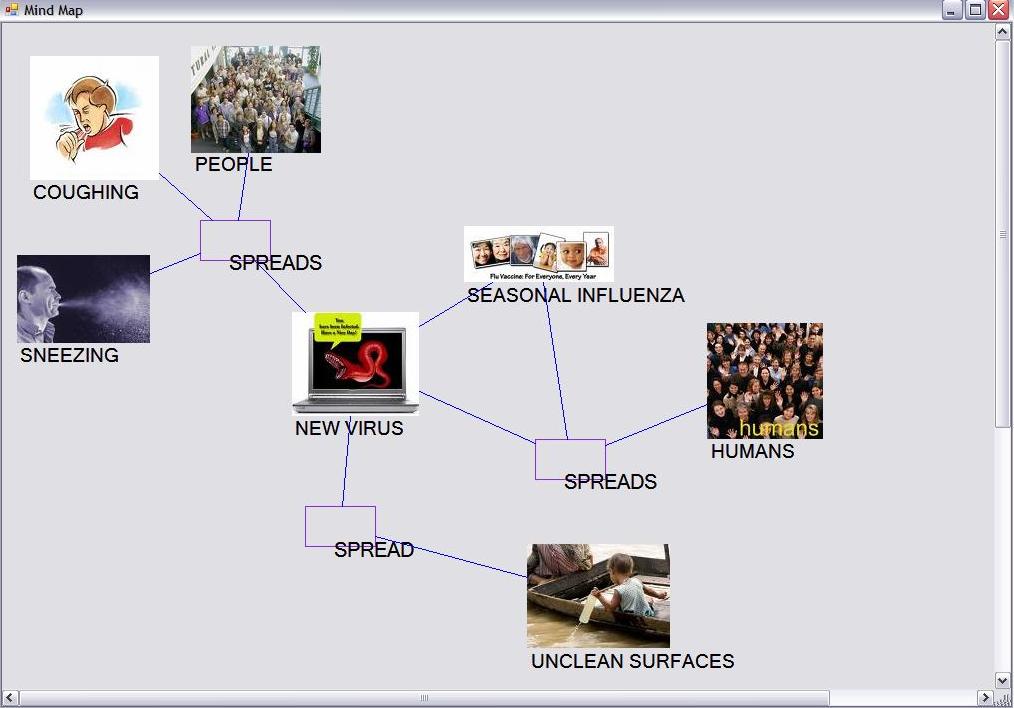


Figure 9.9 the mind map produced by clicking on "Spread."

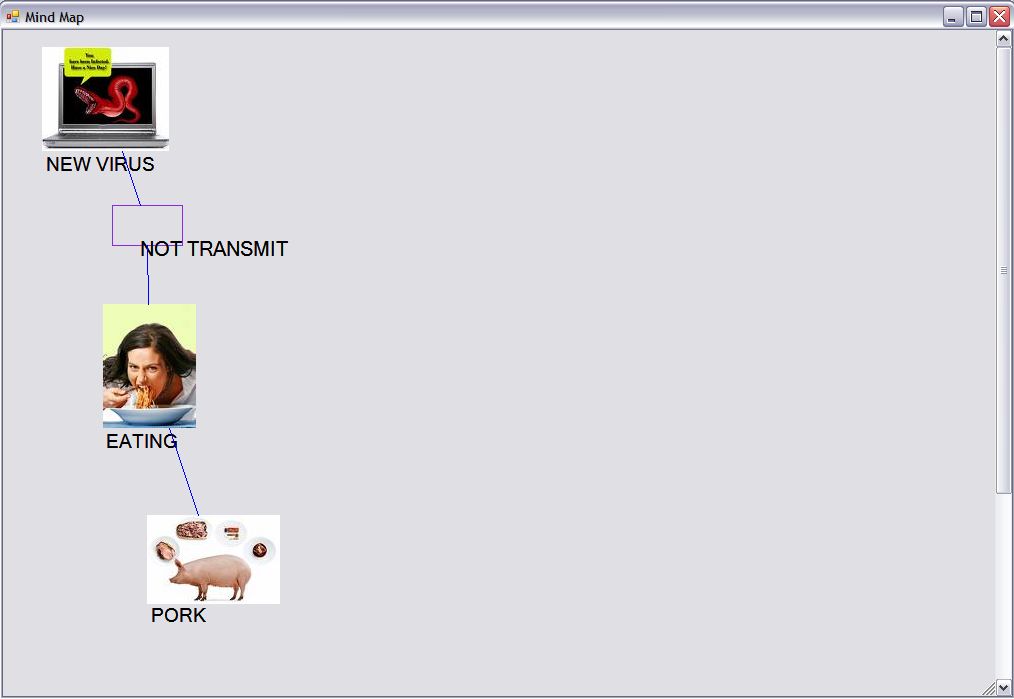
Clicking on "Not Transmit" in Figure 9.8, the following mind map is produced.  
  


Figure 9.10 The mind map produced by clicking on "Not Transmit."

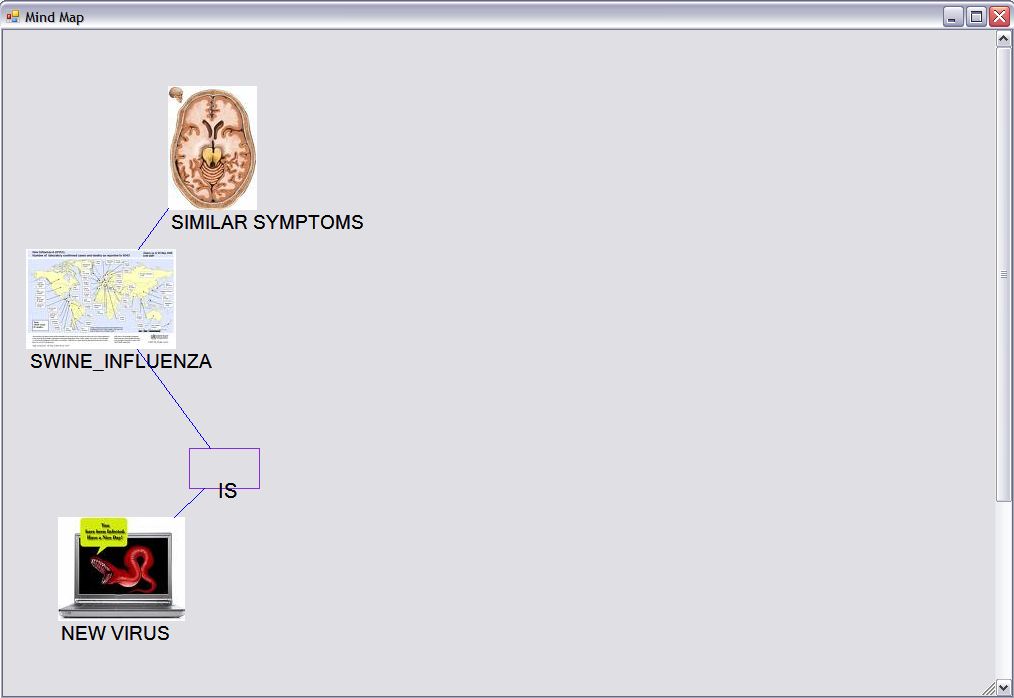
Clicking on "Swine Influenza" in Figure 9.8 produces the following mind map.  
  


Figure 9.11 The mind map produced by clicking on "Swine Influenza"

Clicking on "Similar Symptoms" in Figure 9.11 produces the following mind map.

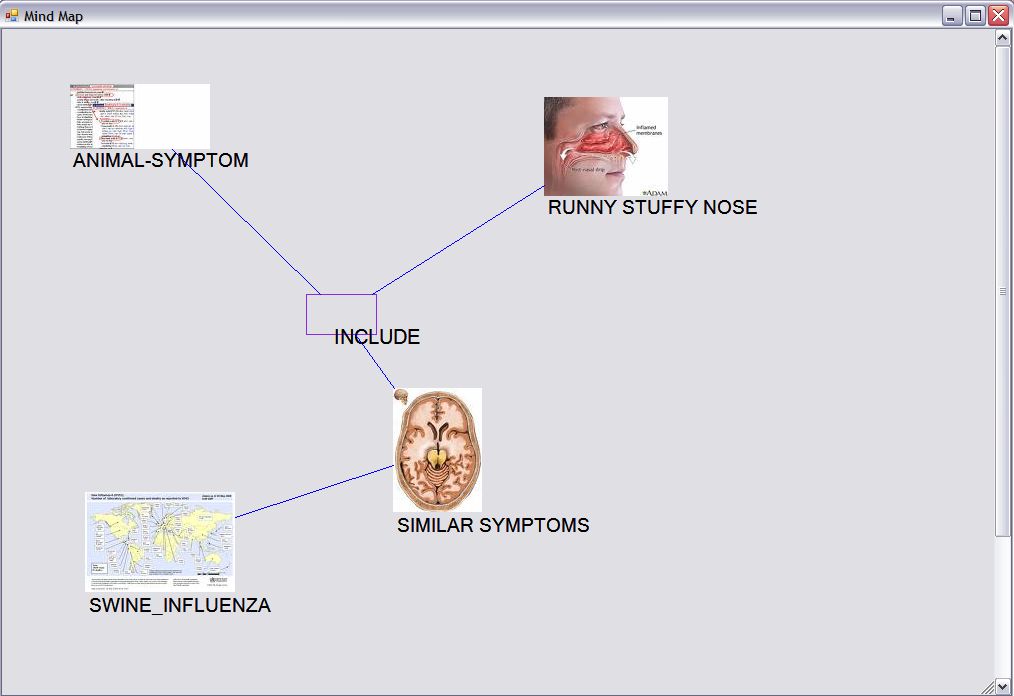


Figure 9.12 The mind map produced by clicking on "Similar Symptoms." Figure.

Clicking on "Animal Symptom" in Figure 9.12 produces the following mind map.

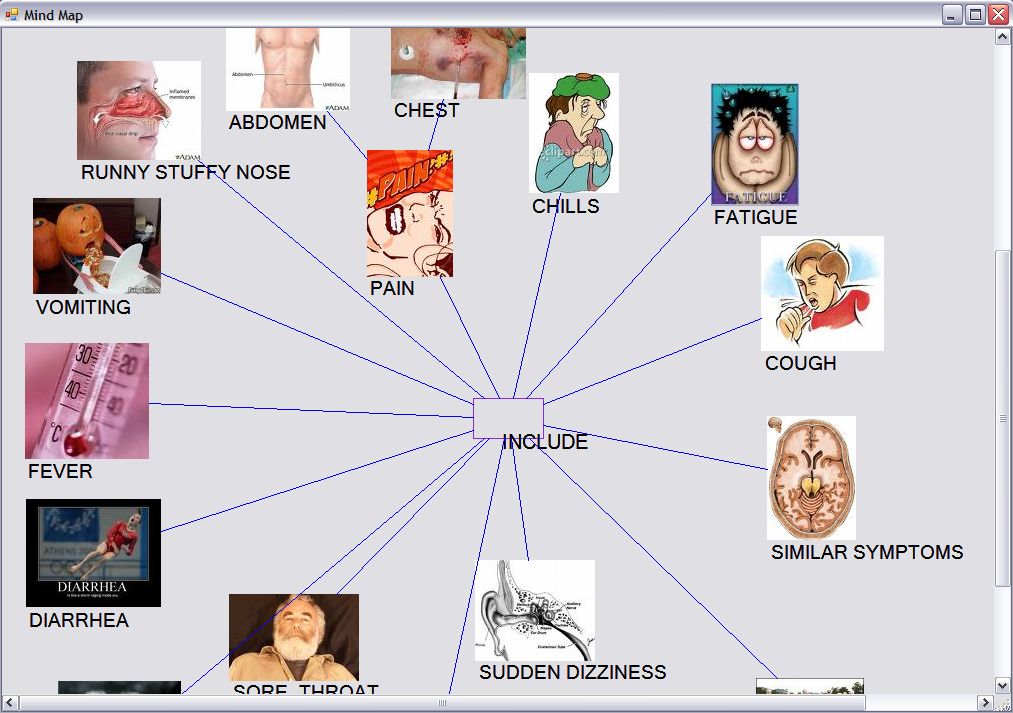


Figure 9.13 The mind map produced by clicking on "Animal Symptoms."

# Evaluation

The time it takes to generate a mind map from input text increases with the increase of the input size; the more text to process the more time it takes. Also, the processing speed and the memory will most probably have an effect.

Figure 9.14 Graph showing the rate of increase of time against the number of words

Conclusion and Future Work

This chapter gives a brief conclusion to what we have reached with our project. The conclusion is then followed by a few future work recommendations that may enrich such work much further within the field of mind map generation.

# Conclusion

The Mind Map Generator, 2nd Edition, is a tool that automatically generates mind maps from plain English text. It is now is capable of processing longer sentences and wider grammatical structures of English than that that it previously did. The output is a multi leveled mind map automatically laid out on the screen.

The Mind Map Generator consists of five main modules:

1. The Frontend (previous work)

It consists of five sub modules:

* Morphological Analysis
* Parsing
* Syntax Analysis
* Discourse Analysis
* Word Sense Disambiguation

1. Text Meaning Representation
2. Multi-level Generation
3. Mind Map Conversion
4. Automatic Layout Allocation

# Future Work

The Mind Map Generator has yet much work to be done to reach its full potential.

1. **Developing an Arabic version of the MMG.**

To generate mind maps automatically for Arabic text.

1. **Semantically Dependant Discourse.**

Assigning each pronoun to the noun which this pronoun refers to depending on the meaning of the sentence and the logic of nature.

1. **Discarding the parser**

Generating text meaning representation module without depending on a parser output of parse trees, consequently accepting any form of English, not necessarily only formal English.

1. **Query based mind map**

Entering a document as an input then querying on a part of this document and producing only the mind map specific with this query.

1. **Generating English text from an input mind map**

To generate English text from an input mind map, the reverse of what this project has been trying to accomplish, so far.

References

1. Mind Map Generator, Ain Shams University, Faculty of Computer & Information Sciences, Computer Science Department, supervised by Prof. Dr. Mustafa Aref and Dr. Mohamed Abdeen
2. Automatic Graph Drawing Algorithms, Susan Sim, [simsuz@turing.utoronto.ca](mailto:simsuz@turing.utoronto.ca), December 17, 1996.
3. K-Means++: The Advantages of Careful Seeding, David Arthur, Stanford University, Supported in part by NDSEG Fellowship, NSF Grant ITR-0331640, and grants from Media-X and SNRC. Sergei Vassilvitskii, Stanford University, Supported in part by NSF Grant ITR-0331640, and grants from Media-X and SNRC.

1. F.J. Brandenberg (editor). *Graph Drawing ‘95, Proceedings of Symposium on Graph*

*Drawing*, Springer Verlag, Berlin, 1996.

1. I.F. Cruz and R. Tamassia. How to visualize a graph: Specification and Algorithms.
2. http://www.cs.brown.edu/people/rt/gdtutorial.html.
3. R. Davidson and D. Harel. Drawing Graphs Nicely Using Simulated Annealing. *ACM*
4. *Transactions on Graphics*, 15(4):301-331, October, 1996.
5. G. Di Battista, P. Eades, R. Tamassia, And I. G. Tollis. Algorithms for Drawing Graphs:

an Annotated Bibliography. June, 1994. Available by anonymous ftp from

ftp.cs.brown.edu:pub/papers/compgeo/.

1. G. Di Battista, H. D Fraysseix, P. Eades, P Rosenstiehl, and R. Tamassia (Eds). *Graph*

*Drawing ‘93, Proceedings of the ALCOM International Workshop on Graph Drawing*

*and Topological Graph Algorithms*. Available by anonymous ftp from

wilma.cs.brown.edu:pub/papers/compgeo/.

1. P. Eades. A Heuristic for Graph Drawing. *Congressus Numerantium*, 42:149-160, 1984.
2. P. Eades and K. Sugiyama. How to Draw a Directed Graph. *Journal of Information*

*Processing*, 13(4):424-437, 1990.

1. M.R. Garey and D.S. Johnson. Crossing Number is NP-Complete. *SIAM Journal of*

*Algebraic and Discrete Methods*, 4(3):312-316, 1983.

1. E. Harley and A.J Bonner. A Flexible Approach to Genome Map Assembly.
2. *Proceedings of the International Symposium on Intelligent Systems for Molecular*

*Biology*, pp. 161-169, 1994.

1. T. Kamada. Visualizing Abstract Objects and Relations: A Constraint-Based Approach,

World Scientific, New Jersey, 1989.

1. M.R. Kramer and J. van Leeuwen. The Complexity of Wire-Routing and Finding

Minimum Area Layouts for Arbitrary VLSI Circuits in “Advances in Computing

Research”, vol. 2, edited by F.P. Preparata, pp. 129-146, JAI Press, Greenwich,

Connecticut, 1984.

1. Z. Miller and J.B. Orlin. NP-Completeness for Minimizing Maximum Edge Length in

Grid Embeddings. *Journal of Algorithms*, 6:10-16, 1985.

1. H.A. Müller, M.A. Orgun, S.R. Tilley and J.S. Uhl. A Reverse-engineering approach to

subsystem structure identification. *Software Maintenance: Research and Practice*,

5:181-204, 1993.

1. E.G. Noik. Dynamic Fisheye Views: Combining Dynamic Queries and Mapping with

Database Views. Doctoral thesis, Department of Computer Science, University of

Toronto, 1996.

1. K. Sugiyama, S. Tagawa, and M. Toda. Methods for Visual Understanding of

Hierarchical System Structures. *IEEE Transactions on Systems, Man and Cybernetics*,

11(2):109-125, 1981.

1. R. Tamassia and I.G. Tollis (editors). *Graph Drawing ‘94, DIMACS International*

*Workshop*, Springer Verlag, Berlin, 1995.

1. W.T. Tutte. Convex Representations of Graphs. *Proceedings of the London*

*Mathematics Society*, 10:304-320, 1960.

1. W.T. Tutte. How to Draw an Graph. *Proceedings of the London Mathematics Society*,

13:743-768, 1963.

1. K. Wagner. Bermerkungen zum Vierfarbenproblem. *Jber. Deutsches Mathematiks*

*Verein*, 46:26-32, 1936.

1. D.B. West. Introduction to Graph Theory. Prentice Hall, Upper Saddle River, New

Jersey, 1996.

1. http://en.wikipedia.org/wiki/Floyd-Warshall\_algorithm
2. <http://duffill.blogs.com/beyond_crayons/2004/11/three_basic_typ.html>
3. <http://www.mapyourmind.com/advantages.htm>
4. <http://www.peterrussell.com/MindMaps/Advantages.php>
5. <http://www.peterrussell.com/MindMaps/Uses.php>
6. <http://www.mind-mapping.org/mindmapping-and-you/basic-introduction-to-mindmapping.html>
7. <http://www.buzanworld.com/userfiles/file/Research%20Papers/Mind%20Mapping%20in%20Executive%20Education%201999.pdf>
8. <http://www.buzanworld.com/Mind_Maps.htm>
9. <http://en.wikipedia.org/wiki/Mind_map>
10. <http://en.wikipedia.org/wiki/Morphology_(linguistics)>
11. <http://en.wikipedia.org/wiki/Natural_language_processing>
12. <http://crl.nmsu.edu/Research/Projects/mikro/htmls/flairs96-htmls/node11.html>
13. <http://crl.nmsu.edu/Research/Projects/mikro/htmls/oldstuff-htmls/case-roles.html>
14. <http://crl.nmsu.edu/Research/Projects/mikro/htmls/relations-htmls/Dom-rel-definitions.html>
15. <http://crl.nmsu.edu/Research/Projects/mikro/htmls/oldstuff-htmls/tmr-intro.html>
16. <http://crl.nmsu.edu/Research/Projects/mikro/htmls/how.to.tmr-htmls/tutor.html>
17. http://crl.nmsu.edu/Research/Projects/mikro/htmls/flairs96-  
    htmls/node2.html#SECTION00020000000000000000
18. <http://en.wikipedia.org/wiki/Ontology_(computer_science)>
19. <http://www-ksl.stanford.edu/kst/what-is-an-ontology.html>
20. <http://protege.stanford.edu/>
21. <http://en.wikipedia.org/wiki/Web_Ontology_Language>